MARK TIMMER

EFFICIENT MODELLING, GENERATION AND ANALYSIS OF MARKOV AUTOMATA

Efficient Modelling, Generation and Analysis of Markov Automata

Mark Timmer

CTIT Ph.D. Thesis Series No. 13-261
Centre for Telematics and Information Technology
University of Twente, The Netherlands
P.O. Box 217 – 7500 AE Enschede

IPA Dissertation Series No. 2013-13
The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).

Netherlands Organisation for Scientific Research
The work in this thesis was supported by the SYRUP project (SYmbolic RedUction of Probabilistic Models), funded by NWO grant 612.063.817.

ISBN 978-90-365-0592-5
ISSN 1381-3617 (CTIT Ph.D. Thesis Series No. 13-261)
Available online at `http://dx.doi.org/10.3990/1.9789036505925`

Typeset with LaTeX
Printed by Ipskamp Drukkers

Cover design by Thomas van den Berg and Mark Timmer
(Image designed originally by Kaj Gardemeister, bought from Dreamstime)

EFFICIENT MODELLING, GENERATION AND ANALYSIS OF MARKOV AUTOMATA

DISSERTATION

to obtain
the degree of doctor at the University of Twente,
on the authority of the rector magnificus,
prof. dr. H. Brinksma,
on account of the decision of the graduation committee,
to be publicly defended
on Friday, September 13th, 2013 at 16:45 o'clock.

by

Mark Timmer

born on 15 April 1984
in Apeldoorn, The Netherlands

This dissertation has been approved by:

Prof. dr. ir. Joost-Pieter Katoen, PDEng (promotor)
Prof. dr. Jaco van de Pol (promotor)
Dr. Mariëlle Stoelinga (co-promotor)

To my parents (in memoriam)

Acknowledgements

"A noble person is mindful and thankful for the favours he receives from others."
Buddha

FIRST, I would like to thank my supervisors Jaco van de Pol, Joost-Pieter Katoen and Mariëlle Stoelinga, who were most influential regarding the contents of this work. Thank you all so much for enabling me to experience the journey that resulted in this thesis. It taught me a lot about computer science as well as myself, and it allowed me to meet people from more countries than I could ever have expected.

Jaco, you were always enthusiastic to talk about technical matters, discussing definitions or proofs and often immediately spotting mistakes or possible difficulties that I may otherwise have missed. Your eye for detail greatly helped to shape and improve this work. As a boss, you gave me the freedom to work on a variety of topics, and allowed me to participate in the "promovendi voor de klas" project. This impacted my life significantly, enabling me to pursue a career in teaching. You recently even gave a guest lecture at the school where I teach, explaining twelve-year-olds about the fascinating aspects of model checking and experiencing how rewarding secondary school teaching is. I can never thank you enough for being a decisive factor in making it possible that I can now experience this joy every day.

Joost-Pieter, even though Aachen is not around the corner, we quite often managed to discuss my work. I really enjoyed the visits to Aachen, allowing me to get to know your other PhD students as well. Thank you for the many motivating talks and emails, and the extremely precise feedback on my papers and this book—you always managed to suggest several interesting and relevant concepts to mention or include in my work, checked several proofs very thoroughly and never failed to notice a missing closing parenthesis. When I was in doubt which direction to go or to which conference to submit next, I could always rely on your confidence and great advice.

Mariëlle, as my daily supervisor you were the one who has taught me the most. While working on my MSc thesis you already spent many hours educating me in the fine art of scientific writing. I learned to be concise and use short and simple sentences where possible—although I do still appreciate some more complicated structures. I learned to use 'may' instead of 'might' more often, and to stop using 'would' in conditional statements that require a *second conditional* or *third conditional* grammatical construct instead. You taught me how to structure an explanation, how to write an introduction and how to define concepts as simple

as possible. Your quest for clean theory sometimes made me slightly desperate, but also greatly helped me to improve my work. When I knew that a certain concept could be explained a bit better but I had not forced myself to do so, I could always count on you to point this out. In the end, of course, I was happy that you did. Also on a more personal level, we had many interesting talks. You always know more gossip than I do and often have several suggestions for interesting books, my favorite being “Eats, shoots and leaves”. I also like to thank you for allowing me to co-organise the Dutch testing day and one of the ROCKS meetings—a lot of fun!

Regarding the work in this thesis, in addition to my supervisors, I want to thank several other people. First of all, a big thanks to Henri Hansen and Arnd Hartmanns for collaborating with me on the work that resulted in Chapters 7 and 8. Luis María Ferrer Fioriti, thanks for your help in analysing the behaviour of the partial order check on the case studies discussed in Chapter 8. I also want to thank Erik de Vink and Michel Reniers for their many helpful comments on a draft of one of the papers on which Chapter 4 is based, as well as Pedro d’Argenio for his useful insights in stochastic process algebras. A big thanks to Dave Parker from Oxford University for useful discussions about the functionality of my tool and several of the underlying theoretical ideas, and Axel Belinfante for implementing a web-based interface. I thank Jan Friso Groote for his specification of the handshake register, upon which one of the case studies in Chapter 9 is based. Furthermore, I thank Michael Weber for fruitful discussions about Hesselink’s protocol. Stefan Blom, thank you so much for your many helpful suggestions regarding several aspects of my work. Dennis Guck, thanks for all your invaluable help regarding the case studies and tool support.

I would like to thank my thesis committee for their extensive efforts, reading this book and providing numerous points of improvement. Boudewijn Haverkort, Holger Hermanns, Jan Friso Groote, Richard Boucherie, Wan Fokkink, thank you for all your hard work during the summer!

An important part of my PhD experience consisted of my collaboration with the people from the research group I was part of: Formal Methods & Tools (FMT). Over the years, I had many colleagues that were interesting coworkers and good friends as well. For approximately 4.5 years I shared office 5078 with Eduardo Zambon. We had a perfect atmosphere for working and having some fun as well, mostly working hard but helping each other when needed and having a laugh every now and then. Eduardo, you showed me that the final months of writing a PhD thesis can be hell—but also that it is not impossible to make it through and that the result is very rewarding. I replicated this process, going through hell as well though with the advantage of having seen that indeed everything can be completed as long as we continue gradually. Thanks for all the good times! And, sorry again for using your desk as additional storage space for all my stuff when you were on holiday or a conference visit. After Eduardo graduated, Paul Bonsma took his place (and hence my storage space). Paul, I enjoyed your company. Thanks for putting up with me in these last few months when I was stressing to finish the thesis and therefore may not have been the most pleasant office mate.

Theo Ruys, your passion for teaching really impressed me, and your (sometimes hard to interpret) sarcasm and cynicism often resulted in a (sometimes somewhat delayed) smile on my face. I was in doubt for quite a while about whether or not to pursue the teaching degree in parallel to my PhD work. You quitting your job at FMT to study for primary school teaching was precisely the trigger I needed to decide (even though in the end you did not pursue this—I am still expecting at least some teaching degree from you in the coming years), thank you so much for that! Thanks as well for being quite perplexed when at some point I had neglected my right to vote for the European Parliament—I voted for every election since then.

Arend Rensink, thanks for all your enthusiasm organising Floor Five Film marathons, dutifully buying crisps for BOCOM (Floor Five's Friday afternoon get-together), and for organising a wonderful barbecue every year—allowing me to be lazy last time and not bring any salad in exchange for this acknowledgement. Elise Rensink, you are always great company at the barbecues and one of the coolest women I know!

Thanks to Rom Langerak for being a great office neighbour, and for stressing even more on your educational self-evaluation than I did on my thesis these last months (that made me feel slightly better). Marieke Huisman, thank you for having us over several times, allowing us to practice babysitting. Also, thanks for always being so relaxed, putting things in perspective, and for persistently trying to get our foreign colleagues to speak Dutch. Hajo Broersma, thanks for your stories about mathematics and for lending me a great calendar with puzzles that I may use in secondary school.

Axel Belinfante, thank you so much for providing the puptol framework! It majorly improved the visibility of my work, impressed quite some colleagues at conferences and even helped me to more easily experiment with my own tool. Also, thanks for all the great collaborations regarding the Testing Techniques course over the years and the organisation of the Dutch testing day! Michael Weber, I was amazed by your positivity when some bad things happened. I was also amazed by your resistance to Marieke's attempts to try to have you speak Dutch, even though you secretly can do this perfectly fine! Stefan Blom, thanks for being our most reliable BOCOM supplier, buying beer at the lowest price and introducing the cans of soft drinks (that have become very popular over the last year!). Also, thanks for always making time to give great tips when I had a confluence-related problem, and for still being nice to me even though I had little time to assist you while you were working on a great embedding of my tool SCOOP in the LTSmin toolset (which, by the way, still deserves a better name in my opinion—Jaco, you managed to rename FMG to FMT, so this should be possible too!).

Maarten de Mol, even though (or maybe even because) we are so different in many ways, I always had a great time having coffee with you in the rappa. It was funny seeing you walk circles to come up with new ideas, or seeing you passionate about functional programming. I was impressed by your work ethic and your persistence in travelling to work all the way from Westervoort every day. Thank you for the many lovely conversations we had! Alfons Laarman, first of all thanks for always being in the office so late that I never had to feel bad if I was sometimes late as well—I could always count on you being even later! Also, thanks for the great company you and Laura Grana-Suarez were during our Western USA within a week trip! I actually had not expected you to survive and/or enjoy our minute-to-minute planning on Mark & Thijs speed (or, as Laura phrased it, the German schedule). I had an amazing time, thanks for being there to share this experience!

Dennis Guck, first of all, thank you so much for developing algorithms for Markov automata and for implementing IMCA! I remember being absolutely thrilled when Joost-Pieter told me about your accomplishments; they really give purpose to my work that would otherwise not have been there. Florian Arnold, Dennis, thanks for an amazing Rome experience at ETAPS this year! I seldom had so much fun at a karaoke bar. Florian, once more my apologies for persuading you to stay up so long that the next day you actually missed your plane! I felt a bit bad the next day, but on the other hand this does make a great story (for me as well as for you, I hope).

Tri Minh Ngo, thanks for very persistently forcing me to take the stairs instead of the elevator and constantly reminding me that I should exercise more! Of course, you were completely right. Actually, I recently even took the stairs while you were in Vietnam so that I could proudly send you an email about this! (I did forget to send this email though.) Also, thanks for having your luggage get lost while travelling to the Netherlands for your job interview and then presenting your work in shorts and a rather crappy T-shirt—this story always makes me feel better when my luggage gets lost at the airport; things can always be worse!

Stefano Schivo, thanks for motivating several people (including me sometimes) to go running every week and get some exercise; we really need it. Marina Zaharieva-Stojanovski, you are one of the friendliest people I have ever met—thanks for always being so involved and interested in other people's lives. Also, I am very happy to have you as a paranymph! Gijs Kant, thanks for your omnipresent enthusiasm to go out and have a drink at conferences, for your great participation in making songs for colleagues, and for sometimes organising nice events in the lunch breaks. Ed Brinksma, thanks for taking some time out of your extremely busy schedule to work on a nice paper on model-based testing together with Mariëlle and me. Afshin Amighi, Lesley Wevers, Mohsin Danish, Steven te Brinke, Tom van Dijk, Waheed Ahmad, Wojciech Mostowski, you are all great colleagues! Amir Ghamarian, Jeroen Ketema, I was sad to see you leave, thanks for the great times we had. Amir, I'm still counting on you to drive a camel some day.

Joke Lammerink, Jeanette Rebel-de Boer, thanks for your administrative support during the last five year, but even more for the many long and interesting talks we had. If I had something I wanted to talk to somebody about, I was always welcome in your offices.

I also owe a great deal to the lovely people of DACS. Aiko Pras and Pieter-Tjerk de Boer, thank you so much for your great supervision of my BSc thesis in 2005! You really gave me confidence to seriously consider pursuing a PhD. Without you, this thesis may never have been written! I also really want to thank you

for encouraging me to write a paper on my BSc work and helping me a lot by providing feedback and writing parts of that paper. Boudewijn, a big thanks to you as well, for allowing me to present the paper at the E2EMON workshop in Vancouver, Canada—additionally, thank you so much for informing me about the possibility to take a return flight from a different airport! This enabled me to make a memorable trip from Vancouver to San Francisco and Los Angeles. Anne Remke, thanks for our many great talks and for offering me to work with you and Rom next year!

Dave Parker, Marta Kwiatkowska, Joel Ouaknine, thank you so much for receiving me at Oxford University ComLab for two months in the autumn of 2010. Marta, once more my apologies for spilling my soup all over the chair while walking back to the lunch table in Trinity College's professor's lounge. As it all looked so fancy I was so nervously trying to avoid doing anything stupid that of course my clumsiness caused me to make a big mess. The change of scenery due to my stay in Oxford inspired me to majorly improve my tool SCOOP, and it taught me a lot about Apex and PRISM (no dear reader, not the NSA surveillance programme). Most importantly, it got me in contact with so many nice people. Björn Wachter, Christian Dehnert, Hristina Palikareva, I had the best time on our trip to Stonehenge, Salisbury, Bath and some castle just over the border of Wales. Christian, thanks for being great company in Oxford, for joining me to Windsor Castle and Cambridge, and for showing me around in Aachen. Vojtech Forejt, Stefan Kiefer, you also quickly became good friend during my visit. Together with the rest of the gang, I greatly enjoyed our visits to evensongs, the opera, the gamelan concert, the Jan Tiersen concert, and a large variety of pubs and restaurants.

Henri Hansen, thanks for accompanying Christian and me on a great trip to Stratford-upon-Avon, and for making every lunch break a delight. Also, many thanks for our wonderful collaborations, resulting in a journal paper that makes me proud and that is presented in Chapter 7 of this thesis. Our collaboration was very fruitful; we both contributed ideas and meticulously checked and if needed improved the other's input. I always felt that I could opt every idea that came to mind, surely receiving constructive criticism. You still have many great ideas on the topics that we worked on, and I hope that we may still publish some nice results together. I also still have to visit you in Finland someday, and would really like to do so some day!

Alexandru Mereacre, Arpit Sharma, Christian Dehnert, Erika Ábrahám, Friedrich Gretz, Falak Sher, Henrik Bohnenkamp, Martin Neuhäußer, Sabrina von Styp, Souy Chakraborty, Tingting Han, Viet Yen Nguyen, thanks for making me feel right at home in Aachen. You were great to be around during my research visits and the Aachen Concurrency and Dependability week. Tingting, I thought we were a great Aladdin and Jasmin on SingStar!

I would like to thank all the students that I supervised during my PhD. Vincent Bloemen, you were a pleasure to work with, and I am very proud that you managed to work on several quite difficult concepts as part of your BSc thesis. Martijn Adolfsen, you worked hard on an interesting topic and obtained several relevant results. No wonder that the company you did the research at wanted to keep you! Elodie Venezia, you were one of the best students in the Testing Techniques course. Your research project on coverage measures demonstrated great perseverance! Gerjan Stokkink, I was very impressed by the amount of work you managed to complete. We don't often see so many scientific results from one MSc project. It was great fun too, exploring the bad neighbourhoods of Tallinn together! Rob Bamberg, thanks for choosing me for your MSc thesis on a topic very related to my work; I enjoyed it a lot. Ferry Olthuis, at the moment of writing you are still working on your MSc thesis. Good work so far! I was very honoured to hear that you wanted to do your thesis on a project further investigating some of my work. You are a pleasure to work with.

During the last few years I visited several conferences, sometimes at far-away destinations. It is infeasible to thank all the people that made these experiences as memorable as they were, but I do like to mention a few. In York, England (ETAPS 2009), I had many nice dinners and a remarkable Ghost Tour, mostly with the Aachen people from Joost-Pieter's group. Thanks for a great initiation to enjoying conferences, guys! In Tianjin, China (TASE 2009), I had an awesome time sightseeing the city together with Richard Banach and Moritz Martens, thank you for that! This conference also resulted in Thijs and me travelling through China for three weeks, showing us how great far-away holidays can be and inspiring us to have many more afterwards. I owe this to a couple of FOSSACS 2009 reviewers; thank you for rejecting my paper so that I could resubmit to TASE, travel to China and have all these amazing experiences!

Macao (ATVA 2009) was also fairly spectacular, bungee jumping the Macao Tower (233 meters) together with Christian Dax. A big thanks to Yael Meller for making pictures during the bungee jump and for teaching me a lot about Israel. And a big apology to Thijs for almost giving you a heart attack upon seeing the video of this jump. ETAPS 2010 brought me to Paphos, Cyprus, where I had a great time with Miguel Andres, Trajce Dimkov and Petur Olsen. Besides drinking cocktails and enjoying a swim in the ocean or the swimming pool, we spent a free afternoon discovering the underwater world through an introduction dive. Thanks for sharing this with me! Rather impulsively Trajce and me even took a road trip to the north of Denmark, visiting Petur during the Aalborg carnival. Great trip, thanks guys for the awesome weekend!

The summer of 2010 took me to Braga, Portugal for ACSD. Anton Wijs, thanks for accompanying me during the Sao Joao festival, which to our great amusement mainly seemed to consist of people smashing each other on the heads with plastic hammers (*martelinhos*) and leek. A big thanks to all the Braga inhabitants for indeed allowing us to smash them on their heads; that's good stuff! I also like to thank Matthias Raffelsieper for spending a great day in Porto with me.

ETAPS 2011 in Saarbrücken, Germany was nice, but I enjoyed the 2012 destination even more: Tallinn, Estonia. As mentioned before, it encompassed an interesting walking tour with Gerjan, but also a shamefully funny evening in a karaoke bar with Arend, Gerjan, Gijs and Sander Bruggink (video material is available upon request). Also, thanks to Arend, Maarten, Marina and Anton for joining me to visit Helsinki, Finland during this week. Although Marina observed that Helsinki is a place where you would rather be found dead than alive (I could see her point), I really enjoyed the trip. Later that year, I spent a wonderful weekend in York with James Williams and Chris Poskitt for mental preparation for CONCUR 2012, taking place in Newcastle, England. Daniel Gebler and Michel Reniers, thanks for joining me to see the (rather unimpressive, as it turned out) Hadrian's wall. Daniel, thanks as well for inviting me to give a talk at your group in Amsterdam!

This year's ETAPS in Rome, Italy was a pleasure as well. Although I was already stressing a bit about finishing this thesis and I could only be there briefly due to teaching responsibilities, the karaoke bar visit with Dennis and Florian made me forget all my worries for a while—fortunately, I did manage to complete my slides the next day. Thanks to Arnd Hartmanns as well for joining me for sightseeing through Rome. Mieke Massink, Erik de Vink, thanks for always providing great companionship at QAPL events.

Even more fun than conferences are summer schools. Katharina Spies, Silke Müller, thank you for organising the best summer school there is, the Marktoberdorf summer school! I had the most amazing time there in the summer of 2010. Hristina Mihajloska, Magdalena Kostoska, you already became good friends of mine there before the first night was over! Yuriy Solodkyy, thanks for teaching us many great salsa moves; this always brought about a perfect atmosphere. Chris Poskitt (Harry!), David Williams, James Sharp, James Williams (Ron!), Luke Herbert, Tuomas Kuismin, go Team GB! You all were so funny and a blast to be around. I had a great time during our visits to Neuschwanstein Castle and Munich, and it was awesome when you came to see me during my Oxford visit (unfortunately without Tuomas), and when we met up in Amsterdam after David moved there. James Sharp, you are one of the best Disney song singers I know! I think we annoyed most of Amsterdam and all of the professors in the bus to the hike near Marktoberdorf, a good accomplishment. Luke, you are one of the strangest persons I have ever met (in a good way)! Aws Albarghouthi, Ken Madlener, it was also very nice to meet you in Marktoberdorf and at other occasions.

In the summer of 2011 I spent a wonderful week in the hilly town of Bertinoro, Italy, for the SFM summer school. Marco Bernardo, thanks for organising this event! Also, a big thanks to my room mate Nuno Oliveira, and to Peter Drabik, Jaroslav Keznikl, Amel Bennaceur, Gianina Ganceanu, Bojana Bislimovska, Pankesh Patel, Sara Hachem, Sergio Di Sebenico, Imen Ben Hafaiedh Marzouk for several great dinners, long evening talks and a great visit to the beach of Rimini. I had loads of fun! Christel Baier, my (kind of late) apologies for leaving and re-entering the room several times during your lectures. After having had a nice talk with you in the bus from the airport, I felt a bit bad about it. There was a good reason, though; I was negotiating the price of the house we currently own, and we managed to reduce it by approximately 10% during your talk!

Maybe even more fun than summer schools, the IPA (Instituut voor Programmatuurkunde en Algoritmiek) Spring Days and Autumn Days were among the highlights of each year. Thanks to Tijn Borghuis, Michel Reniers and

Tim Willemse for subsequently organising these wonderful events together with Meivan Cheng! We learned a lot during the days on various topics, but also (most importantly?) enjoyed the socialising aspects that took place during the evenings. I made many great friends during these events and we had so much fun, even resulting in two IPAZoP (IPA Zonder Praatjes) weekends organised by ourselves (big thanks to Zé Pedro Magalhães!). A huge thanks to Alexandra Silva, Atze van der Ploeg, Carst Tankink, Cynthia Kop, David Costa, Faranak Heidarian, Felienne Hermans, Frank Stappers, Frank Takes, Jeroen Keiren, Joost Winter, Marijn Schraagen, Michiel Helvesteijn, Paul van Tilburg, Zé Pedro Magalhães, Sjoerd Cranen, Stephanie Kemper, Tijs van der Storm and many others for making these events so much more than just work.

Besides the conferences, summer schools and IPA events, I also visited a large number of ROCKS and QUASIMODO meetings. At these events I met so many nice people, among which some of the ones already mentioned above. A special thanks to Arnd Hartmanns, who gave an interesting talk at one particular ROCKS meeting that resulted in a fruitful collaboration. We are both equally perfectionistic (Arnd maybe even slightly more than I am), which actually worked out really well; it got us to the NASA Ames Research Center in Moffett Field, California, and resulted in Chapter 8 of this thesis. Arnd, thank you for the collaboration, for taking me to see Völklinger Hütte, and for your great scheduling skills and companionship regarding our Western USA within a week trip!

Nico van Diepen, you were the first person to inform me about the “Promovendi voor de klas” project. Thank you for thinking about me and being the initial step in my journey towards the most interesting job I can imagine! Nellie Verhoef, you are a great inspiration regarding mathematics teaching and a lovely person! Thank you for always believing in me and my teaching skills, allowing me to collaborate with you on several interesting projects and papers, and for inviting me to the Community of Learners for mathematics teachers! Bert Booltink, Daan van Smaalen, Fokke Hoeksema, Gerard Jeurnink, Jan Keemink, John Heijmans, Nico Alink, Roelf Haverkamp, Ronnie Koolenbrander, Tom Coenen, thanks for making this such an inspiring experience! Petra Hendrikse, thank you for the numerous tips and advice and your guidance during my internship at a secondary school. Henri Ruizenaar, thanks for lending me one of your groups of students at the Stedelijk Lyceum Kottenpark for one chapter to allow me to perform the research project that resulted in several papers, workshops and a nomination for the OnderwijsTopTalentPrijs. Without you, this would not have been possible!

Marita Groote Schaarsberg, you immediately made me feel at home at the Carmel College Salland. I really admire your enthusiasm, passion and dedication for both your work and your personal life. When I was busy, you were always willing to help and support me even though you were busy yourself just as well. I hope we can keep working together for a long time, and that you will enjoy and excell at your new position as much as at teaching mathematics. I also like to express my gratitude towards the rest of the mathematics section. Gerard Tenhagen, Gerrit van Wijk, Harrie Thoben, Henk de Waal, Henk Langbroek, Henri van der Meijden, Jan Swart, Karin Hafkamp, Marcel Hagen, Marian Velding, Marita Groote Schaarsberg, Marjan Schutmaat, Reina Voogd, I could not have hoped for a better group of colleagues. You all value mathematics just as much as I do, provide an amazing setting to work in and all are such nice people. Thank you for receiving me in such a positive way!

Jacqueline Maatman, Ingrid Hegeman, you both fought hard to be able to hire me in a time of budget cuts and downsizing. I am very happy about the positive outcome of your efforts, and want to thank you for your trust in me. I also like to thank team 3, headed by Jacqueline, for the way they made me feel at home right away. Amanda Mulder, Dieuwertje Kuppens, Gemma de Breet, Gert Katgert, Henk Löevering, Jan van Zon, Jeroen Koene, Jolanthe Beenders, Loes Luurs, Louis Bakkenes, Hans Roeland, Mark van Dasler, Marjolein Gerritsen, Mirjam Vrijma, René Schiphorst, Sander Alferink, Siebrich Siemens, Reina Voogd, Yttje Sipma, it has been a pleasure! I am very happy that I can work with you in a team for another year! Mark, it was great fun developing the curriculum for our new course on technologies with you. Our ideas often complemented each other, and your relaxed and seemingly carefree attitude helped me to put things in perspective. Thanks! I would also like to thank Mark Schrijver, Terrence Bos and Harriët Lemstra-Dijk for their collaborations in the excellence working group, which helped shape the Technology course in many ways. Nicola Buthker, thanks so much for coaching me for the last year. Dieuwertje, Marjolein, when you were in our team room, it was always hard to focus since we had such good times. Thanks for making work so much fun! Jolanthe, thanks for introducing me to the world of mentoring! I am really looking forward to sharing A1A3 with you for the coming year.

A big thanks to my friends, and my apologies for having so little time this year; I really hope to make this up to you! Andre Foeken, Liang Hiah, Wilco Hendriksen, Hester Bruikman, George Onderdijk, Veronique Wendt, Thomas van den Berg, Marit Hoekman, Keith Davelaar, Randy Klaassen, Frank Halfwerk, Martijn Schouwstra, Auke Been, Laura Vos, Melissa Martos, Lina Baranowski, Johannes van Wijngaarden, Jasmien de Vette, Maarten Eykelhoff, Fenna Janssen, Mark Kruidenier, Marieke van Amstel, Luuk Pasman, Emilie Klaver, Vincent Kroeze, Koen Blom, Carmen de Schutter, Erik Slomp, Michel Jansen, Remko Nolten, Wim Bos, you all are amazing people! Lina, I as so happy that you introduced me to a mindfulness workshop during the last months of writing this thesis; that really helped me to relax! Thomas, a big thanks for designing the cover of this thesis. You contributed to the part of this book that most people will (only?) see.

Thanks to all the people from Stichting Neverland, for organising the most amazing summer camps for children! Sophie van Baalen, Mieke Boon, Febe van der Zwan, Tianne Numan, Ilona Meij, Teska Numan, Theuntje Steemers, thank you for perfectly arranging a big change in the organisation this year without me having to do much while I was busy finishing this thesis.

I would like to thank my family for supporting me and always being there for me. Mom, dad, you always told me that talents are to be used to their fullest. You taught me a good work ethic that eventually culminated in the accomplishment represented by this thesis. You also often tried to save me from myself a bit when

I was working too hard, a lesson I am still trying to apply (although regularly failing at this). Most importantly, you provided me with the best childhood anyone can ask for. We did a lot of things together, watching movies, going to the zoo, theme parks, the swimming pool, or to France. You always put my needs before yours, and I cannot thank you enough for being the best parents I can imagine. I know you would have been very proud of me if you could have witnessed me finishing my PhD.

Grandma, I cannot think of a better way to spend my Monday evenings than to have dinner with you and watch Lingo together. You always know exactly what to say, and I am so happy that you are still able to watch me defend my thesis. Thank you for always taking an interest in my life and for being an amazing grandma! Grandpa (in memoriam), Edward, Arnold, Gerrie, Bas, Eva, Marloes, Emiliano, Carolien, Marc, Sander, Wies, Jan, Beppie, Wim, Plony, Annemarie, Ralf, Stado, Marjolein, Peter, Jacques (in memoriam), Hans, Gerien, Sander, Emy, Hugo, David, Noortje, Edwin, Bram, you all supported me in one way or another to make me survive these last five years, thank you all for being there!

Thijs, I saved the best for last. Thank you for putting up with me over the last months, when I was often busy and cranky. Thanks for being busy yourself, finishing your MSc in Pedagogical Sciences parallel to your work, which made me feel less guilty about not having so much time myself. Thanks for always remembering everything that I forget, for doing many chores that I hate, for making me buy new clothes when the old ones have holes in them, for making our home a lovely place to live in, for being able to give me so much great advice for helping children with learning disabilities at school, and for enjoying the same type of holidays that I do (labelled by Marcel Hagen as "plannen en rennen"—plan and run). Thanks for sometimes being just as strange as I am, and for enjoying a well-placed semicolon just as much as I do; it just makes a sentence so much prettier! Thanks for knowing what to do when I don't anymore, for often knowing me better than I do myself, and for always being there for me!

Formalities

In addition to all the personal thanks above, I would also like to thank the Nederlandse Organisatie voor Wetenschappelijk Onderzoek (NWO) for supporting my work by means of funding provided by the SYRUP project (Symbolic RedUction of Probabilistic Models); this project paid for the majority of my salary. The first year was funded by the STOP project, paid from our university budget. Thank you for this, University of Twente! Many of my travels were partly funded by the DFG/NWO Bilateral Research Programme ROCKS (Dn 63-257), and by the EU FP7-ICT-2007-1 project QUASIMODO (grant 214755). Many thanks to Jaco and Joost-Pieter for arranging the STOP and SYRUP projects, for Mariëlle for being involved in the acquisition of the ROCKS project, and for a whole bunch of people for setting up QUASIMODO.

A final thanks to all the Dutch tax payers for actually (probably unknowingly) providing the money that is distributed by these agencies!

Abstract

QUANTITATIVE model checking is concerned with the verification of both quantitative and qualitative properties over models incorporating quantitative information. Increases in expressivity of the models involved allow more types of systems to be analysed, but also raise the difficulty of their efficient analysis.

Three years ago, the Markov automaton (MA) was introduced as a generalisation of probabilistic automata and interactive Markov chains, supporting nondeterminism, discrete probabilistic choice as well as stochastic timing (Markovian rates). Later, the tool IMCA was developed to compute time-bounded reachability probabilities, expected times and long-run averages for sets of goal states within an MA. However, an efficient formalism for modelling and generating MAs was still lacking. Additionally, the omnipresent state space explosion also threatened the analysability of these models. This thesis solves the first problem and contributes significantly to the solution of the second.

First, we introduce the process-algebraic language MAPA for modelling MAs. It incorporates the use of static as well as dynamic data (such as lists), allowing systems to be modelled efficiently. A transformation of MAPA specifications to a restricted part of the language—enabled through an encoding of Markovian rates in action—allows for easy parallel composition, state space generation and syntactic optimisations (also known as reduction techniques).

Second, we introduce five reduction techniques for MAPA specifications: constant elimination, expression simplification, summation elimination, dead variable reduction and confluence reduction. The first three aim to speed up state space generation by simplifying the specification, while the last two aim to speed up analysis by reductions in the size of the state space. Dead variable reduction resets data variables the moment their value becomes irrelevant, while confluence reduction detects and resolves spurious nondeterminism often arising in the presence of loosely coupled parallel components. Since MAs generalise labelled transition systems, discrete-time Markov chains, continuous-time Markov chains, probabilistic automata and interactive Markov chains, our techniques and results are also applicable to all these subclasses.

Third, we thoroughly compare confluence reduction to the ample set variant of partial order reduction. Since partial order reduction has not yet been defined for MAs, we restrict both to the context of probabilistic automata. We precisely pinpoint the differences between the two methods on a theoretical level, resolving the long-standing uncertainty about the relation between these two concepts: when preserving branching-time properties, confluence reduction strictly subsumes partial order reduction and hence is slightly more powerful. Also, we compare the techniques in the practical setting of statistical model checking, demonstrating that the additional potential of confluence indeed may provide larger reductions (even compared to a variant of the ample set method that only preserves linear-time properties).

We developed a tool called SCOOP, which contains all our techniques and is able to export to the IMCA tool. Together, these tools for the first time allow the analysis of MAs. Case studies on a handshake register, a leader election protocol, a polling system and a processor grid demonstrate the large variety of systems that can be modelled using MAPA. Experiments additionally show significant reductions by all our techniques, sometimes reducing state spaces to less than a percent of their original size. Moreover, our results enable us to provide guidelines that indicate for each technique the aspects of case studies that predict large reductions.

In the end, MAPA indeed enables us to efficiently specify systems incorporating nondeterminism, discrete probabilistic choice and stochastic timing. It also allows several advanced reduction techniques to be applied rather easily, leading us to define a variety of such techniques. Our comparison of confluence reduction and partial order reduction provides several novel insights in their relation. Also, experiments show that our techniques greatly reduce the impact of the state space explosion: a major step forward in efficient quantitative verification.

Table of Contents

CHAPTER 1

Introduction

"*Joy in looking and comprehending is nature's most beautiful gift.*"

Albert Einstein

OUR society heavily depends on computer systems. Although some people associate these mainly with the desktop computer in their office, computers are used much more ubiquitously. They allow us to watch digital television, call a friend, play games on our consoles, listen to music on our MP3 players and record our favorite movies on DVD. Embedded computer systems can even be found in our microwaves, washing machines, dishwashers and thermostats. Failure of any of such systems would be inconvenient.

Computers are ubiquitous in our financial infrastructure, libraries, and data storage centers—unavailability may have a severe impact on our economy. Maybe even more importantly, computer systems are of vital importance for our transport infrastructure, controlling cars, airplanes, trains, railway crossings and space shuttles. Also, they are present in medical equipment such defibrillators, CT scanners and radiation devices. Failure of any of such systems could very well be life-threatening. Erroneous behaviour by systems operating nuclear power plants may even result in a number of casualties we would rather not imagine.

1.1 Formal methods

"*Software engineers want to be real engineers.*
Real engineers use mathematics.
Formal methods are the mathematics of software engineering.
Therefore, software engineers should use formal methods."

Michael Holloway [BH06]

The omnipresence of computer systems and the accompanying increasing danger of their failure clearly necessitates methods to verify their correctness: we want to be sure that they are *dependable.* A wide variety of techniques can and should be applied to achieve this goal, and due to the complexity and importance of hardware and software we strongly advocate to include the use of *formal methods*: mathematical techniques for system specification and analysis. Former member of the NASA formal methods team Michael Holloway justifies the use of formal methods in an interesting way, as cited above. Recent work at Philips

Healthcare even indicated a possible tenfold reduction in the number of errors and a threefold increase of productivity in software development when using formal techniques [Osa12], illustrating their strength.

Traditionally, formal methods only dealt with *qualitative* aspects of behaviour, verifying for instance that a certain undesirable event (e.g., a buffer overflowing) can never occur, or that a certain desirable event (e.g., a message arriving) is guaranteed to eventually occur. Often, these questions are answered in the presence of *nondeterminism*: unquantified freedom for a system to choose from a set of possible alternative behaviours.

More recently, the focus shifted towards *quantitative* aspects of behaviour, verifying for instance that the *probability* of an undesirable event occurring within a certain amount of *time* is below a given threshold. This asks for more expressive models, that in addition to (1) nondeterminism are also able to model (2) discrete probabilistic behaviour as well as (3) continuous (stochastic) timing. The *Markov automaton* [EHZ10b, EHZ10a] was recently introduced to model precisely those three dimensions.

1.1.1 Formal methods in the development process

The field of formal methods is based on the idea that quality is improved by means of thoroughness through formalisation (i.e., mathematisation). Hence, preferably, formal methods are applied to the specification, testing as well as the verification of hardware and software systems [WLBF09, BH06, ABW10, SSBM11]. We briefly discuss the application of formal methods in these three stages, before zooming in on our subfield within verification in the next section. For all applications of formal methods, tool support is essential—formal methods should be (and are more and more) integrated in model-driven engineering processes [BCP12, BCK$^+$11].

Formal methods for system specification. Software engineers may use modelling languages with *formal semantics* (for instance, Z [ASM80], SDL [FO94] or mCRL2 [CGK$^+$13]) to specify parts of a system that is to be developed. One advantage of using formal methods during this stage in the software engineering process is that formalisation forces us to be precise, thereby hopefully reducing the number of mistakes. Additionally, some languages allow for the *automatic generation* of (parts of) an implementation that satisfies the formalised specification. Finally, a formal specification allows for easier and more thorough testing methods as well as formal verification, as discussed below.

Formal methods for system testing. Once a formal model of a system has been developed, it can be used for *model-based testing* [TBS11]: evaluating the behaviour of a system by means of a large number of executions. Test tools such as TGV [JJ05] and JTorX [Bel10] are able to automatically generate and run many test cases and evaluate the correctness of an implementation in accordance to the formal model of the specification.

Formal methods for system verification. Although testing is applied often, Dijkstra already stated many years ago that it can only be used to show the presence of bugs, but never to show their absence [Dij70]. Hence, especially for mission-critical and safety-critical systems this may not yield sufficient confidence. *Formal verification* of its specification can then be used as an additional technique to check for any remaining imperfections to improve our trust in a system.

Formal verification can roughly be categorised into two main categories: theorem proving and model checking. The field of theorem proving is mostly built on the work of Hoare [Hoa69], who proposed to use preconditions and postconditions to reason about the correctness of a program. Although being applicable to infinite-state systems, an important disadvantage is that theorem proving can only partly be automated, resulting in the fact that theorem provers (such as PVS [ORR$^+$96] and Coq [Ber08]) are often called *interactive theorem provers* or *proof assistants.* The user has to provide the structure of the proof, while the theorem prover assists by validating all steps and possibly automatically completing easy parts of the proof [Duf91, KM04]. We discuss the field of model checking in more detail in the next section.

This thesis focuses on *formal verification* of *quantitative behaviour* by means of *model checking.*

1.2 Model checking

"*Model checking algorithms prior to submitting them for publication should become the norm.*"

Leslie Lamport [Lam06]

"*Many notions of models in computer science provide quantitative information, or uncertainties, which necessitate a quantitative model checking paradigm.*"

Michael Huth and Marta Kwiatkowska [HK97]

This thesis is positioned in the field of model checking, a topic that started with two seminal papers, written independently by Clarke and Emerson [CE81] and by Queille and Sifakis [QS82]. The basic idea is to construct a finite-state model of a system, to specify some properties in a (temporal) logic and to automatically verify the validity of these properties by means of an exhaustive search through the state space. In case the system satisfies all properties we are done, otherwise a counterexample is provided to either improve the system or maybe change the property. Figure 1.1 summarises the approach.

Due to a combinatorial explosion of the size of the state space in the number of variables and parallel components, model checking has shown to be rather difficult to scale to real-life applications. Therefore, methods for reducing the state space have been given quite some attention.

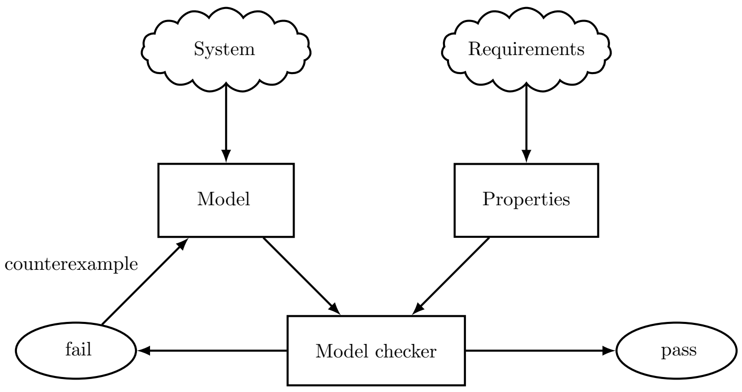


Figure 1.1: An overview of model checking.

1.2.1 Logics for model checking

Since the beginning of model checking [CE81], *temporal logics* [Pnu77] have been deemed a good method for reasoning about concurrent programs [Lam83]. Such logics deal with the *ordering* of events, and traditionally do not care about their timing. They are generally categorised based on whether the properties they specify are either in the *linear-time* domain or the *branching-time* domain.

Linear-time domain. Linear-time properties denote that a certain condition holds for all executions of a system. Such a property is actually just a set of traces, indicating which observable behaviour is considered to be correct. The most well-known logic to specify linear-time properties is *LTL* (Linear Temporal Logic) [Pnu77]. Most importantly, it has operators for saying that a condition over a set of *atomic propositions* holds *eventually* or that it should *always* hold. Later, a probabilistic extension was proposed in the form of *probabilistic LTL* [CY95]. Instead of being applied to verify if a certain condition holds for all paths through a system, it is applied to check if the *probability* of obtaining a path that satisfies the condition is above or below a given threshold.

Branching-time domain. Not all properties are expressible in linear time. For instance, the property "it is always possible to return to the initial state" cannot be translated to certain executions being either correct or incorrect: the possibility to return to the initial state does not mean that all paths indeed at some point have to take it—as long as the *option* to go back is always present.

Branching-time logics do allow such properties to be specified by means of existential and universal quantifications over paths. The most well-known branching-time logic for qualitative model checking is CTL (Computation Tree Logic) [CE81], later generalised to PCTL (probabilistic CTL) [HJ94] and CSL (continuous stochastic logic) [ASSB00, BHHK03, BHHZ11]. In PCTL, the existential and universal quantifications over paths are replaced by a *probabilistic quantification.* In CSL, *intervals* for the timing between events can be specified.

Moreover, it can specify *steady-state* properties that hold in the long run.

There is an ongoing debate about whether LTL or CTL is best [Hol04, Var01]. Luckily, they can be combined into an overarching logic CTL*. Similarly, probabilistic LTL and PCTL can be combined into the logic PCTL* [BdA95] that is able to express linear-time as well as branching-time properties. Since all techniques presented in this work preserve at least a variant of PCTL*, the debate between LTL and CTL does not concern us much.

1.2.2 Quantitative model checking

Over the last two decades, much effort has gone into the field of *quantitative model checking*. This field includes powerful techniques to analyse both qualitative properties and quantitative properties over models featuring discrete probabilities and/or timing (and often still also nondeterminism). They allow us to verify probabilistic as well as hard and soft real-time systems, modelled by timed automata (TAs), discrete-time Markov chains (DTMCs), Markov decision process (MDPs), probabilistic automata (PAs), continuous-time Markov chains (CTMCs), interactive Markov chains (IMCs), and Markov automata (MAs). Other notable extensions are the annotation of models with rewards or costs, yielding priced timed automata and Markov reward models, and enabling the verification of multi-objective problems [FKN$^+$11].

Software tools such as UPPAAL [BDL$^+$06], LiQuor [CB06], MRMC [KKZ05, KZH$^+$11], PRISM [KNP11], APMC [HLP06], and FHP-Murφ [PIM$^+$06] are dedicated quantitative model checkers that have been applied to a wide range of applications. The success of quantitative model checking is also witnessed by its adoption as a major analysis technique by tools that originate from performance analysis, such as GreatSPN [BBC$^+$09], Möbius [BCD$^+$03], PEPA WB [GH94], and SMART [CJMS06].

In this work we focus on the extension of traditional model checking by discrete-time probabilistic behaviour and continuous-time stochastic behaviour. As all extensions are based on labelled transition systems (LTSs), we first discuss their main feature: nondeterminism. Then, we discuss the three main extensions, as well as their practical applications. Finally, we discuss the main limitations of quantitative model checking and our contributions to the field.

Nondeterminism. As mentioned in the beginning of this chapter, nondeterminism is the unquantified uncertainty about a system's behaviour. Stated differently, a system is nondeterministic if at some point its precise behaviour is unknown to us (although the possible alternatives *are* specified). While probabilistic approaches specify the likelihood of each of the alternatives, nondeterminism leaves the choice completely open. A system nondeterministically choosing between providing coffee or tea may always provide coffee, serve coffee on Wednesdays and tea on the other days, throw a coin to decide between the two, or do something even different.

Nondeterminism may arise from the unspecified ordering of events of two or more (partly) independent parallel components, from interaction with an unpredictable environment or just from underspecification. It is a invaluable tool in the presence of uncertainty that cannot be resolved probabilistically. Traditional model checking tools are often able to compute whether a certain property holds for *all* possible ways to resolve the nondeterministic choices, whereas quantitative model checking tools often provide *minimal* and *maximal* probabilities for satisfying a given property (quantifying over all possible ways to resolve the nondeterministic choices in a probabilistic manner).

Probabilistic automata

When adding probabilistic behaviour to traditional labelled transition systems, we obtain Segala's *probabilistic automata* (PAs) [Seg95]—or discrete-time Markov chains (DTMCs) when restricting to deterministic systems (i.e., systems that do not allow multiple actions from the same state). These models allow us to specify transitions that do not have a unique target state anymore, but probabilistically decide on their continuation. This is highly practical, as discrete probabilistic behaviour is omnipresent:

Randomisation by design. Several protocols use randomisation to break their symmetry. For instance, the Itah-Rodeh leader election protocol uses probability to decide on a leader between identical nodes [IR90] and the IEEE 802.11 standard for wireless networks applies random backoffs to avoid collisions when multiple nodes try to access the network [IEE97]. Randomisation is also present in many board and card games, for instance due to the use of dice or because cards are drawn from a randomly shuffled deck.

Involuntary randomisation. Many practical systems also feature some natural uncertainty due to erroneous behaviour. For instance, congestion in the internet results in packet loss happening with a certain probability [KR01]. Additionally, many biological and physical systems behave in an unpredictable way. For instance, we do not know upon conception whether a baby will be a boy or a girl, we do not know which side of a coin will be on top when we toss it, and we do not know for sure if a medicine is going to work on a specific individual.

Note that, in fact, most of these phenomena are not really random anymore if we zoom in to an extremely precise level: in theory, it may be predicted which side of a coin will end up on top. We would need to consider the exact location of the coin, the precise hand movement, the non-perfect shape of the coin, the wind, etc—clearly, this is not feasible in practice (even if we ignore Heisenberg's uncertainty principle). Similarly, packet loss in the internet may be predicted by modelling the entire state of the network. Since such fine-grained analysis if often far from realistic, abstraction is applied and probability arises.

All of these phenomena can be modelled effectively as PAs, allowing us to verify properties in PCTL or probabilistic LTL and answer questions such as

- What is the probability of electing a leader within 5 rounds?
- What is the probability that a message in a network is lost at least once?

- What is the probability that a customer's demand can be satisfied from stock on hand, given a certain inventory management strategy?

Interactive Markov Chains

When adding stochastic behaviour to labelled transition systems, we obtain Hermanns' *interactive Markov chains* (IMCs) [Her02]—or continuous-time Markov chains (CTMCs) when restricting to deterministic systems. In addition to the action-labelled transitions of traditional model checking (an IMC's *interactive* part), this model also supports transitions that take a certain amount of time, determined by an exponential distribution—sometimes we also say that a state has a certain *rate* of going to another state. Instead of moving in discrete time steps, these models work in continuous time. This feature also allows us to model several phenomena that often occur in practice:

Waiting times. When standing in line for a cash register or waiting for someone to finish a phone call, the remaining waiting time may be unknown. Such waiting times are often modelled by exponentially distributed delays in the field of queueing theory [Hav98].

Failure rates. In dependability analysis, it is common to describe failure using a mean time to failure (MTTF). Often, for instance in dynamic fault trees [BCS10], the distribution of such failures is assumed to be determined by an exponential distribution.

All of these phenomena can effectively be modelled as IMCs, allowing us to verify properties in CSL and answer questions such as

- What is the probability that it is my turn at the cash register within 5 time units?
- What is the probability that a hard disk drive crashes within 10,000 hours of operation?
- What is the expected time until a phone call ends?
- What is the fraction of time that a processor will be idle in the long run?
- What is the probability that an emergency cooling system in a nuclear power plant does not switch on in time?

Markov Automata

PAs are great for modelling discrete probabilistic behaviour and IMCs for modelling continuous stochastic behaviour, but they have their separate domain of operation. In this thesis, we like to be as general of possible, and hence work with a recent combination of these two models: the *Markov automaton* [EHZ10b, EHZ10a]. By generalising PAs and IMCs, it also generalises the DTMC, CTMC and LTS. Hence, MAs can be used as a semantic model for a wide range of formalisms, such as generalised stochastic Petri nets (GSPNs) [ACB84], dynamic fault trees [BCS10], Arcade [BCH^{+}08] and the domain-specific Architecture Analysis and Design Language (AADL) [BCK^{+}11]. MAs are very general and,

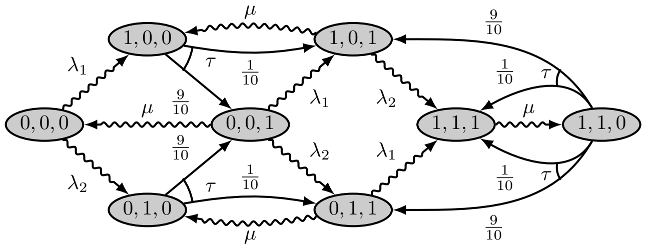


Figure 1.2: A queueing system, consisting of a server and two stations.

except for hard real-time deadlines and hybrid systems, can describe most behaviour that is modelled today in theoretical computer science.

Most work on MAs so far has focused on defining appropriate notions of weak bisimulation. The seminal work by Eisentraut, Hermanns and Zhang first provided a notion of 'naive weak bisimulation' (which is a straightforward generalisation of traditional weak bisimulation for PAs and IMCs) and then improved on this by defining a notion of weak bisimulation that exploits the interplay of rates and probabilistic invisible transitions [EHZ10b, EHZ10a]. Shortly after, [DH11, DH13] introduced another notion of weak bisimulation for MAs. In was shown in [SZG12] that these notions coincide. Additionally, [SZG12] introduced yet another notion of weak bisimulation for MAs, and showed that it is coarser (i.e., equates more systems) than the earlier two notions.

Example 1.1. As an example of an MA, Figure 1.2 shows the state space of a polling system with two arrival stations and probabilistically erroneous behaviour (inspired by [Sri91]). The two stations have incoming requests with rates λ_1, λ_2, which are stored until fetched by the server. If both stations contain a job, the server chooses nondeterministically from which of them to fetch the next task. Jobs are processed with rate μ, and when polling a station there is a $\frac{1}{10}$ probability that the job is erroneously kept in the station after being fetched. Each state is represented as a tuple (s_1, s_2, j), with s_i the number of jobs in station i, and j the number of jobs in the server. For simplicity we assume that each component can hold at most one job.

In Chapter 9 we will discuss a more complicated variant of this system, demonstrating how to compute for instance the expected time until reaching full capacity for the first time, the probability that full capacity is already reached within the first two time units, and the fraction of time that all arrival stations are at full capacity in the long run. $\square$

Logics and algorithms for model checking MAs. At this moment, there is only limited related work on logics for Markov automata. The only logic we are aware of is a variant of CSL introduced in [HH12], containing operators for unbounded and time-bounded reachability, but not dealing with expected times or long-run averages as allowed by other variants of CSL for different models [BHHZ11].

Although not supported by a formal logic, Guck introduced algorithms for computing long-run averages and expected times to reachability [Guc12].

Also, Hafeti and Hermanns showed how to do time-bounded reachability analysis [HH12]. These results were unified in [GHH$^+$13a], also providing tool support by means of the IMCA tool.

1.2.3 Previous limitations and current contributions

No full-fledged formal modelling languages aimed at specifying MAs existed thus far. As it is often infeasible to manually write down a low-level transition system, this greatly limited the applicability of MAs. Additionally, model checking is prone to the state space explosion: data variables and interleavings due to parallel composition quickly yield a large number of states. In quantitative model checking the effects of this explosion are even worse [KKZJ07], as the numerical algorithms for computing quantitative properties are more time-consuming than their non-probabilistic counterparts.

Contributions. We contribute to both issues by providing a process-algebraic modelling language targeted specifically at MAs. It allows MAs to be modelled efficiently by means of data, and enables reduction technique to be defined easily due to its simplicity. The next sections go into more details on both issues.

This thesis aims at *efficient modelling* of *Markov automata*, as well as *reducing* the *state space explosion* during their formal verification.

1.3 Process algebras

"*Process algebra became an underlying theory of all parallel and distributed systems, extending formal language and automata theory with the central ingredient of* interaction."

Jos Baeten [Bae05]

While model checking algorithms are mostly defined on models such as PAs, IMCs and MAs, it would be rather inconvenient to explicitly provide such models. After all, model-based system specifications tend to get incredibly large even for simple systems. Therefore, it is more common to specify systems in some type of higher-level language that is mapped to a formal model. For efficient specification, such a language should have compositionality features, allowing the user to model several components independently. In addition to simplifying the specification phase, higher-level languages also allow us to perform syntactic optimisations on the language level to generate reduced models without even having to generate the unreduced variant in the first place.

In this work, we focus on *process-algebraic* modelling languages (also called process calculi) [Fok07, Bae05]. An important feature of such languages is their mathematical thoroughness, describing behaviour by means of algebraic terms. Additionally, a characterising feature of process algebras is the *parallel composition* operator: a powerful method to compose a system by specifying its various subsystems and their interaction.

Traditionally—in addition to having an operational semantics defined in terms of a model such as the IMC or MA—process algebras are often accompanied by a set of algebraic laws. Preferably this set is an axiomatisation for some notion of equivalence. That is, two process-algebraic descriptions yield equivalent models if and only if they themselves are equivalent according to the algebraic laws. In this work, we rather employ process algebra to specify systems formally and generate the underlying models[1]. We do perform syntactical transformations on the process-algebraic descriptions, but prove them correct by showing a bisimulation relation between the original and the transformed specification.

The framework for working with MAs introduced in this thesis is related to several existing formalisms. Mainly, it is based on μCRL, a process algebra for specifying labelled transition systems. Also, it has been influenced by the language TPCCS that extended CCS with discrete probabilistic choice, and the language IML for specifying IMCs. It shows similarities to the mCCS language for specifying MAs. We discuss these languages in some detail, and explain in the next section how these languages have been used in our framework.

μCRL [GP95] The language μCRL was developed to more easily specify communicating processes in the presence of *data*, by adding equational abstract data types to the process algebra ACP [BK89]. Data can be used to parameterise processes as well as actions, and in conditions to affect a process' behaviour. Also, it can be used to concisely specify behaviour due to the possibility of summing over (possibly infinite) data types—generalising the well-known process-algebraic *alternative composition* operator.

In μCRL, the LPE (linear process equation) [BG94b] was introduced as a restricted part of the language for easier formal manipulation. It is similar to the Greibach normal form in formal language theory, specifications in the language UNITY [CM88], and the precondition-effect style used for describing I/O automata [LT89]. Usenko showed how to *linearise* a general μCRL specification into an LPE [GPU01, Use02], based on ideas first proposed in [BP95].

TPCCS [HJ90] The language TPCCS added a discrete probabilistic operator to the CCS process algebra, introducing the notation $\textstyle\sum$ that we also apply. TPCCS has its semantics defined in the so-called *alternating model*, having disjoint sets of nondeterministic and probabilistic states. During the course of execution of the alternating model, there is a strict alternation between

[1]One may therefore argue about whether or not our language is actually a process algebra, depending on the definition that is used. According to [HJ90], process algebras are "structured description languages for concurrent systems, in which a few simple constructs (e.g., sequential and parallel composition, nondeterministic choice, and recursion) lead to languages capable of describing the various aspects of distributed systems". Similarly, [Fok07] states that process algebra is "a mathematical framework in which system behaviour is expressed in the form of algebraic terms, enhancing the available techniques for manipulation". Our language precisely satisfies the first definition, and could satisfy the second depending on the interpretation of the word 'algebraic'. On the other hand, according to [Bae05] a process algebra is "any mathematical structure satisfying the axioms given for the basic operators". When applying this definition, our work should not be called a process algebra due to the absence of axioms. We could also just call it a *process calculus*, like Milner [Mil80].

these types of states. As mentioned in [Sto02a], any alternating model can be transformed to an equivalent probabilistic automaton.

IML [Her02] The language IML was developed to model IMCs. It contains operators to specify actions and well as Markovian rates. Its operational rules that prescribe how each IML specification is mapped to an IMC take into account the *multiplicity* of each transition. This is vital for stochastic process algebras, since additional occurrences of the same rate increase the speed of a process. There is a wide variety of other stochastic process algebras featuring similar properties; an overview of the foundational ones is provided in [HHK02].

mCCS [DH11, DH13] The language mCCS is the only other process algebra we are aware of that also aims at constructing MAs. It contains discrete probabilistic choice in the same way as defined first in [HJ90], and can specify Markovian rates in the same way as IML. However, as already stated in [DH11], it is incomplete due to the ignorance of multiplicities.

1.3.1 Previous limitations and current contributions

None of the existing languages discussed in the previous section serves the purpose of providing a basis for efficient modelling, generation and analysis of MAs. Except for mCCS, none of them is able to construct MAs. Except for μCRL, none of them is able to deal with data or easily allows reduction techniques to be defined. Still, together these languages contain all the ingredients for a process algebra to efficiently work with MAs.

We note that the Modest language [BDHK06] is in principle also able to model MAs, but it has not yet specifically been applied in this context. It is a very expressive language, but does not contain a restricted form such as μCRL's LPE. Additionally, it does not allow MAs to be reduced for efficient analysis.

Contributions. We took from μCRL its way of dealing with data, as well as the idea of first linearising each specification to a restricted part of the language for easier formal manipulation. We extended this with an operator for discrete probabilistic choice, similar to the one introduced in TPCCS [HJ90] but fully integrated in the language in such a way that probabilistic behaviour can also depend on data. Finally, we took from IML the operator for modelling Markovian rates (and also allow these rates to depend on data parameters), as well as the idea of taking into account the number of derivations for each Markovian transition.

The result is *MAPA* (Markov Automata Process Algebra): a language rich enough to efficiently model nondeterminism, discrete probabilistic choice and stochastic timing. MAPA improves on mCCS—the only other process algebra for MAs—by the incorporation of data and a linear format (in MAPA called the MLPE) that allows reduction techniques to be defined in a rather simple manner, as well as by its proper way of dealing with multiplicities of rates. MAPA includes operators for parallel composition and hiding, allowing systems to be modelled in a compositional manner.

We introduce a novel notion of bisimulation: *derivation-preserving bisimulation.* It is defined on a subset of MAPA (called prCRL), obtained by omitting the stochastic timing. We show how to *encode* MAPA specifications in prCRL, and prove that transformations of prCRL that preserve our new notion of bisimulation can be applied safely before decoding back—the result then is a MAPA specification that is bisimilar to the specification we started with. This procedure allows us to reuse techniques defined on prCRL immediately to the full range of MAPA. We apply it for the transformation from MAPA to MLPE and for multiple reduction techniques.

MAPA can be seen as a maximal analysable subset of MODEST (only disregarding its realtime features). By specialising our language to this subset we are able to generalise μCRL's LPE format to the MLPE, enabling reduction techniques that have not been defined yet for the MODEST language.

This thesis introduces the process-algebraic language *MAPA*, allowing *efficient modelling* of MAs using *data* and enabling their *efficient generation and analysis* by supporting several *reduction techniques* via the *MLPE.*

1.4 Reduction techniques

"*A clever person solves a problem.
A wise person avoids it.*"

Albert Einstein

As mentioned before, the applicability of model checking is bound by the state space explosion: the number of states that has to be visited often grows exponentially with the size of the system. Much research has been devoted to techniques that battle this omnipresent state space explosion [Pel08].

These techniques focus on either reducing the amount of time needed for model checking large systems, the amount of memory, or both. Three different types of approaches can be observed, depending on whether they focus on efficient representation, efficient computation or efficient generation. For all techniques, the significance of their reductions depends on the model at hand.

Efficient representation. One way is to more efficiently store state spaces by means of techniques such as state compression [Hol97, LLPY97, LvdPW11], or to use symbolic representations like binary decision diagrams [McM93, BvdP08]. Although these techniques aim at improved memory usage, they may also reduce analysis time.

Efficient computation. Another approach is to more effectively use the increasing number of cores in today's hardware via smart parallelisations [Web06] or to use multiple machines for distributed model checking [BvdPW10]. These techniques are mostly aimed at speeding up model checking; however, the distribution of work over several machines may also partly solve the memory problem.

Efficient generation. Finally, both memory usage and analysis time may be improved by using techniques that reduce the number of states or only explore part of a state space. There are several techniques that do so without influencing the truth values of the properties of interest. Most notable are ingenuous techniques such as symmetry reduction [ES96, ID96, CJEF96, KNP06], partial order reduction [Val90, Val93, Pel93, GP93, God96, DN04, BGC04, BDG06], confluence reduction [BvdP02], and a recent approach for on-the-fly symbolic bisimulation minimisation [DKP13]. Instead of reducing the model automatically, it is of course also possible to model in such a way to keep the state space as small as possible to begin with [GKO12].

Alternatively, simulation-based approaches such as statistical model checking [HLMP04, LDB10, YS02] can be used to visit only part of a state space without storing any states (requiring only constant memory). As a downside, such techniques can only provide approximate results. Also, abstraction techniques such as counter-example guided abstraction refinement (CEGAR) can be applied to partition a state space into smaller regions [HWZ08].

Whereas all these techniques focus on alleviating the state space explosion *on-the-fly*, it is also possible to minimise a system's state space *after* first having generated the unreduced variant. Whereas for traditional model checking this often does not help us much, it may benefit quantitative model checking [KKZJ07].

1.4.1 Previous limitations and current contributions

Although many of the existing techniques greatly stretch the boundaries of model checking, it is still difficult to verify properties of real-life systems. Therefore, there is a sustained demand for reduction techniques to alleviate the state space explosion. Additionally, while we are very interested in more and more expressive quantitative model checking—it allow us to verify properties about a larger class of systems and unifies existing approaches—this makes the necessity for new and better reduction techniques even larger. After all, quantitative properties over probabilistic or stochastic models require numerical procedures that are obviously more complex than the qualitative decision procedures. Also, techniques such as partial order reduction and confluence reduction do not automatically work for more expressive models; they have to be generalised. Hence, it is far from trivial for model checking to be both efficient and expressive.

Contributions. Our goal is to optimise quantitative model checking by reducing before or during state space generation, rather than afterwards. That way, we never have to generate the larger unreduced state space, we limit memory requirements and we immediately allow for efficient analysis.

Exploiting the MLPE format, we define several automatic reduction techniques for more efficient generation of the underlying state space of a MAPA specification. First of all, we generalise three *basic reduction techniques* from the qualitative to the quantitative domain: constant elimination, summation elimination and expression simplification. These techniques try to rewrite MLPEs to make them more concise, improving readability and facilitating faster state space generation (without actually influencing the state space itself).

Second, we present a state space reduction technique based on *dead variables*, resetting variables to their initial value in case they will always be overwritten before being used again. It is especially powerful due to a novel method for detecting control flow. This technique detects which state parameters act as program counters. Liveness analysis is then performed based on these reconstructed program counters, and hence even takes into account control flow that was manually encoded in a system's global parameters. Dead variable reduction preserves strong bisimulation, and hence leaves invariant all properties in PCTL* when restricting to PAs and in CSL when restricting to IMCs.

Third, we generalise *confluence reduction* to the Markovian realm and show how apply it on MAPA specifications. This technique reduces spurious interleavings, often arising from the parallel composition of largely independent processes. We investigate confluence reduction in great detail by comparing it to its competitor *partial order reduction.* As partial order reduction has not yet been introduced for MAs, we restrict confluence for the subclass of MDPs and theoretically show that it strictly subsumes the partial order reduction variant that preserves branching time properties [BDG06]. We also define confluence reduction for the context of *statistical model checking*, and compare partial order reduction to confluence reduction on a practical level. It turns our that our confluence implementation is able to reduce more than the previous implementation based on partial order reduction, allowing more systems to be analysed with this promising technique. Confluence reduction preserves divergence-sensitive branching bisimulation, and hence leaves invariant all properties in PCTL$^*_{\setminus X}$ when restricting to PAs and all time-bounded reachability properties when restricting to IMCs.

We implemented all our reduction techniques in the tool SCOOP, and provide several case studies that show significant reductions. No such reductions have ever been obtained for MAs before. As MAs generalise LTSs, DTMCs, CTMCs, PAs and IMCs, all our reduction techniques are also applicable to these models.

This thesis introduces *dead variable reduction*, *confluence reduction* and some *basic reduction techniques* for the MLPE. Additionally, it investigates confluence reduction in-depth by *comparing* it to *partial order reduction* from a *theoretical* as well as *practical* perspective.

1.5 Main contributions

Summarising, this thesis contributes to the field of quantitative verification in several ways. Our main goal was to enable *more expressive* models to be specified, generated and analysed in an *efficient manner.* While initially working on probabilistic automata—extending existing techniques with discrete probabilistic choice—we generalised this even further to Markov automata when these were introduced. As this formalism subsumes a large variety of important models, its generality allows our techniques to be applied to an equally large variety of systems. Our main contributions are threefold:

- We enable the *efficient modelling* of MAs, by providing the new process algebra MAPA. It treats data as a first-class citizen, allowing complicated behaviour involving data-dependent probabilistic branching or delays to be modelled quite easily. MAPA includes operators for parallel composition and hiding, allowing systems to be modelled in a compositional manner. Since MAs generalise a wide variety of simpler models, MAPA allows systems of all these types to be modelled in a uniform manner.

- We present an encoding of MAPA into prCRL, a restricted part of the language obtained by omitting its stochastic timing aspect. We also introduce a novel notion of bisimulation for prCRL. We show that transformations on prCRL preserving this notion of bisimulation can safely be used on encoded MAPA specifications—this still guarantees a strongly bisimilar system after decoding back to MAPA. Based on this approach, we also present an algorithm for rephrasing each MAPA specification in a stripped-down (but equally expressive) version of the language: the MLPE. Due to its simplicity, this format allows reduction techniques to be defined much more easily.

  We enable the *efficient generation and analysis* of MAs by defining several reduction techniques on the MLPE. Three basic reduction techniques speedup state space generation, while dead variable reduction and confluence reduction additionally speedup analysis. Case studies demonstrate the usefulness of these techniques through significant reductions.
- We provide new understanding about the relation between confluence reduction and partial order reduction, by providing both a theoretical comparison, and a practical comparison in the context of statistical model checking. We show that confluence reduction is more powerful in theory when restricting to the preservation of branching-time properties, and that its on-the-fly detection may even beat partial order reduction in the linear-time context of statistical model checking.

1.6 Organisation of the thesis

The remainder of this thesis is organised in the following way:

Chapter 2 recalls the necessarily mathematical backgrounds for this work from set theory and probability theory.

Chapter 3 presents the MA and discusses several behavioural equivalence relations for this model and its subclasses.

Chapter 4 introduces the MAPA language, shows how to safely define transformations on a restricted version of the language and introduces the MLPE including its linearisation procedure. Also, it defines three basic reduction techniques on the MLPE.

Chapter 5 introduces our dead variable reduction technique, including the method for control flow reconstruction.

Chapter 6 generalises confluence reduction, formally defining confluence on MAs, presenting heuristics for detecting it on MLPEs and demonstrating how to apply confluence for generating reduced state spaces.

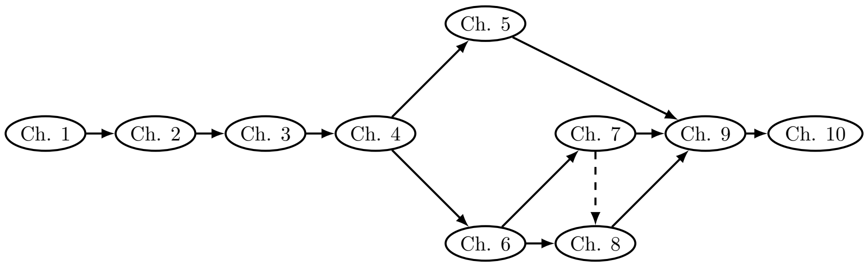


Figure 1.3: Chapters roadmap.

Chapter 7 provides a theoretical comparison of confluence reduction and partial order reduction.

Chapter 8 shows how to apply confluence reduction during statistical model checking, proving that it outperforms partial order reduction in some cases.

Chapter 9 discusses our implementation and validates our reduction techniques by means of a selection of case studies.

Chapter 10 concludes the thesis by providing a discussion on the advantages and disadvantages of the MAPA language and the reduction techniques we introduced, as well as providing directions for future work.

Appendix A provides proofs for all our results.

Appendix B provides a complete list of papers by the author of this thesis.

The introduction of each chapter mentions the papers on which it is based. The chapters are meant to be read sequentially, but some variation to this is possible. The dependencies between chapters are depicted in Figure 1.3 (where the dashed arrow indicates a rather small dependency).

Part I

Background

CHAPTER 2

Preliminaries

"*Pure mathematics is, in its way, the poetry of logical ideas.*"

Albert Einstein

Most of the technicalities in the next chapters rely on set theory and probability theory. As formal treatments of both topics are rather involved, we restrict ourselves to informally recalling the relevant notions and introducing the notations that are used throughout the rest of this thesis.

All material in this chapter is well known. The set-theoretical notions we discuss can be found in almost any textbook on discrete mathematics (e.g., [Gri03]). The basics from probability theory can for instance be found in [Ros11, CK04].

Organisation of the chapter. Section 2.1 discusses the basics of set theory, and Section 2.2 the basics of probability theory.

2.1 Set theory

One of the most fundamental concepts in mathematics is the notion of a *set*: an abstract collection of objects. Finite sets can be described explicitly, enumerating all of their elements: $X = \{1, 3, 5, 7, 9\}$. We use $3 \in X$ to indicate that the element 3 is contained in X, and $4 \notin X$ to indicate that 4 is not. The *empty set* is denoted by $\varnothing$. Moreover, the set of natural numbers is denoted by $\mathbb{N} = \{0, 1, 2, \dots\}$, the set of positive natural numbers by $\mathbb{N}^+ = \{1, 2, \dots\}$, the set of real numbers by $\mathbb{R}$, the set of nonnegative real numbers by $\mathbb{R}^{\geq 0}$ and the set of positive real numbers by $\mathbb{R}^{>0}$ (also known as $\mathbb{R}^+$).

The elements of a set do not have a multiplicity and are unordered: the sets $\{1, 2, 3\}$ and $\{1, 3, 2, 1\}$ are identical. In a *multiset*, elements do have a multiplicity. We denote multisets like $\{\!|1, 1, 2|\!\}$, indicating the multiplicity of each element by the number of times it is written.

2.1.1 Building and comparing sets

Sets can be defined using the *set-builder notation* (also called *set comprehension*), given a certain *universe* of all possible elements and a condition. For instance, we write $\{x \in \mathbb{N} \mid x < 5\} = \{0, 1, 2, 3, 4\}$ for all natural numbers smaller than 5.

Often, the universe is implied by the context and an expression is used at the left-hand side of the vertical bar. In that case, the condition at the right-hand side of the vertical bar specifies for which values from the universe, the value obtained by evaluating the expression on its left-hand side is present in the resulting set. For instance, we could write $\{x^2 \mid x < 5\} = \{0, 1, 4, 9, 16\}$.

Given two sets X, Y, we use $X \cap Y = \{x \mid x \in X \wedge x \in Y\}$ to denote the *intersection* of X and Y, i.e., the set of all elements that are both in X and in Y. We say that X and Y are *disjoint* if $X \cap Y = \varnothing$. We write $X \cup Y = \{x \mid x \in X \vee x \in Y\}$ for the set of all elements that are in either X or Y (or in both): their *union*. We use $X \setminus Y = \{x \in X \mid x \notin Y\}$ to denote the *difference* of X and Y, i.e., the set of elements from X that are not in Y. Finally, the *disjoint union* of two sets X, Y is given by $X \uplus Y = \{(x, 1) \mid x \in X\} \cup \{(y, 2) \mid y \in Y\}$.

The concepts of intersection, union and disjoint union are lifted naturally to be applied to more than two sets. For intersections, we let

$$\bigcap_{i \in \{1,2,3,\dots\}} X_i \quad = \quad \bigcap \{X_1, X_2, X_3, \dots\} \quad = \quad X_1 \cap X_2 \cap X_3 \cap \dots$$

Unions and disjoint unions are generalised in the same manner.

A set X is a *subset* of a set Y, denoted by $X \subseteq Y$, if every element in X is also in Y, i.e., $X \subseteq Y \iff \forall x \in X \,.\, x \in Y$. It is easy to see that this coincides with $X \setminus Y = \varnothing$. If $X \subseteq Y$ and $X \neq Y$, we write $X \subset Y$ and say that X is a *strict subset* of Y. If X is a (strict) subset of Y, then Y is a (strict) *superset* of X, denoted by $Y \supseteq X$ in general and $Y \supset X$ for the strict version. We use $X \not\subseteq Y$ to denote that X is not a subset of Y, and apply similar negations of the other symbols.

Given a set X, we use $\mathscr{P}(X)$ to denote its *powerset*, i.e., the set of all its subsets (including the empty set and X itself).

2.1.2 Relations and functions

The *Cartesian product* of two sets X, Y, denoted by $X \times Y$, is the set of all pairs (x, y) such that $x \in X$ and $y \in Y$. A *binary relation* between two sets X, Y is a subset of $X \times Y$. If $X = Y$, it is called a binary relation *on* X. Given a binary relation R, we sometimes write xRy to indicate that $(x, y) \in R$.

Given two binary relations R_1 and R_2 over a set X, we use $R_2 \circ R_1$ to denote their *composition*:

$$R_2 \circ R_1 = \{(x, z) \in X \times X \mid \exists y \in X \,.\, (x, y) \in R_1 \wedge (y, z) \in R_2\}$$

Equivalence relations. A binary relation R on a set X is called *reflexive* if $(x, x) \in R$ for every $x \in X$, *symmetric* if $(x, y) \in R$ implies $(y, x) \in R$ and *transitive* if $(x, y) \in R$ and $(y, z) \in R$ together imply $(x, z) \in R$. Binary relations that have all these properties are called *equivalence relations*.

Given an equivalence relation $R \subseteq X \times X$, we write $[x]_R$ for the *equivalence class* induced by x, i.e., $[x]_R = \{y \in X \mid (x, y) \in R\}$. We denote the set of all such equivalence classes by X/R. Due to the nature of equivalence relations, the equivalence classes *partition* X. That is, two equivalence classes $[x]_R, [y]_R$ either coincide or are disjoint, and the union of the equivalence classes equals X.

Example 2.1. Let $X = \{1,2,3,4,5,6,7,8\}$, then the relation

$$R = \{(x,y) \in X \times X \mid x \text{ and } y \text{ are both even or both odd}\}$$

is an equivalence relation. To see why R is transitive, note that $(x,y) \in R$ and $(y,z) \in R$ imply that x, y, z are all even or all odd, and hence $(x,z) \in R$ is immediate. Reflexivity and symmetry are shown similarly.

We find that, for instance, $[1]_R = [3]_R = \{1,3,5,7\}$ and $[4]_R = \{2,4,6,8\}$. As these are the only equivalence classes, $X/R = \{\{1,3,5,7\},\{2,4,6,8\}\}$. $\square$

Functions. A (total) *function* from a set X (its *domain*) to a set Y (its *range*) is a relation $R \subseteq X \times Y$ such that $(x,y) \in R$ and $(x,z) \in R$ imply $y = z$, and for each $x \in X$ there is a pair $(x,y) \in R$. Hence, a function from X to Y *maps* each element of X to precisely one element of Y. If $(x,y) \in R$ for a function R, we also write $R(x) = y$ and say that y is the *image* of x under R. Often, we write $f \colon X \to Y$ to indicate that f is a function from X to Y.

A function $f \colon X \to Y$ is *injective* if $f(x) = f(y)$ implies $x = y$ for all $x, y \in X$. Hence, it never maps two different values from X to the same value from Y. It is *surjective* if it uses its complete range, i.e., if for every $y \in Y$ there exists an $x \in X$ such that $f(x) = y$. A function that is both injective and surjective is said to be *bijective*. Such functions represent a one-to-one mapping between the elements of X and Y, and hence can only exist if X and Y have the same number of elements.

Given a function $f \colon X \to Y$ and an element $y \in Y$, the *inverse image of* y *under* f is denoted by $f^{-1}(y)$ and given by the set $\{x \in X \mid f(x) = y\}$. Note that $f^{-1}(y)$ contains at most one value if f is injective and at least one value if f is surjective. Hence, for bijective functions it always contains precisely one value.

Example 2.2. The function $f \colon \mathbb{N} \to \mathbb{N}$ given by $f(n) = n^2$ is injective, but not surjective (and hence not bijective). The image of 3 under f is 9, and the inverse image of 10 is $f^{-1}(10) = \varnothing$.

The function $f \colon \mathbb{R} \to \mathbb{R}$ given by $f(n) = n^2$ is neither surjective nor injective. After all, the negative numbers do not appear as the image of any value from the domain, and $f^{-1}(4) = \{-2, 2\}$ contains more than one value.

The function $f \colon \mathbb{R} \to \mathbb{R}$ given by $f(x) = x^3$ is both injective and surjective, and hence bijective. $\square$

A *partial function* $f \colon X \rightharpoonup Y$ is similar to a function from X to Y, except that it does not necessarily assign a value from Y to each element in X. Hence, it behaves like a function $f' \colon X' \to Y$ for some set $X' \subseteq X$. We use *dom*(f) to denote this domain X'.

Set cardinality. The *size* or *cardinality* of a set X, denoted by $|X|$, is its number of elements. Obviously, $|X \times Y| = |X| \cdot |Y|$. Any set with a finite number of elements is called a *finite* set. We define a set to be *countable* if it contains at most as many elements as the set $\mathbb{N}$ of natural numbers. Formally, a set X is countable if there exists an injective function $f \colon X \to \mathbb{N}$. A set that is not countable is called *uncountable*.

Note that our definition allows countable sets to be finite. Countable sets that are infinite are called *countably infinite*.

Example 2.3. Let $X = \{x \in \mathbb{R} \mid 2x \in \mathbb{N}\}$, i.e., $X = \{0, \frac{1}{2}, 1, \frac{3}{2}, \dots\}$. Although intuitively this sets seems to be bigger than $\mathbb{N}$, there *does* exist an injective function $f \colon X \to \mathbb{N}$. After all, take $f(x) = 2x$ for every $x \in X$. This clearly is a function from X to $\mathbb{N}$, and no element in $\mathbb{N}$ is the image of more than one element from X. Hence, it is injective. □

It can be shown that the set of real numbers $\mathbb{R}$, as well as any nontrivial interval $[a, b] \in \mathbb{R}$, is uncountable.

2.1.3 Summations and sequences

Summations. Given a function $f \colon X \to Y$ and a countable set $X' \subseteq X$, such that $Y \subseteq \mathbb{R}$ and either X' is finite or f is nonnegative (i.e., $f(x) \geq 0$ for every $x \in X$), we use $\sum_{x \in X'} f(x)$ to denote the summation of the image under f of all elements of X'. Given a countable set $X \subseteq \mathbb{R}$ such that $|X| < \infty$ or $X \subseteq \mathbb{R}^{\geq 0}$, we use $\sum X$ as an abbreviation for $\sum_{x \in X} x$. Similarly, for a multiset Y satisfying the same conditions, we let $\sum Y$ be the sum of its elements (taking into account their multiplicities).

We note that the finiteness or nonnegativity constraints are needed for the summation order to be irrelevant and a summation to either diverge or converge (and thus not oscillate), and hence for the notion $\sum_{x \in X} f(x)$ to be well-defined [Bro08]. Often we do not explicitly provide the function f, but write an expression that implicitly captures it. For instance, we write x^2 to denote the function $f \colon \mathbb{R} \to \mathbb{R}$ such that $f(x) = x^2$.

Sequences. For a sequence $\sigma = a_1 a_2 a_3 \dots a_k$ of elements from some set L (sometimes also written as $\langle a_1, a_2, a_3 \dots, a_k \rangle$), we write $|\sigma| = k$ to denote its length. Given an element $a \in L$, we use $a\sigma$ to denote the sequence $a a_1 a_2 a_3 \dots a_k$. The empty sequence is denoted by ϵ. Given a subset $L' \subseteq L$, we let $\sigma \setminus L'$ denote the sequence σ' obtained by removing every occurrence of elements from L' in σ.

2.2 Probability theory

When systems behave in an uncertain manner, probability theory can be used to formalise this behaviour. The basis of probability theory is the *experiment*: some real-world procedure with a predetermined set of possible *outcomes*. Each execution of the experiment will result in one specific outcome; the next execution may have a different one. A standard example is the roll of a die; it has six different outcomes, corresponding to which side ends up on top.

For a fair die, the *probability* of rolling a six is $\frac{1}{6}$. Intuitively, this means that if the experiment is repeated N times (with N going to infinity), the outcome will be six approximately $\frac{1}{6} \cdot N$ times. This intuition immediately explains why probabilities have to be nonnegative and have to sum up to one. After all, an event cannot be expected to occur a negative number of times, and together the outcomes should account for all N experiments.

2.2.1 Probability spaces

Mathematically, an experiment is modelled by a *probability space*, consisting of a *sample space*, a set of *events* and a *probability measure*.

The sample space Ω is just a non-empty set consisting of all possible outcomes of the experiment modelled by the probability space. We want to assign probabilities to the occurrence of the elements of this sample space, but in general it does not suffice to do so only for each individual outcome. After all, for an experiment yielding a value from a continuous domain, we often find that the probability of each individual outcome is 0; still, there is a positive probability of an outcome within a certain *set* of outcomes. Therefore, events are defined as sets of outcomes—i.e., subsets of the sample space—and the probability space contains a set $\mathcal{F}$ of all events to which a probability is assigned by the probability measure P.

Formally, a probability space is a tuple $(\Omega, \mathcal{F}, P)$, where

1. $\Omega \neq \varnothing$ is the sample space
2. $\mathcal{F} \subseteq \mathscr{P}(\Omega)$ is the set of events, such that
   - $\Omega \in \mathcal{F}$
   - $E \in \mathcal{F}$ implies $\Omega \setminus E \in \mathcal{F}$
   - $E_i \in \mathcal{F}$ for $i = 1, 2, \dots$ implies $\bigcup_{i=1}^{\infty} E_i \in \mathcal{F}$
3. $P \colon \mathcal{F} \to [0, 1]$ is the probability measure, such that
   - $P(\Omega) = 1$
   - $P(\bigcup_{i=1}^{\infty} E_i) = \sum_{i=1}^{\infty} P(E_i)$, if $E_j \cap E_k = \varnothing$ for all $j \neq k$.

Example 2.4. Consider the experiment of rolling a die. We may model this experiment by the probability space $(\Omega, \mathcal{F}, P)$, with $\Omega = \{1, 2, 3, 4, 5, 6\}$, $\mathcal{F} = \mathscr{P}(\Omega)$ and $P(E) = \frac{|E|}{6}$ for every $E \in \mathcal{F}$. It is easy to check that $\mathcal{F}$ and P indeed satisfy their constraints.

The event $O = \{1, 3, 5\} \in \mathcal{F}$ represents the scenario that the experiment yields an odd outcome. We easily compute $P(O) = \frac{3}{6} = \frac{1}{2}$. □

Probability measures. To understand the restrictions on the probability measure, first note that the event Ω corresponds to the occurrence of *any* outcome of the sample space. As this is unavoidable, clearly $P(\Omega) = 1$ has to hold. Additionally, if we know the probabilities of two or more disjoint events, then the probability of the union of these events intuitively indeed should be the sum of their probabilities (this property is called *countable additivity*). Any function with range $[0, 1]$ satisfying these two properties is called a *probability measure*.

Measurable spaces and σ-algebras. To understand the restrictions on the set of events, note that the first and third are obviously needed for the probability measure to be well-defined. The second corresponds to the intuition that if we know the probability p of a certain event, we also know the probability of its complement: $1 - p$. Hence, for each event in $\mathcal{F}$, also its complement should be in $\mathcal{F}$. Any set satisfying these properties is called a *σ-algebra*. A pair (Ω, F)

of a sample space and an associated σ-algebra is called a *measurable space*. A well-known example of a measurable space is $(\mathbb{R}, \mathcal{B})$, with $\mathbb{R}$ the set of real numbers and $\mathcal{B}$ the *Borel σ-algebra* over $\mathbb{R}$, i.e., the smallest σ-algebra over the real numbers containing all intervals.

Although it may seem that we can take $\mathcal{F} = \mathscr{P}(\Omega)$ for every Ω, it is well known from measure theory that this is impossible in general. There are sets (e.g., the Vitali sets[1]) that are so inherently complicated, that no probability can be assigned to them in a satisfactory way while satisfying the constraints above. Hence, they must be *excluded* from the domain of the probability measure, in order to still be able to define useful measures. This is precisely the purpose of the set $\mathcal{F}$. However, if Ω is countable, $\mathcal{F} = \mathscr{P}(\Omega)$ can always be used.

2.2.2 Random variables

Instead of dealing with a probability space $(\Omega, \mathcal{F}, P)$ directly, it is more common to work with an additional layer of *random variables*. These are just functions $X \colon \Omega \to S$ from the sample space to some other domain S, under the (often implicit) restriction that S is part of some measurable space $(S, \mathcal{S})$. For a countable set S we can always take $\mathcal{S} = \mathscr{P}(S)$, and for $S = \mathbb{R}$ the Borel σ-algebra often suffices.

As a random variable assigns a value to each outcome of an experiment, we can ask for the probability that this value is in a certain set $E \in \mathcal{S}$, defined by

$$\Pr(X \in E) = P(X^{-1}(E)) = P(\{w \in \Omega \mid X(w) \in E\})$$

Similarly, we use notations like $\Pr(X < 5)$ and $\Pr(X = s_4)$. Note that, for such probabilities to be well-defined, we need $X^{-1}(E) \in \mathcal{F}$ for every $E \in \mathcal{S}$. We then say that X is *$(\mathcal{F}, \mathcal{S})$-measurable*.

Example 2.5. Consider again the rolling of a die, represented earlier by a probability space $(\Omega, \mathcal{F}, P)$. We define a random variable $X \colon \Omega \to \{\mathit{even}, \mathit{odd}\}$, given by $X(1) = X(3) = X(5) = \mathit{odd}$ and $X(2) = X(4) = X(6) = \mathit{even}$.

[1]To understand the intricacies, consider a probability distribution P over the sample space $\Omega = [0, 1]$, assigning to each interval $[a, b]$ in Ω a probability equal to its *length* $b - a$. For unions of disjoint intervals, it takes the sum of their lengths. Additionally, we require that $P(X) = P(\{x + c \pmod 1 \mid x \in X\})$ for every set X and real number c. These are very reasonable assumptions for a probability distribution, as for instance satisfied by the uniform distribution introduced in Example 2.9. Note that $P(\Omega) = 1$, and that for unions of disjoint intervals the requirement of countable additivity is fulfilled; indeed, for all sets in the Borel σ-algebra, everything works fine.

Now consider an equivalence relation R over Ω equating all real numbers x, y such that $x - y \in \mathbb{Q}$. A Vitali set A is then obtained by choosing precisely one value from each equivalence class of R. To see why A cannot be given a probability, consider the family of sets defined by $A_q = \{a + q \pmod 1 \mid a \in A\}$ for every $q \in Q$. Since the sets A_q form a partitioning of $[0, 1]$, countable additivity prescribes that $\sum_{q \in \mathbb{Q}} P(A_q) = P([0, 1]) = 1$. Because each A_q is identical to A except for a translation, we find that $P(A_q) = P(A)$ for every $q \in \mathbb{Q}$ and hence $\sum_{q \in \mathbb{Q}} P(A) = 1$. This is clearly impossible, since an infinite sum of a nonnegative constant is either 0 or ∞. Therefore, no probability can be assigned to the set A under the restrictions of P, and indeed it is not present in the Borel σ-algebra $\mathcal{B}$ that can be used for such measures.

We can now compute, for instance,

$$\Pr(X = \mathit{odd}) = P(\{\omega \in \Omega \mid X(\omega) = \mathit{odd}\}) = P(\{1,3,5\}) = \frac{1}{2} \qquad \square$$

The question may rise why to use this additional layer, and not just directly work with probability spaces. Although that indeed would be possible, the main argument is that we may be interested in several different *observations* based on a probabilistic experiment. For instance, whereas someone may be interested in the parity of the outcome of a die, as given by the random variable in Example 2.5, someone else may be interested in the fact whether or not it is a 5 or higher. Then, the same probability space may be reused, just defining an additional random variable over it. This corresponds to the philosophical argument that 'chance' only does its work once, and that we may have several different observers of it.

Conditional probabilities. A well-known concept when dealing with random variables is the *conditional probability*: the probability that a certain event E_1 occurs, given that we know that an event E_2 occurs. It is defined by

$$\Pr(X \in E_1 \mid X \in E_2) = \frac{\Pr(X \in E_1 \cap E_2)}{\Pr(X \in E_2)}$$

Example 2.6. Again considering the probability space defined in Example 2.4, let $X\colon \Omega \to S$ be given by $X(\omega) = \omega$ (i.e., $S = \{1,2,3,4,5,6\}$ as well). Then, obviously, we find that $\Pr(X = \omega) = \frac{1}{6}$ for every $\omega \in \Omega$. Given that someone tells us that an odd number was rolled, we can compute the probability that a 1 or 2 was rolled:

$$\begin{aligned}\Pr(X \in \{1,2\} \mid X \in \{1,3,5\}) &= \frac{\Pr(X \in \{1,2\} \cap \{1,3,5\})}{\Pr(X \in \{1,3,5\})} \\ &= \frac{\Pr(X = 1)}{\Pr(X \in \{1,3,5\})} = \frac{\frac{1}{6}}{\frac{1}{2}} = \frac{1}{3} \qquad \square\end{aligned}$$

Distribution functions for random variables. Often, it is more convenient to specify the probabilities for the outcomes of a random variable directly, embracing the underlying probability space and basically forgetting about it. Traditionally, for this to be specified feasibly, random variables are classified to be either *discrete* (having a countable domain) or *continuous* (having an uncountable real-valued domain). Although in general more complicated random variables are possible, we restrict to these two basic cases.

From now on, we do not refer to underlying probability spaces anymore. Also, we will use the common standard of referring to the domain of the random variables we consider as their *sample spaces.*

2.2.3 Discrete probability theory

A discrete random variable has a countable sample space S, and a corresponding σ-algebra of events $\mathscr{P}(S)$. Its behaviour can easily be specified by a *discrete probability distribution* (often just called a *probability distribution* or *distribution*) over S. Such a distribution is a function $\mu\colon S \to [0,1]$ such that $\sum_{s\in S} \mu(s) = 1$. That is, it assigns a *probability* to the occurrence of each possible outcome in S in such a way that the total probability is 1. The behaviour of X is then simply given by $\Pr(X = s) = \mu(s)$ and $\Pr(X \in S') = \sum_{s\in S'} \mu(s')$, for $s \in S$ and $S' \subseteq S$.

Sometimes, we use the notation $\mu = \{s_1 \mapsto p_1, s_2 \mapsto p_2, \ldots, s_n \mapsto p_n\}$ to denote that $\mu(s_1) = p_1$, $\mu(s_2) = p_2$, …, $\mu(s_n) = p_n$. Also, we often write $\mu(S')$ as an abbreviation of $\sum_{s\in S'} \mu(s')$ if $S' \subseteq S$.

Example 2.7. The random variable from Example 2.6 is specified by the distribution $\mu\colon S \to [0,1]$ with $\mu(s) = \frac{1}{6}$ for every $s \in S$. □

Given a countable set S, we use $\mathrm{Distr}(S)$ to denote the (uncountable) set of all discrete probability distributions over S. We use $\mathrm{SDistr}(S)$ for the set of all *substochastic* discrete probability distributions over S, i.e., all functions $\mu\colon S \to [0,1]$ such that $\sum_{s\in S} \mu(s) \leq 1$.

Discrete probability distributions. For a discrete probability distribution μ over S and a function $f\colon S \to T$, we let $\mu_f \in \mathrm{Distr}(T)$ be the *lifting* of μ over f, i.e., $\mu_f(t) = \mu(f^{-1}(t))$ for all $t \in T$. Hence, μ_f assigns the same probability to an element $t \in T$ as μ assigns to the corresponding elements $s \in S$. We write $\mathbb{1}_s$ for the *Dirac distribution* over s, given by $\mathbb{1}_s(s) = 1$ (and hence $\mathbb{1}_s(t) = 0$ for every $t \in S$ such that $t \neq s$).

For any distribution μ over a countable set S, we write $supp(\mu) = \{s \in S \mid \mu(s) > 0\}$ for the *support* of μ. Given two distributions $\mu, \nu \in \mathrm{Distr}(S)$, we let $\mu \times \nu \in \mathrm{Distr}(S \times S)$ be their *joint distribution*, defined by $(\mu \times \nu)((s,t)) = \mu(s) \cdot \nu(t)$.

Given two probability distributions μ, μ' over a set S and an equivalence relation R over S, we write $\mu \equiv_R \mu'$ to denote that μ and μ' assign the same probability to each equivalence class of S under R. Formally, $\mu([s]_R) = \mu'([s]_R)$ for every $s \in S$. Obviously, $\equiv_R$ is an equivalence relation itself.

Example 2.8. Let $S = \{1, 2, 3, \ldots, 10\}$, and let R relate all odd and all even numbers, i.e.,

$$S/R = \{\{1,3,5,7,9\}, \{2,4,6,8,10\}\}$$

Also, let $\mu\colon S \to [0,1]$ and $\nu\colon S \to [0,1]$ be the distributions given by

$$\begin{array}{lll} \mu(1) = \dfrac{1}{2} & \mu(2) = \dfrac{1}{4} & \mu(3) = \dfrac{1}{4} \\ \nu(5) = \dfrac{1}{8} & \nu(9) = \dfrac{5}{8} & \nu(4) = \dfrac{1}{4} \end{array}$$

Then, $\mu \equiv_R \nu$, since both assign probability $\frac{3}{4}$ to the equivalence class $\{1, 3, 5, 7, 9\}$ and $\frac{1}{4}$ to $\{2, 4, 6, 8, 10\}$. □

2.2.4 Continuous probability theory

A continuous random variable has an uncountable real-valued sample space S, with the Borel σ-algebra as its set of events. It is often assumed that the probability of each individual outcome is 0, but that a probability *can* be assigned to an *interval* of the sample space. We adopt this assumption, and do not consider random variables with a more complicated structure that cannot be captured by the techniques discussed below.

A *probability density function* (often just called a *density function* or *density*) $f(x)\colon \mathbb{R} \to \mathbb{R}^{\geq 0}$ is employed to specify the probabilities of a continuous random variable X, in such a way that

$$\Pr(X \in [a,b]) = \Pr(X \in (a,b)) = \int_a^b f(x)\,dx$$

for every interval $[a,b] \subset \mathbb{R}$. As every set in the Borel σ-algebra can be constructed from intervals, this completely specifies X. Obviously, any density function should satisfy $\int_{-\infty}^{\infty} f(x)\,dx = 1$.

Example 2.9. Consider a web service that is available to its customers during a 200 minutes time interval. Each customer will access it at some point during this period, but will choose its timing randomly (in the sense that arriving within every equally sized interval in $[0,200]$ is equally likely). Since time is continuous, the probability of access at a specific time (for instance, $t = 83.6$) is 0, as there are infinitely many other times. However, the probability of access within the first 50 minutes is $\frac{1}{4}$.

This behaviour can be described by a random variable W with probability density function

$$f(x) = \begin{cases} 0.005 & \text{if } 0 \leq x \leq 200 \\ 0 & \text{otherwise} \end{cases}$$

In general, this is called the *uniform distribution* over the interval $[0,200]$, and W is said to be distributed *uniformly.*

Indeed, $\Pr(W \in [0,50]) = \int_0^{50} f(x)\,dx = [0.005x]_0^{50} = 0.005 \cdot 50 = 0.25.$ □

It is often more convenient to use a *cumulative distribution function*: a function $F\colon \mathbb{R} \to [0,1]$ assigning to each value $x \in \mathbb{R}$ the probability that the output of the experiment is *at most* equal to x. Hence,

$$F(x) = \Pr(X < x) = \Pr(X \in (-\infty, x)) = \int_{-\infty}^{x} f(t)\,dt$$

Example 2.10. For the density function $f(x)$ of the previous example, we find for $0 \leq x \leq 200$ that

$$F(x) = \int_{-\infty}^{x} f(t)\,dt = \int_{-\infty}^{0} 0\,dt + \int_0^x 0.005\,dt = 0 + [0.005t]_0^x = 0.005x$$

Hence, it is easy to see that

$$F(x) = \begin{cases} 0 & \text{if } x < 0 \\ 0.005x & \text{if } 0 \leq x \leq 200 \\ 1 & \text{if } x > 200 \end{cases}$$

Indeed, $F(50) = 0.005 \cdot 50 = 0.25$, as before. □

Expected values. The *expected value* $E(X)$ of a continuous random variable X is the weighted average of all possible values it can attain. It can be computed by the integral $E(X) = \int_{-\infty}^{\infty} x f(x)\, dx$.

Example 2.11. For the web service described in the previous example, we find

$$E(W) = \int_{-\infty}^{\infty} x f(x)\, dx = \int_{0}^{200} 0.005x\, dx = [0.0025x^2]_0^{200} = 0.0025 \cdot 200^2 = 100$$

Hence, as was to be expected, the average arrival of a customer who arrives uniformly within the interval [0,200] is at time 100. □

Remark 2.12. The expected value of a random variable does not necessarily exist. Like integrals in general, it may be infinite or even undefined (due to oscillation). As an example of a random variable X with expected value ∞, consider the probability density function

$$g(x) = \begin{cases} \frac{1}{x^2} & \text{if } x \geq 1 \\ 0 & \text{otherwise} \end{cases}$$

Indeed, as required

$$\Pr(-\infty \leq X \leq \infty) = \int_{-\infty}^{\infty} g(x)\, dx = \int_{1}^{\infty} \frac{1}{x^2}\, dx = \left[-\frac{1}{x}\right]_1^{\infty} = 1$$

However, we find that

$$E(X) = \int_{-\infty}^{\infty} x g(x)\, dx = \int_{1}^{\infty} \frac{1}{x}\, dx = [\ln x]_1^{\infty} = \infty$$ □

The exponential distribution. The most important continuous probability distribution for this work is the *exponential distribution.* It is parameterised by a value $\lambda > 0$, and given by either the probability density function $f(x)$ below left, or alternatively the cumulative distribution function $F(x)$ below right:

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \qquad F(x) = \begin{cases} 1 - e^{-\lambda x} & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

The parameter λ of an exponentially distributed random variable X dictates its *rate*, which is the reciprocal of its expected value:

$$E(X) = \int_{-\infty}^{\infty} x\lambda e^{-\lambda x}\,dx = \lambda \int_{0}^{\infty} xe^{-\lambda x}\,dx = \frac{1}{\lambda}$$

where the last step is obtained using integration by parts. Hence, the larger λ, the smaller the expected outcome of an exponentially distributed experiment. Note that each exponentially distributed random variable has a finite expected value (since we disallow $\lambda = 0$).

The exponential distribution is the only *memoryless* continuous probability distribution, as explained in the following example.

Example 2.13. The duration of telephone calls is often assumed to be exponentially distributed, with its parameter λ the average duration of a call. The probability that a call takes at least t time units is easily computed by $1 - F(t) = e^{-\lambda t}$. Now, given that a call has already taken s time units, the probability that it will take at most an additional t time units is computed as follows (where T is an exponentially distributed random variable):

$$\Pr(T > s+t \mid T > s) = \frac{\Pr(T > s+t)}{\Pr(T > s)} = \frac{e^{-\lambda(t+s)}}{e^{-\lambda s}} = e^{-\lambda t}$$

Hence, the duration someone was already on the phone does not influence the probability that the call will take an additional t time units. This is called the *memoryless property* of the exponential distribution. □

It is well known that the *minimum* of two independent exponentially distributed random variables X, Y with rates λ_1 and λ_2 is again an exponentially distributed random variable Z, with rate $\lambda_1 + \lambda_2$. To see why, we compute the probability that $\min(X, Y)$ yields a value larger than t:

$$\begin{aligned}\Pr(\min(X,Y) > t) &= \Pr(X > t \wedge Y > t)\\ &= \Pr(X > t) \cdot \Pr(Y > t)\\ &= e^{-\lambda_1 t} \cdot e^{-\lambda_2 t}\\ &= e^{-(\lambda_1+\lambda_2)t} = \Pr(Z > t)\end{aligned}$$

where the second step follows by independence. Since these probabilities coincide for $\min(X, Y)$ and Z, their cumulative distribution functions also coincide. This immediately implies that indeed $\min(X, Y) = Z$. This naturally generalises to the observation that $\min(X_1, X_2, \ldots, X_n) = Z'$, with Z' an exponentially distributed random variable with a rate equal to the sum of the rates of the constituents $X_1, X_2, \ldots, X_n$.

On the other hand, the maximum of two exponentially distributed random variables is not exponentially distributed, and neither is their sum.

CHAPTER 3

Modelling with Automata

"*One thing I have learned in a long life: that all our science, measured against reality, is primitive and childlike—and yet it is the most precious thing we have.*"

Albert Einstein

AUTOMATA are among the most commonly used models in computer science. They represent system behaviour by means of a set of *states* and a set of *transitions* between these states. States are often associated with *atomic propositions*, characterising the system at a certain moment during its execution and allowing them to be identified when verifying properties over the model. Transitions are often labelled by *actions*, indicating events that cause the system to evolve from one state to another. Automata-theoretical models have been extended to incorporate nondeterminism, timing, probability and stochasticity.

In this chapter, we introduce the model acted on by most of our techniques: the *Markov automaton.* Markov automata are among the most general automata-based models in computer science, as they subsume labelled transition systems, discrete-time Markov chains, continuous-time Markov chains, probabilistic automata and interactive Markov chains. Hence, they feature nondeterminism, discrete probabilistic choice as well as Markovian rates, and can be used to specify a wide variety of systems and protocols. Figure 3.1 visualises the subsumption relations between the Markov automaton and all models it generalises.

Markov automata are of vital importance in this thesis, as our process-algebraic specification language MAPA has its semantics defined in terms of Markov automata. Therefore, we discuss the structure of these models, their

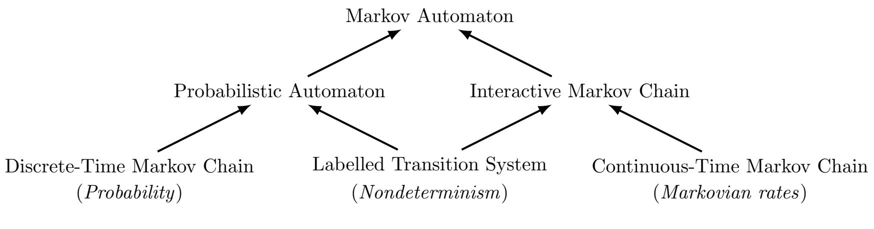


Figure 3.1: Subsumption relations between automata-based models.

interpretation and several important notations for working with them. Additionally, all our reduction techniques will be proven correct by demonstrating that the underlying Markov automaton of an original specification and its reduced counterpart are equivalent. In this chapter, we introduce the behavioural equivalences that will be used in this context: isomorphism, strong bisimulation and (divergence-sensitive) branching bisimulation.

Organisation of the chapter. Section 3.1 provides an informal overview of automata for system specification, followed by a precise coverage of Markov automata in Section 3.2. The latter section also briefly explains how Markov automata can be restricted to obtain probabilistic automata or interactive Markov chains. Then, Section 3.3 introduces isomorphism, strong bisimulation and branching bisimulation, and discusses the properties of these notions. Section 3.4 concludes by summarising the contributions of this chapter.

3.1 Automata for modelling system behaviour

Many different types of automata can be used to represent behaviour. Traditionally in process algebra, *labelled transition systems* (LTSs) were the model of choice to provide the semantics of specifications [Fok07]. These LTSs consist of anonymous *states*, in which a system can reside. Whereas the states themselves are not observable, interaction with the environment takes place via action-labelled *transitions* between those states.

Example 3.1. As an example of an LTS, consider the one depicted in Figure 3.2(a). It represents the behaviour of a coffee machine, that initially (indicated by the incoming arrow without source) requires the user to insert money and then choose between coffee and tea. Afterwards, the correct beverage is provided, and the machine returns to its initial state. □

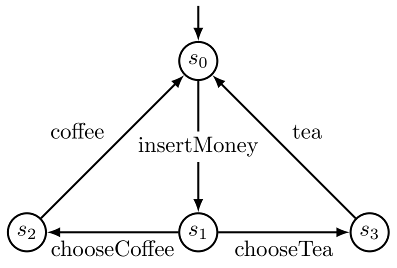
Note that after inserting the money, there are two different ways of continuing: either the *chooseCoffee* transition or the *chooseTea* transition is taken. This kind of behaviour, having a choice between differently labelled transitions, is called *external nondeterminism.* It is even allowed to have multiple equally labelled outgoing transitions from a state; this kind of behaviour is called *internal nondeterminism.* Often, *nondeterminism* refers to either type of behaviour.

Several generalisations of LTSs appeared over the last decades. For us, most important are the probabilistic automaton [Seg95], the interactive Markov chain [Her02] and the Markov automaton [EHZ10b, EHZ10a]. Probabilistic automata generalise LTSs by adding probabilistic choices, whereas interactive Markov chains generalise them by adding exponentially distributed delays. A Markov automaton has both additions, and hence unites the two models.

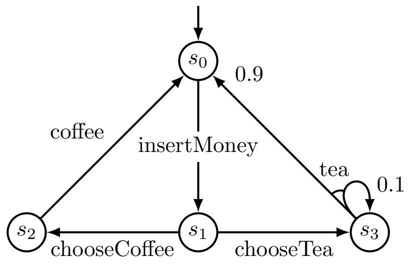
Before going into any details, we first informally introduce all three models.

3.1.1 Informal overview

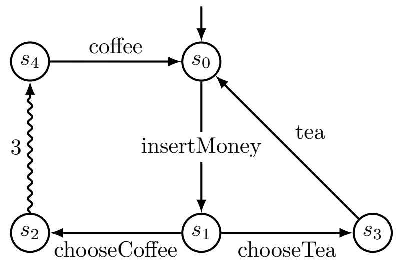
Probabilistic automata. Probabilistic automata (PAs) are similar to LTSs, except that the destination of a transition may be uncertain: it is decided



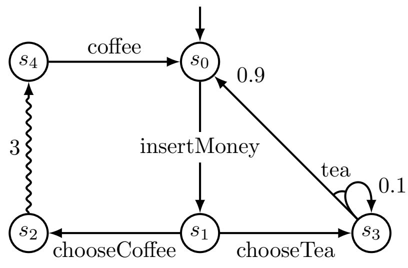
(a) LTS: a coffee machine.



(b) PA: a half-broken coffee machine.



(c) IMC: a delayed coffee machine.



(d) MA: a delayed half-broken coffee machine.

Figure 3.2: Four kinds of models of a coffee machine.

probabilistically [Seg95]. Hence, even if the nondeterministic choices are resolved in some deterministic manner, the behaviour of the system may still vary per execution.

Example 3.2. Figure 3.2(b) shows an example of a PA, modelling a half-broken version of the coffee machine of Example 3.1. After providing tea, the system might forget its action (with probability 0.1) and provide tea again. □

Although some part of a PA's behaviour is probabilistic, there may be nondeterminism present. Hence, we cannot immediately speak about *the* probability of the occurrence of a certain path. After all, this depends on how the nondeterministic choices are resolved. We will see how *schedulers* do precisely this.

Interactive Markov chains. Interactive Markov chains (IMCs) are also similar to LTSs, except that they have an additional type of transitions: Markovian transitions [Her02]. These transitions are each associated with a rate, and their duration is determined by an exponentially distributed random variable with that rate as its parameter.

Example 3.3. Figure 3.2(c) shows an example of an IMC, modelling a timed version of the coffee machine of Example 3.1. It has a delay when selecting coffee, governed by an exponential rate of 3. Hence, the probability of taking this transition within t time units is $1 - e^{-3t}$. □

Markov automata. Markov automata (MAs) combine the forces of PAs and IMCs, allowing nondeterministic choices, probabilistic choices and exponentially distributed delays [EHZ10b, EHZ10a]. Hence, paths through an MA have a probability of occurring (given a scheduler) as well as an expected duration.

Example 3.4. Figure 3.2(d) shows an example of a MA, combining the properties of the coffee machines from Examples 3.2 and 3.3. □

MAs allow for verifying several types of properties, such as the expected time to reach a certain state, the long-run average of residing in a subset of the states or the probability of executing a certain action within a given amount of time [GHH$^+$13a]. Due to possible nondeterministic choices, all these quantities can only be computed given a particular scheduler. In practice, one often computes the *minimal* and *maximal* values, ranging over all possible schedulers. We go into more details on the analysis side of MAs in Section 9.2.

3.2 Markov automata

Formally, we define a Markov automaton as a 7-tuple. It consists of a set of states, an initial state, an alphabet of actions, sets of action-labelled (interactive) and rate-labelled (Markovian) transitions, and a set of atomic propositions together with a state-labelling function. We assume a countable universe of actions *Act*, containing an internal invisible action $\tau \in \mathit{Act}$.

We note that the common definitions of MAs from literature [EHZ10b, EHZ10a, DH11] do not contain state labels; however, for model checking this often comes in handy. Hence, we take a very general approach and allow MAs to have both action labels and state labels. Our framework can thus be applied in both an action-based and a state-based manner, by either having no state labels or by having only one action label.

As we want to allow data with possibly infinite domains in our process algebra over MAs, we allow countable state spaces. Although this is problematic for the complicated notion of weak bisimulation from [EHZ10b], it does not hinder us.

Definition 3.5 (Markov automata). *A* Markov automaton (MA) *is a tuple* $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L\rangle$, *where*

- S *is a countable set of* states*;*
- $s^0 \in S$ *is the* initial state*;*
- $A \subseteq \mathit{Act}$ *is a countable set of* actions*;*
- $\hookrightarrow\, \subseteq S \times A \times \mathrm{Distr}(S)$ *is a countable* interactive probabilistic transition relation*;*
- $\rightsquigarrow\, \subseteq S \times \mathbb{R}^{>0} \times S$ *is a countable* Markovian transition relation*;*
- AP *is a countable set of* state labels *(also called* atomic propositions*);*
- $L \colon S \to \mathscr{P}(\mathrm{AP})$ *is the* state-labelling function.

If $(s, a, \mu) \in \hookrightarrow$, *we write* $s \xhookrightarrow{a} \mu$ *and say that the action a can be* executed *from state s, after which the probability to go to $s' \in S$ is $\mu(s')$. If* $(s, \lambda, s') \in \rightsquigarrow$, *we write* $s \overset{\lambda}{\rightsquigarrow} s'$ *and say that s moves to s' with rate λ.*

We require $\sum \{\!|\lambda \in \mathbb{R}^{>0} \mid \exists s' \in S \,.\, s \overset{\lambda}{\rightsquigarrow} s'|\!\} < \infty$ *for every state* $s \in S$.

We demand a finite exit rate for every state, as forced by the last requirement of the definition above. After all, given a state s with infinite exit rate, there is no obvious probability distribution for the next state of s. Also, if all states reachable from s were considered equivalent by a bisimulation relation (defined in Section 3.3), the bisimulation quotient would be ill-defined as it would yield a Markovian transition with rate ∞—which is not allowed. Fortunately, restricting to finite exit rates is no severe limitation; it still allows infinite chains of states connected by finite rates, as often seen in queueing systems. Also, it still allows infinite branching with for instance rates $\frac{1}{2}\lambda$, $\frac{1}{4}\lambda$, $\frac{1}{8}\lambda$, $\dots$.

Example 3.6. The MA depicted in Figure 3.2(d) is formally given by the tuple $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L\rangle$, with

$$\begin{aligned}
S &= \{s_0, s_1, s_2, s_3, s_4\};\\
s^0 &= s_0;\\
A &= \{\mathit{insertMoney}, \mathit{chooseTea}, \mathit{chooseCoffee}, \mathit{tea}, \mathit{coffee}\};\\
\hookrightarrow &= \{(s_0, \mathit{insertMoney}, \mathbb{1}_{s_1}), (s_1, \mathit{chooseCoffee}, \mathbb{1}_{s_2}), (s_1, \mathit{chooseTea}, \mathbb{1}_{s_3}),\\
&\qquad (s_3, \mathit{tea}, \{s_3 \mapsto 0.1, s_0 \mapsto 0.9\}), (s_4, \mathit{coffee}, \mathbb{1}_{s_0})\};\\
\rightsquigarrow &= \{(s_2, 3, s_4)\}.
\end{aligned}$$

Although not indicated in the picture, we may take $\mathrm{AP} = \{\mathit{paid}, \mathit{chosen}\}$ and

$$L(s_0) = \varnothing \quad L(s_1) = \{\mathit{paid}\} \quad L(s_2) = L(s_3) = L(s_4) = \{\mathit{paid}, \mathit{chosen}\} \qquad \square$$

3.2.1 Interpretation

Like an LTS, an MA moves between states via its transitions (some of which may feature a probabilistic choice). Each state may allow several transitions, both interactive and Markovian. Like IMCs, MAs adhere to the *maximal progress assumption* [Her02], prescribing τ-transitions to never be delayed (as they are assumed to be infinitely fast). Hence, no Markovian transition can ever be taken from a state that also has at least one outgoing τ-transition.

Traditionally, an interpretation was only given for *closed* MAs (or IMCs), i.e., systems obtained after parallel composition and after hiding of all actions (renaming them to τ). In such a setting, each state is either

Interactive: it has one or more outgoing τ-transitions,
Markovian: it has only outgoing Markovian transitions, or
Deadlock: it has no outgoing transitions at all.

Then, the interpretation of an MA is as follows:

- In each interactive state, a nondeterministic choice is made between the enabled τ-transitions. We discuss below how schedulers are used to resolve these choices. When an interactive transition is chosen, its discrete probability distribution determines the next state.

- In each Markovian state, there is a *race* between the outgoing Markovian transitions. The *rate* between two states $s, s' \in S$ (denoted by $\mathit{rate}(s, s')$) and the total *outgoing rate of* s (denoted by $\mathit{rate}(s)$) are given by

$$\mathit{rate}(s, s') = \sum_{(s,\lambda,s') \in \rightsquigarrow} \lambda \qquad\qquad \mathit{rate}(s) = \sum_{s' \in S} \mathit{rate}(s, s')$$

  Each outgoing Markovian transition $s \overset{\lambda}{\rightsquigarrow} s'$ potentially fires after a delay governed by an exponentially distributed random variable with rate λ—i.e., an experiment is conducted to obtain this delay, in such a way that the probability of obtaining a value below x is $1 - e^{-\lambda x}$. The transition whose delay turns out to be shortest 'wins the race', and is taken.

  Since the minimum of a number of exponentially distributed random variables is exponentially distributed with a rate equal to the sum of the individual rates (as discussed in Section 2.2.4), the time of staying in a state s—sometimes called its *sojourn time*—is exponentially distributed with rate $\mathit{rate}(s)$. Hence, the probability of leaving state s within t time units is given by $1 - e^{-\mathit{rate}(s) \cdot t}$. Also, it follows easily that the probability of going to a state s', after waiting for some time in state s, is

$$\mathbb{P}_s(s') = \frac{\mathit{rate}(s, s')}{\mathit{rate}(s)}$$

  We denote by $\mathbb{P}_s$ this *branching probability distribution* of state s.

Remark 3.7. Due to the properties of the exponential distribution, we can assume without loss of generality that there is at most one Markovian transition between each pair of states s, s'. After all, two transitions $s \overset{\lambda'}{\rightsquigarrow} s'$ and $s \overset{\lambda''}{\rightsquigarrow} s'$ can be combined into one transition $s \overset{\lambda}{\rightsquigarrow} s'$ with $\lambda = \lambda' + \lambda''$. Under this assumption, the sojourn time $\mathit{rate}(s)$ and the branching probability distribution $\mathbb{P}_s$ together can be used to completely reconstruct all outgoing Markovian transitions from state s: for each state $s' \in \mathit{supp}(\mathbb{P}_s)$, there is a Markovian transition $s \overset{\lambda}{\rightsquigarrow} s'$ with $\lambda = \mathbb{P}_s(s') \cdot \mathit{rate}(s)$. □

Example 3.8. Consider the MA shown in Figure 3.3(a). Due to the maximal progress assumption, the Markovian transition from state s_0 is irrelevant. Hence, a scheduler may choose to either select the τ-transition to s_1 or the one to s_3. In state s_3, no interactive transitions are enabled: it is a Markovian state. We find $\mathit{rate}(s_3) = 3 + 4 = 7$, and thus the probability of leaving s_3 within t time units is $1 - e^{-7t}$. Then, the probability of going to s_2 is $\mathbb{P}_{s_3}(s_2) = \frac{3}{7}$. □

Interpretation of open Markov automata. For *open* systems, in which some visible actions are present, the precise behaviour of the system still depends on the environment. After all, although τ-transitions can always happen immediately, the timing of visible transitions is assumed to depend on the environment being ready to synchronise on them. Hence, they only get real meaning in a context.

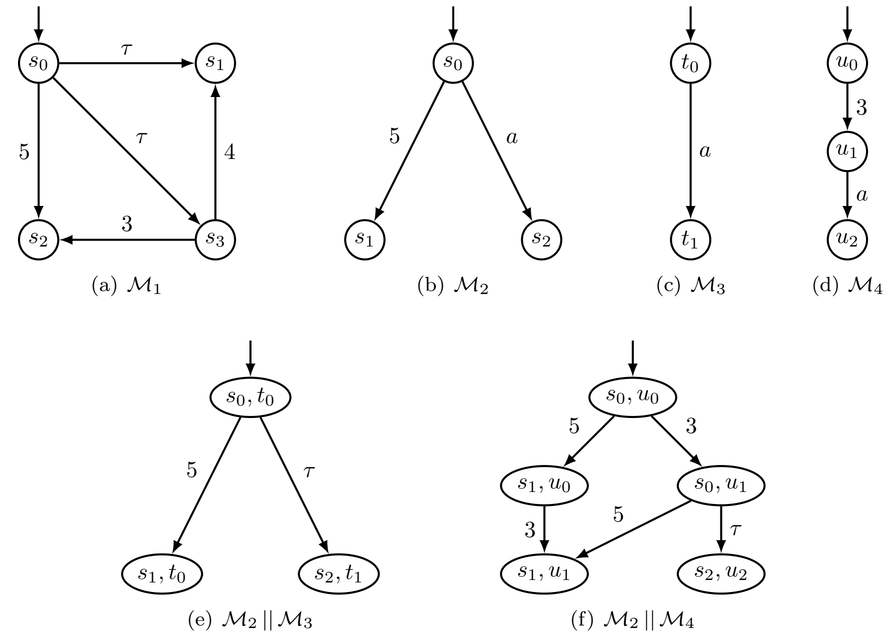

s_0 s_1 s_2 s_3 τ τ 5 4 3
(a) $\mathcal{M}_1$
s_0 s_1 s_2 5 a
(b) $\mathcal{M}_2$
t_0 t_1 a
(c) $\mathcal{M}_3$
u_0 u_1 u_2 3 a
(d) $\mathcal{M}_4$
s_0, t_0 s_1, t_0 s_2, t_1 5 τ
(e) $\mathcal{M}_2 \,||\, \mathcal{M}_3$
s_0, u_0 s_1, u_0 s_0, u_1 s_1, u_1 s_2, u_2 5 3 3 5 τ
(f) $\mathcal{M}_2 \,||\, \mathcal{M}_4$

Figure 3.3: Interpretation of MAs.

Example 3.9. Consider the MA shown in Figure 3.3(b). State s_0 enables both an interactive transition and a Markovian transition. By itself, its behaviour is still unclear. So, we put it in parallel to an environment, forcing them to synchronise on the a-action and hiding this action in the parallel composition[1].

First, we put it in parallel to a system immediately providing an a-action (Figure 3.3(c)). The resulting parallel composition in shown in Figure 3.3(e). Due to maximal progress, the Markovian transition will never fire.

Second, we put it in parallel to a system first having a delay and only then synchronising on the a-action (Figure 3.3(d)). The parallel composition of these systems is shown in Figure 3.3(f). Now, the system ends up in s_1 or s_2, depending on which Markovian transition fires first. □

An open system could behave as any closed system that can be obtained by placing it in parallel to any environment over the same alphabet and subsequently hiding all actions. In our current treatment we chose to in principle consider open MAs. Although the precise behaviour of such a system still depends on its context, that turns out not to be a problem. After all, we are only concerned with transforming MAs into smaller MAs while preserving either strong or branching bisimulation—equivalences that have been defined on open MAs.

Only when starting to model check the MAs resulting from our specifications, we assume all actions to be hidden, as all model checking techniques for MAs

[1]Parallel composition is defined formally in Section 3.2.3.

introduced thus far require closed systems [GHH+13a]. An interesting recent approach is to verify open IMCs, obtaining compositionality properties by means of timed games [BHK+12]. A generalisation of these techniques to MAs does not yet exist.

3.2.2 Behavioural notions

Extended transitions. The fact that there are two types of transitions—interactive and Markovian—is often rather inconvenient. Hence, the concept of extended transitions was introduced to combine them into a single transition relation and treat them in a uniform way [EHZ10b].

Each extended transition has a source state, an action label and a probability distribution, just like an interactive transition. Hence, each interactive transition can be lifted to an extended transition without any difficulty. Furthermore, for every state s with at least one outgoing Markovian transition, all these transitions are combined into a single extended transition $s \xrightarrow{\chi(\mathit{rate}(s))} \mathbb{P}_s$. The dedicated action label χ indicates that this extended transition is Markovian, and its parameter $\mathit{rate}(s)$ and the probability distribution $\mathbb{P}_s$ together determine all Markovian transitions from state s (as discussed in Remark 3.7).

The maximal progress assumption is incorporated in the definition of extended actions, only allowing a Markovian extended transition from states without any outgoing τ-transitions.

Definition 3.10 (Extended action set[2]). *Let* $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L\rangle$ *be an MA, then the* extended action set of $\mathcal{M}$ *is*

$$A^\chi = A \cup \{\chi(r) \mid r \in \mathbb{R}^{>0}\}$$

Given a state $s \in S$ and an action $\alpha \in A^\chi$, we write $s \xrightarrow{\alpha} \mu$ if either

- $\alpha \in A$ *and* $s \xhookrightarrow{\alpha} \mu$, *or*
- $\alpha = \chi(\mathit{rate}(s))$, $\mathit{rate}(s) > 0$, $\mu = \mathbb{P}_s$ *and there is no* μ' *such that* $s \xhookrightarrow{\tau} \mu'$.

A transition $s \xrightarrow{\alpha} \mu$ is called an extended transition*. We use $s \xrightarrow{\alpha} t$ as shorthand for $s \xrightarrow{\alpha} \mathbb{1}_t$, and write $s \rightarrow t$ if there is at least one action α such that $s \xrightarrow{\alpha} t$. We write $s \xrightarrow{\alpha,\mu} s'$ if there is an extended transition $s \xrightarrow{\alpha} \mu$ such that $\mu(s') > 0$.*

Note that each state has an extended transition per interactive transition, while it has only one extended transition for all Markovian transitions together (if there are any).

Example 3.11. Reconsider the MA in Figure 3.3(a). There are three extended transitions: $s_0 \xrightarrow{\tau} \mathbb{1}_{s_1}$, $s_0 \xrightarrow{\tau} \mathbb{1}_{s_3}$ and $s_3 \xrightarrow{\chi(7)} \{s_1 \mapsto \frac{4}{7}, s_2 \mapsto \frac{3}{7}\}$. There is no Markovian extended transition from s_0, as it has τ-transitions. □

[2]Instead of first defining MAs in terms of the interactive transition relation $\hookrightarrow$ and the Markovian transition relation $\rightsquigarrow$, we could also have chosen to define MAs immediately based on extended actions. For historical reasons, we chose not to do so.

Paths, traces and connectivity. As for LTSs and other types of automata, we can define paths and traces through an MA. We define these concepts, and provide notations to work with them. Basically, a path is just a traversal through an MA, indicating which transitions were taken and which states were chosen probabilistically. Paths abstract from timing: they do not contain any information on the precise duration of these steps. As paths are based on extended transitions, they may contain interactive as well as Markovian steps. Traces are obtained from paths by only listing the observable aspects of a path: the state labels and the visible action labels.

Definition 3.12 (Paths and traces). *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L\rangle$ be an MA. Then,*

- *A* path *in $\mathcal{M}$ is a finite sequence*

$$\pi^{\mathrm{fin}} = s_0 \xrightarrow{\alpha_1,\mu_1} s_1 \xrightarrow{\alpha_2,\mu_2} s_2 \xrightarrow{\alpha_3,\mu_3} \dots \xrightarrow{\alpha_n,\mu_n} s_n,$$

possibly with $n = 0$, or an infinite sequence

$$\pi^{\mathrm{inf}} = s_0 \xrightarrow{\alpha_1,\mu_1} s_1 \xrightarrow{\alpha_2,\mu_2} s_2 \xrightarrow{\alpha_3,\mu_3} \dots,$$

with $s_i \in S$ for all $0 \leq i \leq n$ and all $i \geq 0$, respectively, and such that each extended transition $s_i \xrightarrow{\alpha_{i+1},\mu_{i+1}} s_{i+1}$ is indeed present in $\mathcal{M}$. We refer to π^{fin} as a path from s_0 to s_n.
- *For any path π as above, we define*

$$\mathit{trace}(\pi) = L(s_0)\alpha_1 L(s_1)\alpha_2 L(s_2)\alpha_3 \dots$$

to be its sequence of state labellings and action labels. Additionally, we write

$$\begin{aligned} \mathit{actionTrace}(\pi) &= \alpha_1\alpha_2\alpha_3 \dots \setminus \{\tau\} \\ \mathit{stateTrace}(\pi) &= L(s_0)L(s_1)L(s_2)\dots \end{aligned}$$

for the sequences of visible actions and state labellings of π, respectively.
- *Given an infinite or sufficiently long finite path π, we use prefix(π, i) to denote the path fragment $s_0 \xrightarrow{\alpha_1,\mu_1} \dots \xrightarrow{\alpha_i,\mu_i} s_i$, and step$(\pi, i)$ for the transition $s_{i-1} \xrightarrow{\alpha_i} \mu_i$. When π is finite we define $|\pi| = n$ and last$(\pi) = s_n$, otherwise $|\pi| = \infty$ and no final state exists.*
- *We use finpaths$_{\mathcal{M}}$ for the set of all* finite paths *in $\mathcal{M}$ (not necessarily beginning in the initial state), and finpaths$_{\mathcal{M}}(s)$ for all such paths with $s_0 = s$.*
- *We say that a path π is* invisible *(denoted by invisible(π)) if it never alters the state labelling and only consists of invisible actions. That is, actionTrace$(\pi) = \epsilon$ and stateTrace(π) is a sequence of identical labellings.*

Example 3.13. Again, consider the MA $\mathcal{M}$ in Figure 3.3(a). A possible path in $\mathcal{M}$ is

$$\pi = s_0 \xrightarrow{\tau,\mathbb{1}_{s_3}} s_3 \xrightarrow{\chi(7),\mu} s_1$$

with $\mu(s_1) = \frac{4}{7}$ and $\mu(s_2) = \frac{3}{7}$. We have $|\pi| = 2$, $\mathit{last}(\pi) = s_1$, and obviously $\pi \in \mathit{finpaths}_{\mathcal{M}}$ and $\pi \in \mathit{finpaths}_{\mathcal{M}}(s_0)$. Additionally, we find

$$\begin{aligned} \mathit{trace}(\pi) &= L(s_0)\tau L(s_3)\chi(7)L(s_1) \\ \mathit{actionTrace}(\pi) &= \chi(7) \\ \mathit{stateTrace}(\pi) &= L(s_0)L(s_3)L(s_1) \end{aligned}$$

As $\mathit{actionTrace}(\pi) \neq \epsilon$, the path π is visible, regardless of the state labelling. □

We now introduce the three most important connectivity relations: reachability, joinability and convertibility.

Definition 3.14 (Connectivity). *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L\rangle$ be an MA, $s, t \in S$, and consider again the binary relation $\rightarrow \subseteq S \times S$ from Definition 3.10 that relates states $s, t \in S$ if there is a transition $s \xrightarrow{\alpha} \mathbb{1}_t$ for some α.*

We let $\twoheadrightarrow$ (reachability) *be the reflexive and transitive closure of $\rightarrow$, and we let $\twoheadleftarrow\!\!\twoheadrightarrow$* (convertibility) *be its reflexive, transitive and symmetric closure. We write $s \twoheadrightarrow\twoheadleftarrow t$* (joinability) *if there is a state u such that $s \twoheadrightarrow u$ and $t \twoheadrightarrow u$.*

Note that the relation $\twoheadrightarrow\twoheadleftarrow$ is symmetric, but not necessarily transitive. Also note that, intuitively, $s \twoheadleftarrow\!\!\twoheadrightarrow t$ means that s is connected by extended transitions to t—disregarding the orientation of these transitions, but requiring them all to have a Dirac distribution. Clearly, $s \twoheadrightarrow t$ implies $s \twoheadrightarrow\twoheadleftarrow t$, and $s \twoheadrightarrow\twoheadleftarrow t$ implies $s \twoheadleftarrow\!\!\twoheadrightarrow t$. These implications do not hold the other way.

Example 3.15. In Figure 3.3(a), we find $s_0 \twoheadrightarrow s_2$, but not $s_2 \twoheadrightarrow s_0$. Since both $s_0 \twoheadrightarrow s_2$ and $s_3 \twoheadrightarrow s_2$, also $s_0 \twoheadrightarrow\twoheadleftarrow s_3$ (alternatively, they could join in s_3). Finally, $s_1 \twoheadleftarrow\!\!\twoheadrightarrow s_2$, since they are connected by s_3 (but not $s_1 \twoheadrightarrow\twoheadleftarrow s_2$). □

Schedulers. Although some part of an MA's behaviour is probabilistic, there may be nondeterminism present. Hence, we cannot immediately speak about the probability of the occurrence of a certain path. After all, this depends on the nondeterministic choices that are made. Therefore, we define schedulers to precisely resolve these choices (basing our notations on [Sto02a]).

Basically, a scheduler is a function defining for each finite path which transition to take next. The decisions of schedulers are allowed to be *randomised*, i.e., instead of choosing a single transition, a scheduler may resolve a nondeterministic choice by a probabilistic choice. Schedulers can also be *partial*, i.e., they may assign some probability to the decision of not choosing any next transition at all (and hence terminate). Our schedulers can select from interactive transitions as well as Markovian transitions, as both may be enabled at the same time. This is due to the fact that we consider *open* MAs, in which the timing of visible actions is still to be determined by their context (as discussed above in Section 3.2.1).

Definition 3.16 (Schedulers). *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L\rangle$ be an MA, and $\rightarrow \subseteq S \times A^\chi \times \mathrm{Distr}(S)$ its set of extended transitions. Then, a* scheduler *for $\mathcal{M}$ is a function*

$$\mathcal{S} \colon \mathit{finpaths}_{\mathcal{M}} \to \mathrm{Distr}(\{\bot\} \cup \rightarrow)$$

such that, for every $\pi \in \mathit{finpaths}_{\mathcal{M}}$, the transitions $s \xrightarrow{\alpha} \mu$ that are scheduled by $\mathcal{S}$ after π are indeed possible, i.e., $\mathcal{S}(\pi)(s, \alpha, \mu) > 0$ implies $s = \mathit{last}(\pi)$. The decision of not choosing any transition is represented by $\perp$.

Note that our schedulers are *time-homogeneous*, i.e., they cannot take into account the amount of time that already passed during a path. *Time-inhomogeneous* schedulers are important in some contexts, for instance, for defining the type of time-bounded reachability properties [HH12] that we also consider. However, in this work we can do without as we will not formally define such properties. Since time-homogeneous schedulers do take into account the rates of an MA, we can use them to define notions of bisimulation that do preserve time-bounded properties (similar to weak bisimulation in [EHZ10b] being defined in terms of 'time-homogeneous' labelled trees).

As discussed in [ZN10], measurability is not an issue for time-homogeneous schedulers. See [NSK09] for a thorough analysis of different types of schedulers. Since Markovian extended transitions only emanate from states without any outgoing τ-transitions, schedulers cannot violate the maximal progress assumption.

We now define finite and maximal paths of an MA given a scheduler. The finite paths given a scheduler are those finite paths of the MA for which each step has been assigned a nonzero probability. The maximal paths are again a subset of those; they are the paths after which the scheduler may decide to terminate.

Definition 3.17 (Finite and maximal paths). *Given an MA $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L\rangle$ and a scheduler $\mathcal{S}$ for $\mathcal{M}$, the set of* finite paths of $\mathcal{M}$ under $\mathcal{S}$ *is*

$$\mathit{finpaths}^{\mathcal{S}}_{\mathcal{M}} = \{\pi \in \mathit{finpaths}_{\mathcal{M}} \mid \forall 0 \leq i < |\pi| \,.\, \mathcal{S}(\mathit{prefix}(\pi, i))(\mathit{step}(\pi, i+1)) > 0\}$$

We define $\mathit{finpaths}^{\mathcal{S}}_{\mathcal{M}}(s) \subseteq \mathit{finpaths}^{\mathcal{S}}_{\mathcal{M}}$ as the set of all such paths starting in s.

The set of maximal paths of $\mathcal{M}$ under $\mathcal{S}$ *is given by*

$$\mathit{maxpaths}^{\mathcal{S}}_{\mathcal{M}} = \{\pi \in \mathit{finpaths}^{\mathcal{S}}_{\mathcal{M}} \mid \mathcal{S}(\pi)(\perp) > 0\}$$

Similarly, $\mathit{maxpaths}^{\mathcal{S}}_{\mathcal{M}}(s)$ is the set of maximal paths of $\mathcal{M}$ under $\mathcal{S}$ starting in s.

Example 3.18. Reconsider the MA $\mathcal{M}$ from Figure 3.2(d). For convenience, it is repeated in Figure 3.4.

A possible scheduler $\mathcal{S}$ for this system is given by

$$\mathcal{S}(\pi) = \begin{cases} \{(s_0, \mathit{insertMoney}, \mathbb{1}_{s_1}) \mapsto 1\} & \text{if } \mathit{last}(\pi) = s_0 \wedge |\pi| = 0 \\ \{\perp \mapsto 1\} & \text{if } \mathit{last}(\pi) = s_0 \wedge |\pi| > 0 \\ \{(s_1, \mathit{chooseCoffee}, \mathbb{1}_{s_2}) \mapsto 0.5, & \text{if } \mathit{last}(\pi) = s_1 \\ \ (s_1, \mathit{chooseTea}, \mathbb{1}_{s_3}) \mapsto 0.5\} & \\ \{(s_2, \chi(3), \mathbb{1}_{s_4}) \mapsto 1)\} & \text{if } \mathit{last}(\pi) = s_2 \\ \{(s_3, \mathit{tea}, \{s_3 \mapsto 0.1, s_0 \mapsto 0.9\}) \mapsto 1)\} & \text{if } \mathit{last}(\pi) = s_3 \wedge |\pi| \leq 2 \\ \{(s_3, \mathit{tea}, \{s_3 \mapsto 0.1, s_0 \mapsto 0.9\}) \mapsto 0.25, & \text{if } \mathit{last}(\pi) = s_3 \wedge |\pi| > 2 \\ \ \perp \mapsto 0.75)\} & \\ \{(s_4, \mathit{coffee}, \mathbb{1}_{s_0}) \mapsto 1)\} & \text{if } \mathit{last}(\pi) = s_4 \end{cases}$$

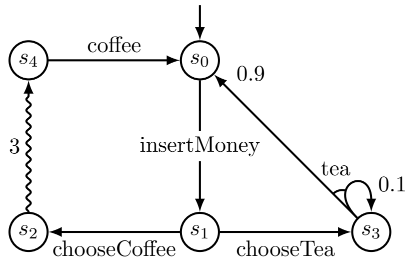


Figure 3.4: A delayed half-broken coffee machine.

This scheduler tries to order one beverage, and chooses randomly between coffee and tea. Upon returning to the initial state, it terminates. If it receives a tea without returning to the initial state, termination only happens with probability 0.75.

Given this scheduler, there are infinitely many finite and maximal paths starting from the initial state. We list part of the maximal paths, and note that the finite paths are these maximal paths as well as all their prefixes.

$$\begin{aligned}
\mathit{maxpaths}^{\mathcal{S}}_{\mathcal{M}}(s_0) = \{ & s_0 \xrightarrow{\mathit{iM},\mathbb{1}_{s_1}} s_1 \xrightarrow{cC,\mathbb{1}_{s_2}} s_2 \xrightarrow{\chi(3),\mathbb{1}_{s_4}} s_4 \xrightarrow{\mathit{coffee},\mathbb{1}_{s_0}} s_0, \\
& s_0 \xrightarrow{\mathit{iM},\mathbb{1}_{s_1}} s_1 \xrightarrow{cT,\mathbb{1}_{s_3}} s_3 \xrightarrow{\mathit{tea},\mu} s_0, \\
& s_0 \xrightarrow{\mathit{iM},\mathbb{1}_{s_1}} s_1 \xrightarrow{cT,\mathbb{1}_{s_3}} s_3 \xrightarrow{\mathit{tea},\mu} s_3, \\
& s_0 \xrightarrow{\mathit{iM},\mathbb{1}_{s_1}} s_1 \xrightarrow{cT,\mathbb{1}_{s_3}} s_3 \xrightarrow{\mathit{tea},\mu} s_3 \xrightarrow{\mathit{tea},\mu} s_0, \\
& s_0 \xrightarrow{\mathit{iM},\mathbb{1}_{s_1}} s_1 \xrightarrow{cT,\mathbb{1}_{s_3}} s_3 \xrightarrow{\mathit{tea},\mu} s_3 \xrightarrow{\mathit{tea},\mu} s_3, \dots \}
\end{aligned}$$

Here, we used *iM*, cC and cT as abbreviations for *insertMoney*, *chooseCoffee* and *chooseTea*, respectively. Also, we used $\mu = \{s_3 \mapsto 0.1, s_0 \mapsto 0.9\}$. □

We can now define the behaviour of an MA $\mathcal{M}$ under a scheduler $\mathcal{S}$. As schedulers resolve all nondeterministic choices, this behaviour is fully probabilistic. We can compute the probability that, starting from a given state s, the path generated by $\mathcal{S}$ has some finite prefix π. This probability is denoted by $P^{\mathcal{S}}_{\mathcal{M},s}(\pi)$.

Definition 3.19 (Path probabilities). *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L\rangle$ be an MA, $\mathcal{S}$ a scheduler for $\mathcal{M}$, and $s \in S$ a state of $\mathcal{M}$. Then, we define the function $P^{\mathcal{S}}_{\mathcal{M},s} \colon \mathit{finpaths}_{\mathcal{M}}(s) \to [0,1]$ by*

$$P^{\mathcal{S}}_{\mathcal{M},s}(s) = 1 \qquad P^{\mathcal{S}}_{\mathcal{M},s}(\pi \xrightarrow{\alpha,\mu} t) = P^{\mathcal{S}}_{\mathcal{M},s}(\pi) \cdot \mathcal{S}(\pi)(\mathit{last}(\pi), \alpha, \mu) \cdot \mu(t)$$

Based on these probabilities we can compute the (substochastic) probability distribution $F^{\mathcal{S}}_{\mathcal{M}}(s)$ over the states where an MA $\mathcal{M}$ under a scheduler $\mathcal{S}$ terminates, when starting in state s. Note that $F^{\mathcal{S}}_{\mathcal{M}}(s)$ is potentially substochastic (i.e., the probabilities do not add up to 1), as $\mathcal{S}$ does not necessarily terminate.

Definition 3.20 (Final state probabilities). *Given an MA $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L\rangle$ and a scheduler $\mathcal{S}$ for $\mathcal{M}$, we define $F^{\mathcal{S}}_{\mathcal{M}} \colon S \to \mathrm{SDistr}(S)$ by*

$$F^{\mathcal{S}}_{\mathcal{M}}(s) = \Big\{ s' \mapsto \sum_{\substack{\pi \in maxpaths^{\mathcal{S}}_{\mathcal{M}}(s) \\ last(\pi) = s'}} P^{\mathcal{S}}_{\mathcal{M},s}(\pi) \cdot \mathcal{S}(\pi)(\perp) \mid s' \in S \Big\} \qquad \forall s \in S$$

Example 3.21. We continue with Example 3.18. Since all maximal paths end in either state s_3 or state s_0, $F^{\mathcal{S}}_{\mathcal{M}}(s_0)$ will be a probability distribution over these two states.

For the scheduler $\mathcal{S}$ defined before, we compute $P^{\mathcal{S}}_{\mathcal{M},s_0}(\pi_i) \cdot \mathcal{S}(\pi_i)(\perp)$ for the following three paths (again with $\mu = \{s_3 \mapsto 0.1, s_0 \mapsto 0.9\}$):

$$\begin{aligned}
\pi_1 &= s_0 \xrightarrow{iM, \mathbb{1}_{s_1}} s_1 \xrightarrow{cC, \mathbb{1}_{s_2}} s_2 \xrightarrow{\chi(3), \mathbb{1}_{s_4}} s_4 \xrightarrow{coffee, \mathbb{1}_{s_0}} s_0 \\
\pi_2 &= s_0 \xrightarrow{iM, \mathbb{1}_{s_1}} s_1 \xrightarrow{cT, \mathbb{1}_{s_3}} s_3 \xrightarrow{tea, \mu} s_0 \\
\pi_3 &= s_0 \xrightarrow{iM, \mathbb{1}_{s_1}} s_1 \xrightarrow{cT, \mathbb{1}_{s_3}} s_3 \xrightarrow{tea, \mu} s_3
\end{aligned}$$

Using the definition of path probabilities, we find

$$\begin{aligned}
P^{\mathcal{S}}_{\mathcal{M},s_0}(\pi_1) \cdot \mathcal{S}(\pi_1)(\perp) &= (1 \cdot 1) \cdot (\tfrac{1}{2} \cdot 1) \cdot (1 \cdot 1) \cdot (1 \cdot 1) \cdot 1 = 0.5 \\
P^{\mathcal{S}}_{\mathcal{M},s_0}(\pi_2) \cdot \mathcal{S}(\pi_2)(\perp) &= (1 \cdot 1) \cdot (\tfrac{1}{2} \cdot 1) \cdot (1 \cdot 0.9) \cdot 1 = 0.45 \\
P^{\mathcal{S}}_{\mathcal{M},s_0}(\pi_3) \cdot \mathcal{S}(\pi_3)(\perp) &= (1 \cdot 1) \cdot (\tfrac{1}{2} \cdot 1) \cdot (1 \cdot 0.1) \cdot 0.75 = 0.0375
\end{aligned}$$

As π_3 ends in s_3, its path probability 0.0375 contributes to the total probability of terminating in s_3. To precisely obtain the probability of terminating in s_3, we need to sum the probabilities of all paths ending in s_3. Note that these paths always first have to take the *chooseTea* transition (probability 0.5). Then, the *tea* transition should be taken (probability 1) and it should fail (probability 0.1). Due to independence of these probabilities, this combination happens with probability $0.5 \cdot 1 \cdot 0.1 = 0.05$.

Then, either we immediately terminate (probability 0.75), or we terminate after any number of times continuing and failing again (probability $0.25 \cdot 0.1 = 0.025$ per time). So, this happens with probability

$$0.75 + 0.025 \cdot 0.75 + 0.025^2 \cdot 0.75 + \cdots = 0.75 \cdot \sum_{i=0}^{\infty} 0.025^i = \frac{0.75}{1 - 0.025} = \frac{10}{13}$$

Combining these results, the total probability of terminating in state s_3 is found to be $0.05 \cdot \frac{10}{13} = \frac{1}{26}$. In the same way, we can compute that the probability of terminating in s_0 is $\frac{25}{26}$. Hence,

$$F^{\mathcal{S}}_{\mathcal{M}}(s_0) = \left\{ s_0 \mapsto \frac{25}{26}, s_3 \mapsto \frac{1}{26} \right\}$$

□

3.2.3 Parallel composition

To enable modular system specification, MAs can be composed in parallel. This construction is a straightforward generalisation of the constructions for parallel composition of PAs and IMCs. As usual, each state of the parallel composition is a pair of states of the individual components.

Parallel processes by default interleave all their actions. Additionally, we assume a partial function $\gamma\colon A_1 \times A_2 \rightharpoonup A_1 \cup A_2$, that specifies which actions can communicate. More precisely, $\gamma(a, b) = c$ denotes that a and b can communicate, resulting in the action c (as in ACP [BK89]). Still, even if $(a, b) \in \mathit{dom}(\gamma)$, the component processes are also allowed to act individually. Clearly, it would just as well be possible to define parallel composition using CCS or CSP style synchronisations.

Due to the memoryless property of exponential distributions, parallel composition of Markovian transitions is also very straight-forward. Consider two interleaving processes, one having a transition with rate λ_1 and the other having a transition with rate λ_2. Then, the properties of the exponential distribution (as discussed in Section 2.2.4) tell us that together they work with rate $\lambda_1 + \lambda_2$. Hence, we can just take the union of the Markovian transitions of the parallel components. After for instance the first component took its transition, the memoryless property of the exponential distribution tells us that the other still has a rate of λ_2 to take its transition. This makes the MA (and before that the IMC) a very suitable model for parallel composition. For some easy examples, see Figure 3.3 on page 37.

Except for the addition of state labels and the slightly different manner of synchronisation, our definition of parallel composition coincides with the definition in [EHZ10b]. As already mentioned there, it also agrees with the standard definitions of parallel composition for PAs and IMCs if the MA does not contain any rates or probabilities, respectively.

Definition 3.22 (Parallel composition). *Given two MAs $\mathcal{M}_1 = \langle S_1, s_1^0, A_1, \hookrightarrow_1, \rightsquigarrow_1, \mathrm{AP}_1, L_1\rangle$ and $\mathcal{M}_2 = \langle S_2, s_2^0, A_2, \hookrightarrow_2, \rightsquigarrow_2, \mathrm{AP}_2, L_2\rangle$, and a partial communication function $\gamma\colon A_1 \times A_2 \rightharpoonup A_1 \cup A_2$, the parallel composition of $\mathcal{M}_1$ and $\mathcal{M}_2$ is the system $\mathcal{M}_1 \,||\, \mathcal{M}_2 = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L\rangle$, where*

- $S = S_1 \times S_2$;
- $s^0 = (s_1^0, s_2^0)$;
- $A = A_1 \cup A_2$;
- $\mathrm{AP} = \mathrm{AP}_1 \cup \mathrm{AP}_2$;
- $L((s_1, s_2)) = L_1(s_1) \cup L_2(s_2)$,

and $\hookrightarrow$ and $\rightsquigarrow$ are the smallest relations fulfilling the inference rules in Table 3.5 (i.e., if all conditions above the line of a rule hold, then so should the condition below the line). We use $\lambda(s_1, s_2)$ to denote $\mathit{rate}(s_1, s_1) + \mathit{rate}(s_2, s_2)$.

It may seem that we could just omit the last SOS rule and the inequality constraints of the rules above. However, if in that case $s_1 \overset{\lambda}{\rightsquigarrow} s_1$ and $s_2 \overset{\lambda}{\rightsquigarrow} s_2$, the parallel composition would have a transition $(s_1, s_2) \overset{\lambda}{\rightsquigarrow} (s_1, s_2)$. This would clearly be erroneous, as it would neglect the fact that two systems trying to

$$\frac{s_1 \xhookrightarrow{a} \mu_1}{(s_1, s_2) \xhookrightarrow{a} \mu_1 \times \mathbb{1}_{s_2}} \qquad \frac{s_2 \xhookrightarrow{a} \mu_2}{(s_1, s_2) \xhookrightarrow{a} \mathbb{1}_{s_1} \times \mu_2}$$

$$\frac{s_1 \xhookrightarrow{a} \mu_1 \qquad s_2 \xhookrightarrow{b} \mu_2 \qquad \gamma(a,b) = c}{(s_1, s_2) \xhookrightarrow{c} \mu_1 \times \mu_2}$$

$$\frac{s_1 \overset{\lambda}{\rightsquigarrow} s_1' \qquad s_1 \neq s_1'}{(s_1, s_2) \overset{\lambda}{\rightsquigarrow} (s_1', s_2)} \qquad \frac{s_2 \overset{\lambda}{\rightsquigarrow} s_2' \qquad s_2 \neq s_2'}{(s_1, s_2) \overset{\lambda}{\rightsquigarrow} (s_1, s_2')} \qquad \frac{\lambda(s_1, s_2) > 0}{(s_1, s_2) \overset{\lambda(s_1, s_2)}{\rightsquigarrow} (s_1, s_2)}$$

Figure 3.5: Inference rules for the transitions of a parallel composition.

self-loop produce a faster self-loop than one system doing so. The current formalisation correctly produces the transition $(s_1, s_2) \overset{2\lambda}{\rightsquigarrow} (s_1, s_2)$.

Although our treatment of state labels in the parallel composition is rather standard [BK08, CGP01], it does need some attention if $\mathrm{AP}_1 \cap \mathrm{AP}_2 \neq \varnothing$. After all, for $p \in \mathrm{AP}_1 \cap \mathrm{AP}_2$ it may be the case that $p \in L_1(s_1)$ and $p \notin L_2(s_2)$, and hence by definition $p \in L((s_1, s_2))$ in the parallel composition. If we interpret $p \notin L_2(s_2)$ to mean $\neg p \in L_2(s_2)$, this basically would imply that p and $\neg p$ are combined to p. Hence, this only makes sense if the atomic propositions in the parallel composition can somehow be taken to be the *disjunction* of the atomic propositions of the constituent processes. In our setting, this is indeed the case, as will become clear in Section 4.2.3 when defining the state labelling of the MAs underlying specifications in our process algebra MAPA.

3.2.4 Probabilistic automata and interactive Markov chains

As discussed in Section 3.1.1, PAs are MAs without any Markovian transitions, and IMCs are MAs without any probabilistic choices. Hence, both models can easily be defined as special MAs and all our concepts and results for MAs are applicable to these types of models as well. This is important, since it has already been argued that both IMCs [Her02] and PAs [Seg95, Sto02b] are important models for working with stochastically timed and probabilistic systems.

Definition 3.23 (PAs). *A* PA *is an MA* $\mathcal{A} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L\rangle$ *with* $\rightsquigarrow = \varnothing$.

Sometimes, PAs are assumed not to have any state labels. Clearly, this can easily be achieved by taking $\mathrm{AP} = \varnothing$.

Definition 3.24 (IMCs). *An* IMC *is an MA* $\mathcal{A} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L\rangle$ *such that* $s \xhookrightarrow{a} \mu$ *implies that* $|supp(\mu)| = 1$.

For PAs, the set of extended transitions coincides with the set of interactive transitions, and thus they can be used interchangeably.

3.3 Isomorphism and bisimulation relations

It is often important to be able to say that two models are equivalent, for instance when reducing a model before analysing it. After all, we do not want to lose behaviour or change the properties of the model. Several behavioural equivalences have already been defined for MAs, most importantly notions of bisimulation. There is one important criterion that distinguishes them: some equivalences require internal transitions to be mimicked (*strong* equivalences), while others abstract from them (*weak* equivalences). Both types of bisimulation relations have their own merits [BK08].

Strong equivalences preserve more properties; most importantly, they preserve properties involving the length of a computation. Weak equivalences on the other hand allow more aggressive state space reductions, but preserve only properties that are oblivious to stuttering (repetition) of state labels and do not refer to internal actions or computation length. For most of our techniques we are able to prove a strong equivalence, showing that they can safely be applied while preserving a large set of properties. For confluence reduction, we explicitly aim at reducing the amount of internal behaviour; hence, a weak equivalence is needed.

3.3.1 Strong equivalences

We first discuss the two most important strong equivalences: *isomorphism* and *strong bisimulation.*

Isomorphism. Isomorphism is the strongest equivalence on MAs that we consider. For two models to be isomorphic, they have to be identical except for state names.

Definition 3.25 (Isomorphism). *Given an MA $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L\rangle$, two states $s, t \in S$ are* isomorphic *(denoted by $s \approx_{\mathrm{iso}} t$) if there exists a bijection $f \colon S \to S$ such that $f(s) = t$ and*

$$\forall s' \in S, \alpha \in A^{\chi}, \mu \in \mathrm{Distr}(S) \;.\; L(f(s')) = L(s') \wedge s' \xrightarrow{\alpha} \mu \iff f(s') \xrightarrow{\alpha} \mu_f$$

with μ_f as defined in Section 2.2.3. Two MAs $\mathcal{M}, \mathcal{M}'$ are isomorphic (denoted by $\mathcal{M} \approx_{\mathrm{iso}} \mathcal{M}'$) if their initial states are isomorphic in their disjoint union.

Strong bisimulation. The notion of strong bisimulation we introduce is taken from [EHZ10b], while adding support for state labels. Based on the traditional notions of strong bisimulation [Mil89, LS91, SL95], it is the most straightforward kind of bisimulation, equating any two states that cannot be distinguished. The system is considered to be quite transparant, in the sense that it can observe state labels, action labels and even internal transitions. Hence, bisimilar states should have the same state label. Moreover, any α-labelled transition enabled by one state should also be enabled by all bisimilar states, and all target states of these transitions should again be bisimilar. More precisely, the probability distributions of a transition and its mimicking transitions should assign the same probabilities to the equivalence classes of the state space under bisimulation (as formalised by the relation $\equiv_R$, defined in Section 2.2.3).

Definition 3.26 (Strong bisimulation). *Given an MA $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L\rangle$, an equivalence relation $R \subseteq S \times S$ is a* strong bisimulation *for $\mathcal{M}$ if for every $(s,t) \in R$ and all $\alpha \in A^\chi, \mu \in \mathrm{Distr}(S)$, it holds that $L(s) = L(t)$ and*

$$s \xrightarrow{\alpha} \mu \implies \exists \mu' \in \mathrm{Distr}(S) \,.\, t \xrightarrow{\alpha} \mu' \wedge \mu \equiv_R \mu'$$

Two states $s,t \in S$ are strongly bisimilar *(denoted by $s \approx_{\mathrm{s}} t$) if there exists a strong bisimulation R for $\mathcal{M}$ such that $(s,t) \in R$. Two MAs $\mathcal{M}, \mathcal{M}'$ are strongly bisimilar (denoted by $\mathcal{M} \approx_{\mathrm{s}} \mathcal{M}'$) if their initial states are strongly bisimilar in their disjoint union.*

It is easy to see that strong bisimulation is implied by isomorphism, by taking $R = \{(s,t) \in S \times S \mid t = f(s)\}$ as the bisimulation relation associated with an isomorphism f. We note that there are also notions of strong bisimulation allowing convex combinations of transitions [Seg95]; we do not need to consider these notions, as our procedures already preserve the current (stronger) variant of strong bisimulation. As future work, it might be beneficial to define more powerful reduction techniques that do take advantage of such a more liberal notion of bisimulation.

The following proposition (taken from [EHZ10b]) states that strong bisimulation is a congruence for parallel composition. It is of vital importance to the rest of this thesis, since we will often transform components while preserving strong bisimulation, prior to composing them in parallel.

Proposition 3.27. *Let $\mathcal{M}_1$, $\mathcal{M}_2$ and $\mathcal{M}_3$ be MAs such that $\mathcal{M}_1 \approx_{\mathrm{s}} \mathcal{M}_2$. Then,*

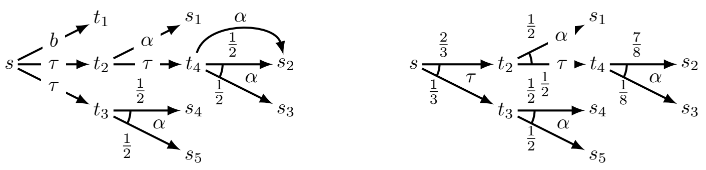
$$\mathcal{M}_1 \,||\, \mathcal{M}_3 \approx_{\mathrm{s}} \mathcal{M}_2 \,||\, \mathcal{M}_3$$

It is easy to see that the validity of this proposition is not influenced by the fact that our notion of strong bisimulation also requires state labels to coincide or by our slightly different manner of synchronisation.

3.3.2 Weak equivalences

Often, from an outside perspective not all behaviour of a system is observable. In action-based process algebras, the τ-action is used to model unobservable behaviour. In state-based model checking, *stuttering* transitions (i.e., transitions leaving the state labelling invariant) are assumed to be invisible. Since our definition of MAs contains both action and state labels, we assume transitions to be invisible if they are labelled by τ *and* their source state and all possible target states have the same state labelling.

As mentioned before, it is often desirable to abstract from internal behaviour and consider systems to be equivalent if the visible transitions coincide. This insight resulted in many weak equivalences for LTSs and PAs, among which weak bisimulation [Mil89, SL95, BH97, BS00, PLS00], and branching bisimulation [vGW96, SL95]. Weak bisimulation was also generalised to



(a) An MA $\mathcal{M}$. (b) Tree of $s \xRightarrow{\alpha} \mu$.

Figure 3.6: Weak transitions.

MAs [EHZ10b, EHZ10a, DH11, SZG12], but branching bisimulation not yet. In order to understandably introduce branching bisimulation for MAs, we first discuss weak transitions and then strengthen this concept to obtain branching transitions and branching bisimulation.

Weak transitions. In the non-probabilistic process-algebraic setting, abstraction from unobservable behaviour is formalised via the *weak transition*. A state s can do a weak transition to s' under an action a, denoted by $s \xRightarrow{a} s'$, if there exists a path $s \xrightarrow{\tau} s_1 \xrightarrow{\tau} \dots \xrightarrow{\tau} s_n \xrightarrow{a} s'$ with $n \geq 0$ (often, also τ-steps after the a-action are allowed, but this will not concern us). Traditionally, $s \xRightarrow{a} s'$ is thus satisfied by an *appropriate path*.

In our setting of Markov automata, $s \xRightarrow{\alpha} \mu$ is satisfied by an *appropriate scheduler*. A scheduler $\mathcal{S}$ is appropriate if its final state distribution $F^{\mathcal{S}}_{\mathcal{M}}(s)$ equals μ, and for every maximal path π that is scheduled from s with non-zero probability, it holds that

$$\mathit{trace}(\pi) = L(s)\,\tau\,L(s)\,\tau\,L(s)\ \dots\ L(s)\,\tau\,L(s)\,a\,X$$

where X is any state labelling. Hence, π always moves invisibly (with respect to both action labels and state labels) to a state from which an α-transition is possible, takes this transition and terminates.

Example 3.28. Consider the MA shown in Figure 3.6(a), and assume that $L(s) = L(t_2) = L(t_3) = L(t_4)$. We demonstrate that $s \xRightarrow{\alpha} \mu$, with

$$\mu(s_1) = \tfrac{8}{24} \qquad \mu(s_2) = \tfrac{7}{24} \qquad \mu(s_3) = \tfrac{1}{24} \qquad \mu(s_4) = \tfrac{4}{24} \qquad \mu(s_5) = \tfrac{4}{24}$$

To show the existence of this weak transition, we define a scheduler $\mathcal{S}$ such that

$$\begin{aligned}
\mathcal{S}(s) &= \{(s, \tau, \mathbb{1}_{t_2}) \mapsto 2/3, (s, \tau, \mathbb{1}_{t_3}) \mapsto 1/3\}\\
\mathcal{S}(t_2) &= \{(t_2, \alpha, \mathbb{1}_{s_1}) \mapsto 1/2, (t_2, \tau, \mathbb{1}_{t_4}) \mapsto 1/2\}\\
\mathcal{S}(t_3) &= \{(t_3, \alpha, \{s_4 \mapsto 1/2, s_5 \mapsto 1/2\}) \mapsto 1\}\\
\mathcal{S}(t_4) &= \{(t_4, \alpha, \mathbb{1}_{s_2}) \mapsto 3/4, (t_4, \alpha, \{s_2 \mapsto 1/2, s_3 \mapsto 1/2\}) \mapsto 1/4\}\\
\mathcal{S}(t_1) &= \mathcal{S}(s_1) = \mathcal{S}(s_2) = \mathcal{S}(s_3) = \mathcal{S}(s_4) = \mathcal{S}(s_5) = \mathbb{1}_{\perp}
\end{aligned}$$

Here we used $\mathcal{S}(s)$ to denote the choice made for every possible path ending in s.

The scheduler is depicted in Figure 3.6(b). Where it chooses probabilistically between two transitions with the same label, this is represented as a *combined transition*. For instance, from t_4 the transition $(t_4, \alpha, \{s_2 \mapsto 1\})$ is selected with probability 3/4, and $(t_4, \alpha, \{s_2 \mapsto 1/2, s_3 \mapsto 1/2\})$ with probability 1/4. This corresponds to the combined transition $(t_4, \alpha, \{s_2 \mapsto 7/8, s_3 \mapsto 1/8\})$.

Clearly, all maximal paths enabled from s move invisibly to a state from which they execute an α-transition and end directly afterwards.

The path probabilities can also be calculated. For instance,

$$\begin{aligned}
P^{\mathcal{S}}_{\mathcal{M},s}(s \xrightarrow{\tau,\{t_2\mapsto 1\}} t_2 \xrightarrow{\tau,\{t_4\mapsto 1\}} t_4 \xrightarrow{\alpha,\{s_2\mapsto 1\}} s_2) &= \tfrac{2}{3}\cdot 1\cdot \tfrac{1}{2}\cdot 1\cdot \tfrac{3}{4}\cdot 1 \\
&= \tfrac{6}{24} \\
P^{\mathcal{S}}_{\mathcal{M},s}(s \xrightarrow{\tau,\{t_2\mapsto 1\}} t_2 \xrightarrow{\tau,\{t_4\mapsto 1\}} t_4 \xrightarrow{\alpha,\{s_2\mapsto 1/2,s_3\mapsto 1/2\}} s_2) &= \tfrac{2}{3}\cdot 1\cdot \tfrac{1}{2}\cdot 1\cdot \tfrac{1}{4}\cdot \tfrac{1}{2} \\
&= \tfrac{1}{24}
\end{aligned}$$

As no other maximal paths from s go to s_2, and the scheduler always terminates directly after these paths, we find that $F^{\mathcal{S}}_{\mathcal{M}}(s)(s_2) = \frac{6}{24} + \frac{1}{24} = \frac{7}{24} = \mu(s_2)$.

Similarly, it can be shown that $F^{\mathcal{S}}_{\mathcal{M}}(s)(s_i) = \mu(s_i)$ for every $i \in \{1, 3, 4, 5\}$, so indeed $F^{\mathcal{S}}_{\mathcal{M}}(s) = \mu$. □

Branching bisimulation. The notion of branching bisimulation for non-probabilistic systems was first introduced in [vGW96]. Basically, it relates states that have an identical branching structure in the presence of τ-actions. Segala defined branching bisimulation for PAs [SL95], which we generalise here to MAs using the simplified notations of [Sto02a].

To introduce branching bisimulation, we need a restriction on weak transitions to obtain the *branching transition*. Intuitively, a state s can do a branching step $s \overset{\alpha}{\Longrightarrow}_R \mu$ if there exists a scheduler that terminates according to μ, always schedules precisely one α-transition (immediately before termination), does not schedule any other visible transitions and does not leave the equivalence class $[s]_R$ before doing an α-transition. Additionally, every state can do a branching τ-step to itself. Due to the use of extended transitions as a uniform manner of dealing with both interactive and Markovian transitions, this definition precisely coincides with the definition of branching steps for PAs, as defined in [TSvdP11].

Definition 3.29 (Branching transitions). *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L\rangle$ be an MA, $s \in S$, and R an equivalence relation over S. Then, $s \overset{\alpha}{\Longrightarrow}_R \mu$ if either (1) $\alpha = \tau$ and $\mu = \mathbb{1}_s$, or (2) there exists a scheduler $\mathcal{S}$ such that*

- *$F^{\mathcal{S}}_{\mathcal{M}}(s) = \mu$;*
- *For every maximal path*

$$s \xrightarrow{\alpha_1,\mu_1} s_1 \xrightarrow{\alpha_2,\mu_2} \dots \xrightarrow{\alpha_n,\mu_n} s_n \in \mathit{maxpaths}^{\mathcal{S}}_{\mathcal{M}}(s)$$

it holds that $\alpha_n = \alpha$. Moreover, for every $1 \le i < n$ we have $\alpha_i = \tau$, $(s, s_i) \in R$ and $L(s) = L(s_i)$.

Example 3.30. The tree in Figure 3.6(b) visualised a weak transition $s \overset{\alpha}{\Longrightarrow} \mu$. If for some equivalence relation R over the state space we find $t_2, t_3, t_4 \in [s]_R$, then it also demonstrates $s \overset{\alpha}{\Longrightarrow}_R \mu$. □

Based on branching transitions, we define branching bisimulation for MAs as a natural extension of the notion of naive weak bisimulation from [EHZ10b][3]. Naive weak bisimulation is an intuitive generalisation of weak bisimulation from PAs and IMCs to MAs. Naive weak bisimulation is implied by our notion of branching bisimulation, as it is obtained by omitting the requirement that $(s, s_i) \in R$ for all $1 \leq i < n$, and allowing convex combinations of transitions.

Definition 3.31 (Branching bisimulation). *Given an MA $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L \rangle$, an equivalence relation $R \subseteq S \times S$ is a* branching bisimulation *for $\mathcal{M}$ if for every $(s,t) \in R$ and all $\alpha \in A^\chi, \mu \in \mathrm{Distr}(S)$, it holds that $L(s) = L(t)$ and*

$$s \xrightarrow{\alpha} \mu \implies \exists \mu' \in \mathrm{Distr}(S) \,.\, t \overset{\alpha}{\Longrightarrow}_R \mu' \wedge \mu \equiv_R \mu'$$

Two states $s, t \in S$ are branching bisimilar *(denoted by $s \approx_{\mathrm{b}} t$) if there exists a branching bisimulation R for $\mathcal{M}$ such that $(s,t) \in R$. Two MAs $\mathcal{M}, \mathcal{M}'$ are branching bisimilar (denoted by $\mathcal{M} \approx_{\mathrm{b}} \mathcal{M}'$) if their initial states are branching bisimilar in their disjoint union.*

Note that, since each branching bisimulation relation R has the property that $(s,t) \in R$ implies $L(s) = L(t)$, the condition "$L(s) = L(s_i)$ for every $1 \leq i < n$" in Definition 3.29 is already implied by $(s, s_i) \in R$, and hence does not explicitly need to be checked for $t \overset{\alpha}{\Longrightarrow}_R \mu'$.

This notion of branching bisimulation has some appealing properties. First, the definition is robust in the sense that it can be adapted to using $s \overset{\alpha}{\Longrightarrow}_R \mu$ instead of $s \xrightarrow{\alpha} \mu$ in its condition. It turns out that this altered definition equates exactly the same systems. Second, the relation $\approx_{\mathrm{b}}$ induced by the definition is an equivalence relation.

Proposition 3.32. *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L \rangle$ be an MA. Then, an equivalence relation $R \subseteq S \times S$ is a branching bisimulation for $\mathcal{M}$ if and only if for every $(s,t) \in R$ and all $\alpha \in A^\chi, \mu \in \mathrm{Distr}(S)$, it holds that $L(s) = L(t)$ and*

$$s \overset{\alpha}{\Longrightarrow}_R \mu \implies \exists \mu' \in \mathrm{Distr}(S) \,.\, t \overset{\alpha}{\Longrightarrow}_R \mu' \wedge \mu \equiv_R \mu'$$

Proposition 3.33. *The relation $\approx_{\mathrm{b}}$ is an equivalence relation.*

Divergence sensitivity. If infinite paths of τ-actions can be scheduled with non-zero probability, then minimal probabilities (e.g., of eventually seeing an a-action) are not preserved by branching bisimulation. Consider for instance the two systems in Figure 3.7.

Note that, for the system on the left, the a-transition is not necessarily ever taken. After all, it is possible to indefinitely and invisibly loop through state

[3]Since our notion of branching bisimulation for MAs is just as naive as naive weak bisimulation for MAs, we could have called it *naive branching bisimulation*. However, since naive weak bisimulation for MAs is actually very related to weak bisimulation for PAs and IMCs, we argue that it would have made more sense to omit the 'naive' in the existing notion of naive weak bisimulation for MAs and prefix 'smart' to the existing notion of weak bisimulation for MAs. Our terminology still leaves room for a notion of smart branching bisimulation.

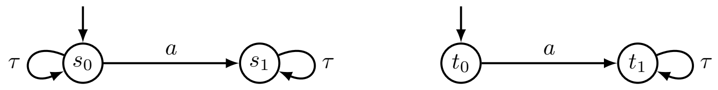


Figure 3.7: Two systems to illustrate divergence.

s_0 (*diverge*). For the system on the right, the a-transition cannot be avoided, assuming that termination cannot occur in states with outgoing transitions (as is always assumed by model checking tools). Still, these systems are branching bisimilar, as invisible behaviour does not necessarily have to be mimicked. Hence, branching bisimulation does not leave invariant all properties—in this case, the minimal probability of traversing an a-transition.

To solve this problem, *divergence-sensitive* notions of bisimulation have been introduced [BK08]. They force diverging states to be mapped to diverging states. This concept can be used in the same way for Markovian branching bisimulation. Since Markovian transitions already need to be mimicked, and the same holds for transitions that change the state labelling (since these cannot stay within the same equivalence class), divergence is defined as the traversal of an infinite path π that contains only τ-actions and never changes the state labelling (i.e., a path π such that $\mathit{invisible}(\pi)$). We formalise divergence by the existence of a scheduler that never terminates and only yields such paths.

Definition 3.34 (Divergence-sensitive relations). *An equivalence relation $R \subseteq S \times S$ over the states of an MA $\mathcal{M}$ is* divergence sensitive *if for all $(s, s') \in R$ it holds that*

$$\exists \mathcal{S} \,.\, \forall \pi \in \mathit{finpaths}_{\mathcal{M}}^{\mathcal{S}}(s) \,.\, \mathit{invisible}(\pi) \wedge \mathcal{S}(\pi)(\perp) = 0$$
$$\iff$$
$$\exists \mathcal{S}' \,.\, \forall \pi \in \mathit{finpaths}_{\mathcal{M}}^{\mathcal{S}'}(s') \,.\, \mathit{invisible}(\pi) \wedge \mathcal{S}'(\pi)(\perp) = 0$$

where $\mathcal{S}$ ranges over all possible schedulers for $\mathcal{M}$. Two MAs $\mathcal{M}_1, \mathcal{M}_2$ are divergence-sensitive branching bisimilar*, denoted by $\mathcal{M}_1 \approx_{\mathrm{b}}^{\mathrm{div}} \mathcal{M}_2$, if they are branching bisimilar and the equivalence relation to show this is divergence sensitive.*

Hence, if $(s, s') \in R$ and R is divergence sensitive, then s can diverge if and only if s' can. The proof of Proposition 3.33 can easily be extended to show that $\approx_{\mathrm{b}}^{\mathrm{div}}$ is an equivalence relation as well.

Weak bisimulation. Earlier work [EHZ10b] introduced an intricate notion of weak bisimulation for MAs, able to equate considerably more models than the naive weak bisimulation on which our notion of branching bisimulation is based. Our reduction techniques are not yet able to profit from the additional power of this notion of weak bisimulation—they already preserve the much finer notion of branching bisimulation presented above. Hence, we decided not to present the weak bisimulation from [EHZ10b] here. As mentioned in [EHZ10b],

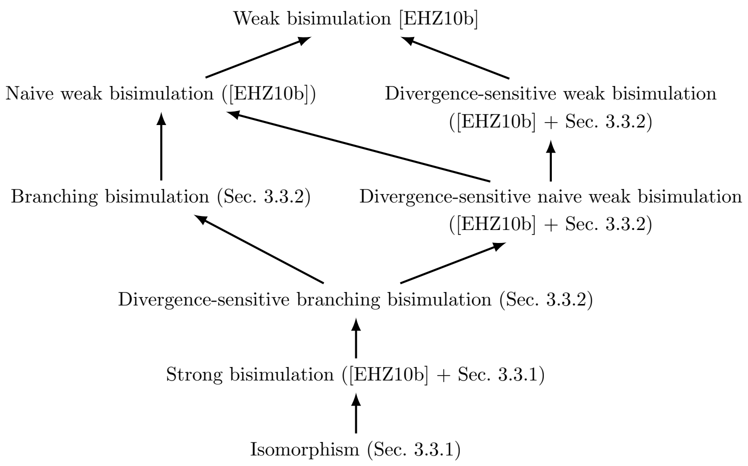

Weak bisimulation [EHZ10b]
Naive weak bisimulation ([EHZ10b])
Divergence-sensitive weak bisimulation ([EHZ10b] + Sec. 3.3.2)
Branching bisimulation (Sec. 3.3.2)
Divergence-sensitive naive weak bisimulation ([EHZ10b] + Sec. 3.3.2)
Divergence-sensitive branching bisimulation (Sec. 3.3.2)
Strong bisimulation ([EHZ10b] + Sec. 3.3.1)
Isomorphism (Sec. 3.3.1)

Figure 3.8: Hierarchy of equivalences for MAs.

though, systems equivalent according to naive weak bisimulation (and hence also systems equivalent according to our notion of branching bisimulation) are also equivalent according to weak bisimulation. Obviously, future work should focus on developing reduction techniques that do benefit from the additional power of this more involved concept.

Figure 3.8 summarises the equivalences that we introduced, together with the notions of weak bisimulation and naive weak bisimulation from [EHZ10b] and divergence-sensitive variants of them.

3.3.3 Property preservation by our notions of bisimulation

A notion of bisimulation is called *sound* with respect to a logic if bisimilar systems satisfy the same properties in that logic, and it is called *complete* if preservation of the same properties in the logic implies bisimulation. If a notion of bisimulation is sound as well as complete with respect to a logic, we say that it is *characterised* by that logic. For DTMCs and CTMCs such characterisations were given in [BKHW05]. For probabilistic automata, it took several more years to find a one-to-one correspondence [SZG11]. Often, we are already happy if a notion of bisimulation is sound with respect to a logic[4]; after all, this implies that we can safely replace systems by (hopefully smaller) bisimilar systems without influencing the validity of their properties.

Since MAs are still a rather recent formalism, not many logics have been defined for them so far. The only logic we are aware of is a variant of CSL (Continuous Stochastic Logic) introduced in [HH12], containing operators for

[4]Since isomorphic systems are identical except for state names, they clearly satisfy the same properties in any conceivable logic—as long as this logic cannot refer to the state names. Hence, isomorphism is sound with respect to every such logic.

unbounded and time-bounded reachability, but not dealing with expected times or long-run averages as allowed by other variants of CSL for different models [BHHZ11] (see Section 9.2.1 for an explanation of these types of properties). Additionally, there are no results yet on the relation between notions of bisimulation for MAs and the logic of [HH12]. Instead, the notions of bisimulation that have been defined were motivated by the fact that they coincide with their corresponding notions for PAs and IMCs in case an MA does not have any rates or probabilities, respectively [EHZ10b]. Hence, in these cases there does exist a logic that is preserved. We discuss the preservation of logical properties by strong bisimulation and branching bisimulation, for the subclass of PAs as well as for IMCs.

Strong bisimulation. In the degenerate case of an MA actually being an IMC, our notion of strong bisimulation coincides with the notion of strong bisimulation for IMCs [HK09]. It is well-known that strongly bisimilar IMCs satisfy exactly the same properties in the logic CSL, including time-bounded and unbounded reachability probabilities, expected reachability times and long-run averages [NK07].

When restricting to action-based probabilistic automata, our notion of strong bisimulation coincides with the notion of strong bisimulation from [SL95]. It was shown there that systems that are equivalent according to this notion satisfy exactly the same properties in an action-based variant of PCTL (Probabilistic Computation Tree Logic). Additionally, when restricting to a state-based setting, our notion is a strengthening of the strong bisimulation in [SZG11]. There, it was shown that strongly bisimilar systems satisfy the same properties in state-based PCTL* [BdA95].

Branching bisimulation. When restricting MAs to IMCs, we find that our notion of branching bisimulation almost coincides with the notion of weak bisimulation for IMCs [HK09]—our notion being slightly more restrictive with the branching requirement. It was shown in [HK09] that this notion preserves maximal time-bounded reachability properties (i.e., the probability of reaching a certain set of goal states within a certain time bound, under the most optimal scheduler maximising this probability, remains invariant). As always, minimal probabilities are not preserved due to divergences. This problem was solved in [Her02] by requiring bisimulation relations to be divergence sensitive, as later also used in [SZG12] for MAs to deal with the indefinite ignoring of behaviour. Hence, if we restrict our notion of branching bisimulation to be applied in a divergence-sensitive manner, then we also preserve minimal reachability properties of IMCs.

When restricting to action-based probabilistic automata, our branching bisimulation coincides with Segala's notion. He showed that branching bisimulation preserves all properties that can be expressed in the probabilistic temporal logic WPCTL (Weak Probabilistic Computation Tree Logic), provided that no infinite path of τ-actions can be scheduled with non-zero probability [SL95]. Again, it is well-known that this limitation to systems without divergences can be lifted by introducing divergence sensitivity, as for instance used in [BDG06]. When restricting to state-based probabilistic automata, our branching bisimulation coincides with the notion of probabilistic visible bisimulation [Grö08] (except that we only require invisibility of transitions, whereas [Grö08] requires invisibility of actions—i.e., they consider a transition invisible only if *all* transitions with the same action label are invisible as well). In [Grö08], it was stated that systems equivalent according to this notion preserve $\mathrm{PCTL}^*_{\setminus X}$ (PCTL* without the 'next' operator).

All these logics that are preserved by weak or branching bisimulation are very expressive, with the only limitation that a property cannot take into account the action or state labelling in the *next* state. After all, trace lengths are cut back by omitting irrelevant internal transitions. However, many interesting properties can still be specified without a next operator, such as time-bounded and unbounded reachability probabilities, expected reachability times and long-run averages.

We are mostly interested in time-bounded and unbounded reachability probabilities, expected times to reachability and long-run averages for MAs. Although not yet published, Sergey Sazonov from RWTH Aachen University showed that strong bisimulation and branching bisimulation indeed preserve expected times to reachability and long-run averages for MAs (in the absence of divergences). For unbounded reachability probabilities it follows easily from the fact that each MA can be transformed into a PA by changing rates to probabilities. Since unbounded reachability probabilities in an MA correspond to reachability probabilities in such an associated PA, and since branching bisimilar MAs yield branching bisimilar PAs, the fact that branching bisimulation preserves reachability probabilities in PAs implies that it preserves unbounded reachability probabilities in MAs. We conjecture that time-bounded reachability probabilities are preserved as well. For the subclass of IMCs it is a known result, and all our case studies confirm this conjecture also for MAs containing probabilities. It is additionally supported by the observation that our notion of branching bisimulation is a strengthening of all notions of weak bisimulation for MAs introduced thus far [EHZ10b, EHZ10a, DH11, SZG12].

3.4 Contributions

The main concepts concerning Markov automata, probabilistic automata and interactive Markov chains are well-established in the field. Relevant citations were already made throughout the section. Our presentation of these concepts was inspired by the way Stoelinga rephrased Segala's theory on probabilistic automata [Sto02a].

The definition of branching bisimulation for MAs, introduced in Section 3.3, and the corresponding Propositions 3.32 and 3.33, are original work, as well as the discussion in Section 3.3.3.

Part II

MAPA: Markov Automata Process Algebra

CHAPTER 4

Process Algebra for Markov Automata

"Whatever mathematical models are studied,
I believe that process calculi provide an
essential perspective for the study."

Robin Milner

While the models described in Chapter 3 are well-suited for formal verification, they tend to be very large for any non-trivial system. Hence, it is rather inconvenient and error-prone to model a specification directly as an MA or one of its subclasses. Instead, many specification languages have been introduced to model systems on a much higher level—just like programming languages. Among the most prominent specification techniques in model checking are *process algebras*: formal languages to describe behaviour in terms of actions, allowing compositional modelling using parallel composition.

Related work. No full-fledged process algebra for MAs existed thus far. There were already some modelling formalisms for probabilistic systems, but they often suffer from two major deficiencies:

Restricted treatment of data. The focus of probabilistic process algebras has mainly been on understanding random phenomena and the interplay between randomness and nondeterminism. Data is mostly treated in a restricted manner: probabilistic process algebras typically only allow a random choice over a fixed distribution, and input languages for probabilistic model checkers such as the reactive module language of PRISM [KNP11] or the probabilistic variant of Promela [BCG04] only support basic data types, but neither support more advanced data structures. To model realistic systems, however, convenient means for data modelling are indispensable. In the non-probabilistic world, μCRL [GP95] and LOTOS NT [GS98] are able to handle advanced data structures.

State space explosion. In addition to the problem of a restricted treatment of data, the *state space explosion* is always threatening the feasibility of most model-based analysis techniques (including model checking). After all, although parameterised probabilistic choice is semantically well-defined [BDHK06], the incorporation of data yields a significant increase of, or even an infinite, state space. However, current probabilistic minimisation techniques are not well-suited to be applied in the presence of data: aggressive abstraction techniques for probabilistic models (e.g., [DJJL01, dAR07, HMW09, KKLW07, KKNP10]) reduce at the model level, but the successful analysis of data requires *symbolic* reduction techniques. Such methods reduce stochastic models using syntactic transformations at the *language level*, minimising state spaces *prior to* their generation while preserving functional and quantitative properties (see for instance [DKP13]). Other approaches that try to deal with data in an efficient manner are two variants of probabilistic CEGAR (counterexample-guided abstraction refinement) [HWZ08, KKNP09], as well as the probabilistic guarded command language [MM99].

We refer back to Section 1.3 for a brief overview of the process algebras on which our work is founded.

Our approach. We alleviate both problems mentioned above, allowing systems with nondeterminism, probability and Markovian rates to be specified in an efficient data-dependent manner.

We introduce a generalisation of the process-algebraic language μCRL [GP95], named MAPA (Markov Automata Process Algebra), which treats data as a first-class citizen and adds support for both probability and Markovian rates. To the best of our knowledge, it is the first full-fledged process-algebraic language for specifying MAs. The language contains a carefully chosen minimal set of basic operators—on top of which syntactic sugar can be defined easily—and allows data-dependent probabilistic branching as well as Markovian delays. Additionally, operators for parallel composition, hiding, renaming and encapsulation allow a modular approach to system specification. Also, because of its process-algebraic nature, message passing can be used to define systems in a more modular manner than with for instance the PRISM language.

To battle the state space explosion, MAPA has been developed in such a way that reduction techniques can be defined rather easily. Our aim is to support symbolic minimisation techniques that operate at the syntax level and reduce our MAPA specifications in such a way that their underlying MAs become smaller while remaining equivalent. To enable such symbolic reductions, we introduce the MLPE: a generalisation of the *linear process equations* (LPEs) of μCRL [BG94b], which are a restricted form of process equations akin to the Greibach normal form in formal language theory, specifications in the language UNITY [CM88], and the precondition-effect style used for describing I/O automata [LT89]. Like the LPE for μCRL, these MLPEs allow for easy state space generation and parallel composition. Also, they simplify the definition of syntactic reduction techniques that either optimise state space generation or reduce the MA underlying a MAPA specification prior to its generation.

Hence, instead of immediately instantiating a MAPA specification to an MA, we always first transform the specification to an MLPE. Then, the corresponding state space can be generated, but more often we will exploit the MLPE's potential for reduction and first transform it to an equivalent MLPE that will generate a smaller (but still either strongly or branching bisimilar) MA. Figure 4.1 illustrates the general concepts of this approach.

$$\begin{tikzcd}
\text{MAPA} \arrow[r, "\text{transform}"] \arrow[d, dashed, "\text{instantiate}"'] & \text{MLPE}_1 \arrow[r, "\text{reduce}"] \arrow[d, "\text{instantiate}"'] & \text{MLPE}_2 \arrow[d, "\text{instantiate (optimised)}"] \\
\text{MA}_0 \arrow[r, dash, dashed, "\approx_s"] & \text{MA}_1 \arrow[r, dash, dashed, "\approx_s / \approx_b"] & \text{MA}_2
\end{tikzcd}$$

Figure 4.1: Efficient MA generation.

As some reduction techniques can be oblivious to the Markovian aspect of MAs, it is much more convenient to only define them on prCRL: the subclass of MAPA that does not contain Markovian rates. To support reduction techniques to be defined on this subclass, we show how to encode a MAPA specification into prCRL. Figure 4.2 illustrates this approach. We discuss under what circumstances transformations defined on prCRL can be used safely on encoded MAPA specifications, in such a way that the decoded transformed specification is still strongly bisimilar to the specification we started with. This requires a novel notion of bisimulation on prCRL specifications, based on preservation of the number of derivations for each transition. We immediately apply this paradigm by defining a two-phase *linearisation* procedure to transform prCRL specifications into LPPEs (MLPEs without any Markovian rates) and showing that it indeed can be applied to MAPA specifications as well. Similar linearisations have been provided for plain μCRL [BP95], as well as a real-time variant [Use02] and a hybrid variant [vdBRC06] thereof. Note that the procedure of encoding, linearising and decoding can be substituted for the *transform* arrow in Figure 4.1.

In this chapter, we also introduce three basic reduction techniques that optimise state space generation: constant elimination, expression simplification and summation elimination. The next two chapters are concerned with two state space reduction techniques based on the MLPE.

Organisation of the chapter. We first present the theoretical basics of process algebras in Section 4.1. Then, we introduce our process algebra MAPA in Section 4.2, presenting its syntax and semantics, defining the MLPE format and showing how it can be restricted to obtain the probabilistic process algebra prCRL. Also, this section demonstrates how to encode MAPA specifications in prCRL and apply transformations on these encodings in such a way that they are also valid for MAPA. Section 4.3 provides a procedure to linearise a prCRL specification to LPPE and proves that it can also be applied to MAPA specifications. Section 4.4 introduces an extension of MAPA with parallel com-

$$\begin{tikzcd}
\text{MAPA} \arrow[r, "\text{encode}"] \arrow[d, dash, dashed, "\approx_s"'] & \text{prCRL} \arrow[d, "\text{linearise}"] \\
\text{MLPE} \arrow[loop left, "\text{reduce}"] & \text{LPPE} \arrow[l, "\text{decode}"'] \arrow[loop right, "\text{reduce}"]
\end{tikzcd}$$

Figure 4.2: Linearising MAPA specifications using prCRL linerarisation.

position, encapsulation, hiding and renaming, and shows how systems specified using these constructs can be linearised as well. In Section 4.5 we exploit the MLPE format to define the three basic reduction techniques. Finally, Section 4.6 concludes by summarising the contributions of this chapter.

Origins of the chapter. The results in this chapter on prCRL were first published in the proceedings of the *10th International Conference on Application of Concurrency to System Design* (ACSD) [KvdPST10a] and a corresponding technical report [KvdPST10b]. Later, they were published more extensively in the journal *Theoretical Computer Science* [KvdPST12]. The language MAPA was introduced afterwards, in the proceedings of the *23rd International Conference on Concurrency Theory* (CONCUR) [TKvdPS12a] and a corresponding technical report [TKvdPS12b].

4.1 Process algebras

Before discussing our process algebra for MAs, we give a brief introduction to the field of process algebras. We based our concepts and notations on the treatment in [Fok07], and refer to that work for a more detailed discussion.

Each process algebra basically consists of two parts: a definition of its *syntax* (the *process terms* that are allowed) and a definition of its *semantics* (the meaning of its syntactic elements in terms of automata). Together, these two aspects combine to allow us to specify automata more efficiently by means of process-algebraic descriptions. Often, additionally *axioms* are introduced to more easily equate process terms that yield bisimilar automata. We are not concerned with such axioms, since we will justify our reduction techniques defined on process-algebraic descriptions by proving that the underlying automata of an original process term and its reduced variant are bisimilar.

4.1.1 Syntax: signatures and process terms

Process algebras are defined over a *signature* $\Sigma = \{f, g, \dots\}$, consisting of a finite set of *operators*. Each operator f is equipped with an *arity*, denoted by $ar(f)$, indicating its number of *arguments*. Operators with arity zero are often called *constants*, and operators with arity two are called *binary operators*.

In addition to the signature, a countable set of *variables* $V = \{x, y, z, \dots\}$, disjoint from Σ, is assumed. Based on a signature Σ and set of variables V, *process terms* can be constructed. Basically, they consist of operators applied to as many arguments as prescribed by their arity.

Definition 4.1 (Process terms). *Given a finite signature Σ and a countable set of variables V,* process terms over Σ and V *are constructed recursively as follows:*

- *Any variable $v \in V$ is a process term;*
- *If $f \in \Sigma$ and $p_1, p_2, \dots, p_n$ are process terms, with $n = ar(f)$, then also $f(p_1, p_2, \dots, p_n)$ is a process term.*

A process term is said to be open *if it contains at least one variable, and* closed *if it does not. Process terms containing binary operators, such as $f(p_1, p_2)$, are often written in* infix notation, *as in $p_1 \ f \ p_2$ or $(p_1 \ f \ p_2)$. Constants in process terms are often written without parentheses, i.e., as f instead of $f()$.*

Note that this is a rather simple view on process algebra. It provides a basic and thorough framework for process algebras without any data, binding operators or recursion; our MAPA language does contain such more complicated constructs. Still, the current treatment provides a solid basis for understanding various concepts such as SOS rules and proof trees.

Example 4.2. We define the syntax of a very simple process algebra, based on CCS [Mil80]. It consists of one constant $\emptyset$ for the deadlock process, a binary operator $+$ for alternative composition and two unary operators $a\cdot$ and $b\cdot$ for action prefix.

Formally, the signature of our process algebra is $\Sigma = \{\emptyset, +, a\cdot, b\cdot\}$, with $ar(\emptyset) = 0$, $ar(+) = 2$ and $ar(a\cdot) = ar(b\cdot) = 1$. We assume a set of variables $V = \{x, x', y, y'\}$. Often, we just write an action name a to abbreviate $a \cdot \emptyset$.

An infinite number of process terms can be constructed for this process algebra, such as:

- $a \cdot \emptyset$ (also written as a);
- $+(a \cdot \emptyset, b \cdot \emptyset)$ (also written as $a + b$);
- $+(b \cdot \emptyset, x)$ (also written as $b + x$);
- $+(a \cdot (b \cdot x), +(b, \emptyset))$ (also written as $(a \cdot b \cdot x) + (b + \emptyset)$).

The first two process terms are closed, whereas the last two are open. □

To transform an open process term into a closed one, *substitutions* can be applied. This entails renaming variables to (either closed or open) process terms.

Definition 4.3 (Substitutions). *Given a signature Σ and a countable set V of variables, a* substitution *is a list $\sigma = [x_1 := q_1, \dots, x_n := q_n]$, where $x_i \in V$ and q_i is a process term for each $1 \leq i \leq n$.*

Given a process term p and a substitution $\sigma = [x_1 := q_1, \dots, x_n := q_n]$, we define $\sigma(p)$ inductively on the structure of p:

$$\begin{aligned} \sigma(x) &= \begin{cases} q_i & \text{if } x = x_i \text{ for some } 1 \leq i \leq n \\ x & \text{otherwise} \end{cases} \\ \sigma(f(p_1, \dots, p_n)) &= f(\sigma(p_1), \dots, \sigma(p_n)) \end{aligned}$$

for any variable $x \in V$, process terms $p_1, \dots, p_n$ and operator $f \in \Sigma$. We lift this to sets of process terms P in the obvious way: $\sigma(P) = \{\sigma(p) \mid p \in P\}$.

Given two vectors $\boldsymbol{x} = (x_1, \dots, x_n)$ and $\boldsymbol{q} = (q_1, \dots, q_n)$, we write $[\boldsymbol{x} := \boldsymbol{q}]$ as an abbreviation for the substitution $\sigma = [x_1 := q_1, \dots, x_n := q_n]$.

4.1.2 Semantics

To give meaning to closed process terms, they are mapped to a semantic model. While our process algebra MAPA will map to the full spectrum of MAs, in this introductory section we restrict to semantics in terms of LTSs. Hence, there are no Markovian transitions, no state labels and all transitions have a Dirac distribution to determine their next state.

We use structural operational semantics [Plo81] to provide the semantics of process algebras, still assuming a signature Σ and a set V of variables. The basic building blocks of an SOS specification are transitions between process terms.

Definition 4.4 (Process term transitions). *Given a signature Σ and a set of variables V, a* process term transition *is an expression of the form $t \xrightarrow{a} t'$, such that*

- *$a \in \Sigma$ is an action;*
- *t, t' are (possibly open) process terms over Σ and V.*

A process term transition is called open *if it contains at least one variable, otherwise it is* closed*.*

Based on such process term transitions, the structural operational semantics of a process algebra are given by a set of *transition rules* (also called *SOS rules*).

Definition 4.5 (Transition rules). *Given a signature Σ and a set of variables V, a transition rule is a structure of the form*

$$\text{TR}\ \frac{p_1 \quad p_2 \quad \dots \quad p_n}{c}$$

where every p_i, as well as c, is a process term transition. For a transition rule as above, the process term transitions p_i are called its premises *and the process term transition c its* conclusion*. A transition rule is called* closed *if all its process term transitions are closed.*

Given a substitution σ, we write $\sigma(\text{TR})$ to denote the instantiated *transition rule*

$$\sigma(\text{TR})\ \frac{\sigma(p_1) \quad \sigma(p_2) \quad \dots \quad \sigma(p_n)}{\sigma(c)}$$

where a substitution σ applied to a process term transition $t \xrightarrow{a} t'$ is meant to be the process term transition $\sigma(t) \xrightarrow{a} \sigma(t')$.

Before formally defining how an LTS is obtained from a set of transition rules, we first present the transition rules for our basic process algebra and informally discuss their meaning.

Example 4.6. Figure 4.3 shows the transition rules for the simple process algebra introduced in Example 4.2 over $\Sigma = \{\emptyset, +, a\cdot, b\cdot\}$ and $V = \{x, x', y, y'\}$. We use a variable v that ranges over the actions a, b, to prevent having to include a copy of each rule. However, technically we just assume each rule to be present twice, once with $v = a$ and once with $v = b$. Due to the simplicity of our algebra, all transition rules have at most one premise.

$$\textsc{ActPrefix}\ \frac{}{v \cdot x \xrightarrow{v} x}$$

$$\textsc{NChoiceL}\ \frac{x \xrightarrow{v} x'}{x + y \xrightarrow{v} x'} \qquad \textsc{NChoiceR}\ \frac{y \xrightarrow{v} y'}{x + y \xrightarrow{v} y'}$$

Figure 4.3: SOS rules for our basic process algebra.

Each rule (after having chosen any action for v) can be *instantiated* by substituting closed process terms for (some of) x, x', y and y, in such a way that no closed process terms remain. For example, for $v = a$ and the substitution $\sigma = [x := a \cdot b, x' := b, y := b \cdot b \cdot a]$, we obtain

$$\sigma(\textsc{NChoiceL})\ \frac{a \cdot b \xrightarrow{a} b}{(a \cdot b) + (b \cdot b \cdot a) \xrightarrow{a} b}$$

This means that if we can prove the transition $a \cdot b \xrightarrow{a} b$, then $(a \cdot b) + (b \cdot b \cdot a) \xrightarrow{a} b$ immediately follows from this rule. More precisely: the transition given by a transition rule's conclusion is derivable if all transitions given by its premises are derivable.

The ActPrefix rule can be seen as our base case: it does not have any premises. It states that every action prefix yields a transition labelled by that action, and then transforms into the process that is prefixed by the action. The NChoiceL and NChoiceR rules describe that the behaviour of $x + y$ consists of the behaviours of x and y together: if x has a v-labelled transition to x', then so does $x + y$. □

Formally, a transition $p \xrightarrow{v} p'$ follows from a *proof* (also called *derivation*). This is a finite upwardly branching tree, i.e., a graph with a *root* node (drawn at the bottom) that is connected to a finite number of children (upwards neighbours), which in turn can have children, and so one. All nodes are labelled with process term transitions, and a node is depicted to be connected to its children by means of a horizontal line (as above in Example 4.6 and below in Example 4.8). The idea is that each node, together with its children, is an instantiation of a transition rule.

Definition 4.7 (Proofs). *Given a signature Σ, a set of variables V and a set of transition rules $R = \{\frac{P_i}{c_i}\}$ for Σ and V, a* proof *for a closed process term transition $p \xrightarrow{a} p'$ is a finite upwardly branching tree such that*

- *The root of the tree is $p \xrightarrow{a} p'$;*
- *For every node c, with set K of nodes directly above it, there is a transition rule $\frac{K'}{c'} \in R$ and a substitution σ such that $K = \sigma(K')$ and $c = \sigma(c')$.*

In the process algebra presented in the examples above, as well as the MAPA process algebra we define later on, all variables present in the premises of a transition rule are also present in its conclusion. Hence, since the root r of the tree is a closed process term transition, the restriction $c = \sigma(c')$ applied to $c = r$

implies that all process term transitions directly above the root are also closed. This argument can be repeated to see that a proof tree for our process algebras will never contain any variables.

Example 4.8. To show $(a \cdot b) + (b \cdot b \cdot a) \xrightarrow{a} b$, we provide the following finite upwardly branching tree. The rule names on the right-hand side are officially not part of the proof tree, but demonstrate how its correctness can be shown.

$$\dfrac{\dfrac{}{a \cdot b \xrightarrow{a} b}\ \textsc{ActPrefix}}{(a \cdot b) + (b \cdot b \cdot a) \xrightarrow{a} b}\ \textsc{NChoiceL}$$

Indeed, the root of this tree is the transition we want to prove. The tree has two nodes for which we have to show that they indeed arise for one of the transition rules:

- First, let $c = a \cdot b \xrightarrow{a} b$. It has no nodes directly above it, so $K = \varnothing$. Choosing $v = a$, we find that rule ActPrefix equals $\frac{K'}{c'}$ with $K' = \varnothing$ and $c' = a \cdot x \xrightarrow{a} x$. Taking $\sigma = [x := b]$, indeed $K = \sigma(K')$ and $c = \sigma(c')$.
- Second, let $c = (a{\cdot}b){+}(b{\cdot}b{\cdot}a) \xrightarrow{a} b$. It has one child; we find $K = \{a{\cdot}b \xrightarrow{a} b\}$. Choosing $v = a$, we find that rule NChoiceL equals $\frac{K'}{c'}$ with $K' = \{x \xrightarrow{a} x'\}$ and $c' = x + y \xrightarrow{a} x'$. Taking $\sigma = [x := a \cdot b, x' := b, y := b \cdot b \cdot a]$, indeed $K = \sigma(K')$ and $c = \sigma(c')$. □

4.1.3 Alternative syntax descriptions

Whereas the notations above—based on an explicit signature—fit a very basic process algebra, we often apply different techniques to easily specify more complicated process algebras. The most important difference is the use of a Backus-Naur form (BNF) definition of the syntax [BBG+63]. A BNF grammar is a set of rules

$$\begin{aligned} p_1 &::= P_1^1 \mid P_1^2 \mid \dots \mid P_1^{k_1} \\ &\dots \\ p_n &::= P_n^1 \mid P_n^2 \mid \dots \mid P_n^{k_n} \end{aligned}$$

where each expression P_j^i may contain operators (often called *terminals* in this context) and the symbols $p_1, \dots, p_n$ (called the *non-terminals*). The grammar has a starting point, for instance p_1, and generates a language consisting of all terms that can be obtained by rewriting p_1 into one of the expressions P_1^i and then recursively rewriting all non-terminals in this expression according to their rules. The set of process terms of the corresponding process algebra is then precisely this language.

BNF more easily allows us to define a process algebra in which operators also take parameters that are not process terms themselves.

Example 4.9. A process term in our basic process algebra of Example 4.2 is any term that can be generated by the following grammar:

$$p \ ::= \ \emptyset \ \mid \ p + p \ \mid \ v \cdot p$$

where $v \in \{a, b\}$. Note that this grammar yields an infinite language. □

4.2 Markov Automata Process Algebra

We introduce Markov Automata Process Algebra (MAPA), a language based on μCRL [GP95]. The process algebra μCRL allows the standard process-algebraic constructs, such as nondeterministic choice and action prefix, to be used in a data-rich context: processes are equipped with a set of variables over user-definable data types, and actions can be parameterised based on the values of these variables. Additionally, conditions can be used to restrict behaviour, and nondeterministic choices over data types are possible. This enables efficient modelling of many types of systems [BG94a, KS94, FGK97, GPW03, PS07].

MAPA adds two operators: a probabilistic choice over data types and a Markovian delay. The language therefore features all constructs available in MAs, and indeed we will show that MAPA specifications naturally map to MAs. Both the probabilistic behaviour and the delays may depend on data parameters, generalising the efficient way of modelling from μCRL to MAPA.

We assume an external mechanism for the evaluation of expressions (e.g., equational logic, or a fixed data language), able to handle at least boolean and real-valued expressions. The data language contains variables and allows concrete values to be substituted for these variables. Also, we assume that any expression that does not contain variables can be evaluated to a concrete value. Note that this restricts the expressiveness of the data language. In the examples we use an intuitive data language, containing basic arithmetic and boolean operators.

4.2.1 Syntax

We generally refer to data types with upper-case letters $D, E, \dots$ and to variables with lower-case letters $u, v, \dots$. Vectors, sets of vectors and Cartesian products are denoted in bold. We use $\{*\}$ to denote a singleton set with a dummy element. As in Section 3.2, we still assume a countable universe of actions *Act*. Additionally, we assume a set *Proc* of *process names*.

Definition 4.10 (Process terms). *A* process term *in MAPA is any term that can be generated by the following grammar:*

$$p ::= Y(\boldsymbol{t}) \ \mid \ c \Rightarrow p \ \mid \ p + p \ \mid \ \sum_{\boldsymbol{x}:\boldsymbol{D}} p \ \mid \ a(\boldsymbol{t}) \sum_{\boldsymbol{x}:\boldsymbol{D}}\!\!\!\bullet\ f : p \ \mid \ (\lambda) \cdot p$$

When omitting the $(\lambda) \cdot p$ construct, we obtain a process term in the language prCRL (discussed in Section 4.2.5).

Here, $Y \in \mathit{Proc}$ is a process name, $\boldsymbol{t}$ a vector of data expressions, c a boolean expression, $\boldsymbol{x}$ a vector of variables ranging over a countable[1] type $\boldsymbol{D}$, $a \in \mathit{Act}$ a (parameterised) atomic action, f a real-valued expression yielding values in $[0,1]$, and λ an expression yielding positive real numbers (rates). All expressions are allowed to contain variables—although later we will put some restrictions on them, requiring each data variable to either be a process variable or to be bound by a nondeterministic or probabilistic choice operator.

We write $p = p'$ only for syntactically identical process terms, not to relate process terms with identical meaning but different syntax.

Remark 4.11. Not all possible process terms have meaning. Since this depends on their context, we defer this issue to Section 4.2.2, where we specify several well-formedness criteria. □

Given an expression t, a process term p, a vector of variable names $\boldsymbol{x} = (x_1, \dots, x_n)$ and a vector of closed data expressions $\boldsymbol{d} = (d_1, \dots, d_n)$, we use $t[\boldsymbol{x} := \boldsymbol{d}]$ to denote the result of simultaneously substituting every x_i in t by d_i, and $p[\boldsymbol{x} := \boldsymbol{d}]$ for the result of applying this to every expression in p (only substituting each x_i by d_i in expressions in p that do not occur within a construct $\sum_{\boldsymbol{x}:\boldsymbol{D}}$ or $\sum\hspace{-0.75em}\bullet_{\boldsymbol{x}:\boldsymbol{D}}$ such that x_i is an element of $\boldsymbol{x}$ (not considering the context of p)).

Example 4.12. The grammar above provides the MAPA language with an infinite number of process terms. One of these is

$$\sum_{n:\mathbb{N}} n < 3 \Rightarrow (2 \cdot n + 1) \cdot \mathit{send}(n) \sum\hspace{-0.9em}\bullet_{x:\{1,2\}} \frac{x}{3} : (Y(n+x) + Z(n+x))$$

For the expression $t = \frac{x}{3}$ we find $t[x := 2] = \frac{2}{3}$, and for the process term $p' = Y(x) + Z(x)$ we find $p'[x := 2] = Y(2) + Z(2)$.

Note that two of our operators ($\sum$ and $\sum\hspace{-0.75em}\bullet$) range over *vectors* of data parameters. If a vector $\boldsymbol{x}$ is indeed nontrivial (i.e., it has more than one element), then its accompanying data type $\boldsymbol{D}$ is a Cartesian product. This is the case in the following process term:

$$\sum_{(m,i):\{m_1,m_2\}\times\{1,2,3\}} \mathrm{send}(m,i) \cdot Y()$$

□

To make it easier to understand the rest of this section, we already give a very informal idea of the meaning of the operators. In Section 4.2.3 we will go into more details, after having completed our discussion on the syntax.

In a process term, $Y(\boldsymbol{t})$ denotes *process instantiation*, where $\boldsymbol{t}$ instantiates Y's process variables as defined below. The term $c \Rightarrow p$ behaves as p if the *condition* c holds, and cannot do anything otherwise. The $+$ operator denotes *nondeterministic choice*, and $\sum_{\boldsymbol{x}:\boldsymbol{D}} p$ a (possibly infinite) *nondeterministic choice over data type* $\boldsymbol{D}$. The term $a(\boldsymbol{t}) \sum\hspace{-0.75em}\bullet_{\boldsymbol{x}:\boldsymbol{D}} f : p$ performs the action $a(\boldsymbol{t})$ and then

[1]Although uncountable choice may be well-defined process-algebraically, this would not work out for us as we want to provide semantics in terms of MAs.

has a *probabilistic choice* over $\boldsymbol{D}$. It uses the value $f[\boldsymbol{x} := \boldsymbol{d}]$ as the probability of choosing each $\boldsymbol{d} \in \boldsymbol{D}$. Finally, $(\lambda) \cdot p$ behaves as p after a delay, determined by a negative exponential distribution with rate λ.

Example 4.13. Consider again the first process term of Example 4.12. Intuitively, it behaves as follows:

1. The variable n nondeterministically gets assigned any natural number.

2. If $n < 3$, then the process continues with a delay, governed by an exponential distribution with rate $2 \cdot n + 1$.

3. The process does the action *send*, parameterised by the number n that was chosen earlier.

4. Probabilistically, x gets assigned a value from the set $\{1, 2\}$. Each value x has probability $\frac{x}{3}$ to be chosen, so 1 has probability $\frac{1}{3}$ and 2 has with probability $\frac{2}{3}$. Note that, as expected and will be required later on, these probabilities add up to 1.

5. Nondeterministically, the behaviour continues as either $Y(n{+}x)$ or $Z(n{+}x)$, with the value chosen nondeterministically in the first step substituted for n and the value chosen probabilistically in the previous step substituted for x.

Combining all these steps, and in anticipation of the formal semantics that are given later on, this yields the MA given in Figure 4.4, where each state t_i behaves as $Y(i) + Z(i)$. The behaviour of these processes was not specified yet. □

We do not consider sequential composition of process terms (i.e., a term of the form $p \cdot p$). Already in the non-probabilistic case sequential composition

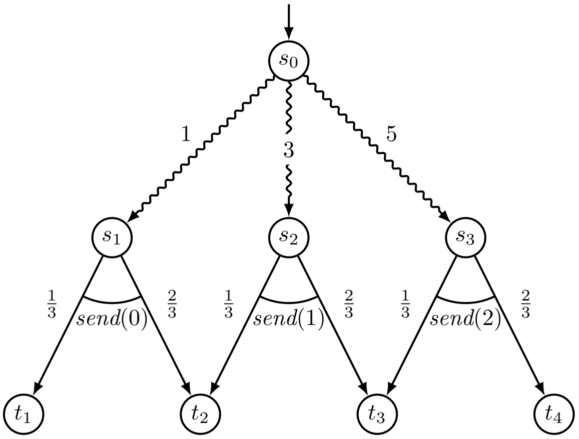


Figure 4.4: Semantics of the first process term of Example 4.12.

significantly increases the difficulty of linearisation (especially when using recursion) [Use02]. Therefore, it would distract from our main purpose: combining probabilities and Markovian rates with data. Moreover, many specifications can easily be written without sequential composition (as long as action prefix is present). Actually, [Use02] even showed that *all* sequential composition can be encoded by means of data.

We now formally introduce MAPA specifications. They consist of a set of process equations (defining behaviour for processes that can again be instantiated by other processes) and an initial process.

Definition 4.14 (Specifications). *A* MAPA specification *is given by a tuple* $M = (\{X_i(\boldsymbol{x_i} : \boldsymbol{D_i}) = p_i\}, X_j(\boldsymbol{t}))$ *consisting of a finite set of uniquely-named* processes X_i*, each defined by a* process equation $X_i(\boldsymbol{x_i} : \boldsymbol{D_i}) = p_i$*, and an* initial process $X_j(\boldsymbol{t})$ *with* $\boldsymbol{t}$ *a vector of closed data expressions. Here,* $\boldsymbol{x_i}$ *is a vector of* process variables[2] *with type* $\boldsymbol{D_i}$*, and* p_i *(the* right-hand side*) is a* process term *specifying the behaviour of* X_i*.*

A variable v *in an expression in a right-hand side* p_i *is* bound *if it is an element of* $\boldsymbol{x_i}$ *or it occurs within a construct* $\sum_{\boldsymbol{x}:\boldsymbol{D}}$ *or* $\textstyle\sum_{\boldsymbol{x}:\boldsymbol{D}}$ *such that* v *is an element of* $\boldsymbol{x}$*. Variables that are not bound are said to be* free.

We generally refer to process terms with lower-case letters p, q, r, and to processes with capitals X, Y, Z. Also, we write $X(x_1 : D_1, \dots, x_n : D_n)$ instead of $X((x_1, \dots, x_n) : (D_1 \times \cdots \times D_n))$. Finally, for brevity we often abuse notation by interpreting a single process equation as a specification (additionally mentioning the initial state if this is not clear from the context).

Example 4.15. The first process term introduced in Example 4.12 can be made into a proper MAPA specification by using it as a right-hand side of a process, adding behaviour for the processes Y and Z, and specifying an initial state. Hence, we obtain for instance the specification $M = (\mathit{Procs}, X)$, where *Procs* is the set of process equations

$$\begin{aligned} X &= \sum_{n:\mathbb{N}} n < 3 \Rightarrow (2 \cdot n + 1) \cdot \mathit{send}(n) \sum_{x:\{1,2\}} \frac{x}{3} : (Y(n + x) + Z(n + x)) \\ Y(y : \mathbb{N}) &= \mathit{beep}(y) \sum_{x:\{1\}} 1 : Y(y) \\ Z(z : \mathbb{N}) &= \mathit{ring}(z) \sum_{x:\{1\}} 1 : Z(z) \end{aligned}$$

□

Syntactic sugar. For notational ease, we define some syntactic sugar. First, given a process name X and an action a, we write X instead of $X()$ and a instead of $a()$. Second, given a process p and a (possibly parameterised) action $a(\boldsymbol{t})$, we write $a(\boldsymbol{t}) \cdot p$ as an abbreviation for the process term $a(\boldsymbol{t})\sum_{x:\{1\}} 1 : p$, where x is

[2]Note that we use the term *process variables* to denote data variables *within* a process (like a class variable is a variable within a class in object-oriented programming), not to denote variables referencing processes.

a fresh variable, not occurring freely in p. Third, for finite probabilistic sums we introduce the notation

$$a(\boldsymbol{t})(u_1 : p_1 \oplus u_2 : p_2 \oplus \cdots \oplus u_n : p_n)$$

for the process term that executes the action $a(\boldsymbol{t})$ and then behaves as process term p_i with probability u_i. This could be modelled in MAPA as the process term $a(\boldsymbol{t}) \sum_{x:\{1,\ldots,n\}} f : p$, where x is chosen such that it does not occur freely in any p_i, the probability expression f is such that $f[x := i] = u_i$ for every $1 \leq i \leq n$, and p is given by $(x = 1 \Rightarrow p_1) + (x = 2 \Rightarrow p_2) + \cdots + (x = n \Rightarrow p_n)$.

4.2.2 Static semantics

Not all syntactically correct MAPA specifications are meaningful. The following definition formulates additional well-formedness conditions. The first two constraints ensure that a specification does not refer to undefined variables or processes, the third is needed to obtain valid probability distributions, and the fourth ensures that the specification has a unique solution (modulo strong probabilistic bisimulation).

Additionally, all outgoing rates should be finite. This requirement is discussed in Remark 4.28, after providing the operational semantics and MLPE format—it is not easily defined statically, since it cannot be checked locally due to Markovian delays possibly being nested within several nondeterministic choices and conditions and hence depending on the operational semantics of these operators.

To define well-formedness, we require the concept of *unguardedness*. We say that a process term $Y(\boldsymbol{t})$ can go *unguarded* to Y. Moreover, $c \Rightarrow p$ can go unguarded to Y if p can, $p + q$ if either p or q can, and $\sum_{\boldsymbol{x}:\boldsymbol{D}} p$ if p can, whereas $a(\boldsymbol{t}) \sum_{\boldsymbol{x}:\boldsymbol{D}} f : p$ and $(\lambda) \cdot p$ cannot go unguarded anywhere.

Definition 4.16 (Well-formedness). *A MAPA specification*

$$M = (\{X_i(\boldsymbol{x_i} : \boldsymbol{D_i}) = p_i\}, X_j(\boldsymbol{t}))$$

is well-formed *if the following four constraints are all satisfied:*

1. *None of the right-hand sides p_i contains a free variable.*

2. *There are no instantiations to undefined processes. That is, for every instantiation $Y(\boldsymbol{t'})$ occurring in some right-hand side p_i, there exists a process equation $(X_k(\boldsymbol{x_k} : \boldsymbol{D_k}) = p_k)$ such that $X_k = Y$ and $\boldsymbol{t'}$ is of type $\boldsymbol{D_k}$. Also, the vector $\boldsymbol{t}$ used in the initial process is of type $\boldsymbol{D_j}$.*

3. *The probabilistic choices are well-defined. That is, for every construct $a(\boldsymbol{t}) \sum_{\boldsymbol{x}:\boldsymbol{D}} f : p$ occurring in a right-hand side p_i we have*

$$\sum_{\boldsymbol{d} \in \boldsymbol{D}} f[\boldsymbol{x} := \boldsymbol{d}] = 1$$

for every possible valuation of the free variables in $f[\boldsymbol{x} := \boldsymbol{d}]$ (the summation now used in the mathematical sense)[3].

4. *There is no unguarded recursion*[4]. *That is, for every process Y, there is no sequence of processes $X_1, X_2, \ldots, X_n$ (with $n \geq 2$) such that $Y = X_1 = X_n$ and p_j can go unguarded to X_{j+1} for every $1 \leq j < n$.*

We assume from now on that every MAPA specification is well-formed.

Note that the first two and the last requirement are easily checked statically. The third requirement is undecidable in general, but could of course simply be checked on-the-fly during state space generation.

Example 4.17. As an example of a well-formed MAPA specification, we consider a system that first writes the number 1, and then continuously writes natural numbers (excluding zero) in such a way that the probability of writing n is each time given by $\frac{1}{2^n}$. This system can be modelled by the MAPA specification $M = (\{X\}, X(1))$, where X is given by

$$X(n : \mathbb{N}^+) = \text{write}(n) \sum_{m:\mathbb{N}^+} \tfrac{1}{2^m} : X(m)$$

We demonstrate that all four well-formedness rules are satisfied:

1. The right-hand side contains two variables: n and m. The variable n is bound since it is a process variable of X. The variable m is bound since it only occurs within the construct $\sum_{m:\mathbb{N}^+}$.
2. There is one process instantiation: $X(m)$. Indeed, there is a process equation for this process, namely $X(n : \mathbb{N}^+) = \ldots$, and m is of type $\mathbb{N}^+$. Also, the initial parameter value 1 is of type $\mathbb{N}^+$, as required.
3. There is one probabilistic choice in the specification: $\sum_{m:\mathbb{N}^+} \frac{1}{2^m}$. Indeed,

$$\sum_{d \in \mathbb{N}^+} \tfrac{1}{2^m}[m := d] = \sum_{d \in \mathbb{N}^+} \tfrac{1}{2^d} = 1$$

   where the last step is well-known in mathematics since it concerns the geometric series.
4. There is one process: X. The only way for this process to yield unguarded recursion, would be to go unguarded to itself. However, X always starts with an action, and hence cannot go anywhere unguarded.

Note that the data types in MAPA specifications can be countably infinite. Also, probabilistic choices over countably infinite domains are allowed. Hence, countably infinite branching may occur, as long as no infinite exit rates are present. Since this is a semantic well-formedness rule, we will come back to it in Remark 4.28. □

[3]We could be slightly more liberal, only requiring this for all *reachable* valuations of the free variables in $f[\boldsymbol{x} := \boldsymbol{d}]$.

[4]This constraint could be relaxed a bit, as contradictory conditions of the processes may make an unguarded cycle harmless.

$$\text{INST}\ \frac{p[\boldsymbol{x} := \boldsymbol{d}] \xrightarrow{\alpha}_{\mathcal{D}} \beta}{Y(\boldsymbol{d}) \xrightarrow{\alpha}_{\text{INST}\,\mathcal{D}} \beta}\ \text{if } Y(\boldsymbol{x} : \boldsymbol{D}) = p \qquad \text{COND}\ \frac{p \xrightarrow{\alpha}_{\mathcal{D}} \beta}{\texttt{true} \Rightarrow p \xrightarrow{\alpha}_{\text{COND}\,\mathcal{D}} \beta}$$

$$\text{NCHOICEL}\ \frac{p \xrightarrow{\alpha}_{\mathcal{D}} \beta}{p + q \xrightarrow{\alpha}_{\text{NCHOICEL}\,\mathcal{D}} \beta} \qquad \text{NCHOICER}\ \frac{q \xrightarrow{\alpha}_{\mathcal{D}} \beta}{p + q \xrightarrow{\alpha}_{\text{NCHOICER}\,\mathcal{D}} \beta}$$

$$\text{NSUM}(\boldsymbol{d})\ \frac{p[\boldsymbol{x} := \boldsymbol{d}] \xrightarrow{\alpha}_{\mathcal{D}} \beta}{\sum_{\boldsymbol{x}:\boldsymbol{D}} p \xrightarrow{\alpha}_{\text{NSUM}(d)\,\mathcal{D}} \beta}\ \text{if } \boldsymbol{d} \in \boldsymbol{D} \qquad \text{MSTEP}\ \frac{-}{(\lambda) \cdot p \xrightarrow{\lambda}_{\langle \text{MSTEP} \rangle} p}$$

$$\text{PSUM}\ \frac{-}{a(\boldsymbol{t}) \displaystyle\sum_{\boldsymbol{x}:\boldsymbol{D}} f : p \xrightarrow{a(\boldsymbol{t})}_{\langle \text{PSUM} \rangle} \mu}\ \text{where } \mu(p[\boldsymbol{x} := \boldsymbol{d}]) = \sum_{\substack{\boldsymbol{d}' \in \boldsymbol{D} \\ p[\boldsymbol{x} := \boldsymbol{d}] = p[\boldsymbol{x} := \boldsymbol{d}']}} f[\boldsymbol{x} := \boldsymbol{d}'],\ \text{for every } \boldsymbol{d} \in \boldsymbol{D}$$

Figure 4.5: SOS rules for MAPA.

4.2.3 Operational semantics

We now formally define the operational semantics of well-formed MAPA specifications in terms of MAs. Most importantly, we explain how a MAPA specification is translated to a set of interactive transitions $p \overset{a}{\hookrightarrow} \mu$ and a set of Markovian transitions $p \overset{\lambda}{\rightsquigarrow} p'$. We define these sets in a two-phase manner: our SOS rules first provide *derivations* that can be either interactive or Markovian. Later, in Definition 4.22 we show how to obtain the sets $\hookrightarrow$ and $\rightsquigarrow$ from these derivations. The need to keep track of derivations is due to the well-known fact that the multiplicity of Markovian transitions is important when giving semantics to a Markovian process algebra [HHK02].

The SOS rules that define our semantics are provided in Figure 4.5, where p, q are process terms, $\lambda \in \mathbb{R}^{>0}$ is a rate, $a \in Act$ is an action, α is either a (possibly parameterised) action or a rate, $\boldsymbol{t}$ is a vector of data expressions, f is a real-valued expression, Y is a process name, $\boldsymbol{x}$ is a vector of variables, $\boldsymbol{D}$ is a vector of data types, β is either a process term or a probability distribution over process terms, and $\mathcal{D}$ is a sequence of SOS rule names (representing the derivation).

The intuition behind the rules is as follows:

INST. The behaviour of a process term $Y(\boldsymbol{d})$ depends on the right-hand side p of the process Y, substituting the actual parameters $\boldsymbol{d}$ of the instantiation for the process variables $\boldsymbol{x}$ of Y. Hence, if $p[\boldsymbol{x} := \boldsymbol{d}]$ can do a transition, then so can $Y(\boldsymbol{d})$.

COND. The behaviour of a process term $c \Rightarrow p$ is guarded by its condition c. If c holds, then $c \Rightarrow p$ behaves precisely as p; otherwise, it does not have any behaviour.

NCHOICEL / NCHOICER. The behaviour of a process term $p+q$ is the union of the behaviours of p and q. Hence, if either p or q can carry out a transition, then so can $p + q$.

NSUM($\boldsymbol{d}$). The behaviour of a process term $\sum_{\boldsymbol{x}:\boldsymbol{D}} p$ depends on p, which may contain variables from the vector $\boldsymbol{x}$. All behaviours that are possible by substituting the variables of $\boldsymbol{x}$ by any of their possible values from $\boldsymbol{D}$ are also enabled by $\sum_{\boldsymbol{x}:\boldsymbol{D}} p$. That is, every transition of $p[\boldsymbol{x} := \boldsymbol{d}]$, with $\boldsymbol{d} \in \boldsymbol{D}$, is possible from $\sum_{\boldsymbol{x}:\boldsymbol{D}} p$.

MStep. The behaviour of a process term $(\lambda) \cdot p$ does not depend on anything. It can just do a Markovian delay with rate λ and continue as p, and hence we can immediate derive $(\lambda) \cdot p \xrightarrow{\lambda}_{\langle \text{MStep} \rangle} p$.

PSum. The behaviour of a process term $a(\boldsymbol{t})\sum_{\boldsymbol{x}:\boldsymbol{D}} f : p$ can also be obtained immediately. It can execute its $a(\boldsymbol{t})$ action, and continue as p. However, p may contain free variables from the vector $\boldsymbol{x}$, which are instantiated according to the probability distribution implied by f. More concretely: the probability to instantiate $\boldsymbol{x}$ by $\boldsymbol{d} \in \boldsymbol{D}$ is given by $f[\boldsymbol{x} := \boldsymbol{d}]$. Hence, we would be tempted to say that the probability to continue as $p[\boldsymbol{x} := \boldsymbol{d}]$ is $f[\boldsymbol{x} := \boldsymbol{d}]$, but that is not correct. After all, there could well be several different valuations for $\boldsymbol{x}$ that all yield the same continuation. So, for any $\boldsymbol{d} \in \boldsymbol{D}$, the probability to continue as $p[\boldsymbol{x} := \boldsymbol{d}]$ (denoted by $\mu(p[\boldsymbol{x} := \boldsymbol{d}])$) is given by the *sum* of the probabilities $f[\boldsymbol{x} := \boldsymbol{d}']$ for all values $\boldsymbol{d}' \in \boldsymbol{D}$ such that $p[\boldsymbol{x} := \boldsymbol{d}] = p[\boldsymbol{x} := \boldsymbol{d}']$. (The need for summing probabilities of equal processes was already observed in [YL92].)

We note that the $+$ operator could not be omitted from the basic set of MAPA constructs, even though it may seem similar to the $\sum$ operator. After all, it can be observed from the operational semantics that a process term such as $a + b$ cannot be expressed by means of the $\sum$ operator.

The following proposition states the validity of the PSum rule.

Proposition 4.18. *The target μ of every transition derived using the SOS-rule* PSum *is a probability distribution over closed process terms.*

Compared to non-Markovian process algebras, our SOS rules are slightly more complicated: in addition to providing transitions from process terms to process terms, they also keep track of the proof tree that is employed to derive that transition. Since all of our SOS rules have at most one premise, each proof tree is actually just a linear sequence of nodes, which can be identified by the SOS rules that they instantiate. We call such a sequence of SOS rules a *derivation*, and write Δ for the set of all derivations. We construct the derivation as part of the transition, by adding it as a subscript to the transition relation—it is needed later to construct the Markovian transitions. Note that NSum is instantiated with a data element to keep track of the specific substitution that was performed to obtain a transition using this rule.

The SOS rules yield a transition $p \xrightarrow{\alpha}_{\mathcal{D}} \beta$ (note that this is not an extended transition but a new concept) if we can find a proof for it, precisely as in Definition 4.7. The target β can be either a probability distribution or a single state, depending on whether the derivation contains an MStep or a PSum rule.

Example 4.19. We show that the SOS rules yield the transition

$$(\lambda_1) \cdot q + \sum_{n:\{1,2,3\}} n < 3 \Rightarrow (\lambda_2) \cdot q \xrightarrow{\lambda_2}_{\langle \text{NChoiceR}, \text{NSum}(1), \text{Cond}, \text{MStep} \rangle} q$$

For this, consider the following proof tree:

$$\dfrac{\dfrac{\dfrac{\dfrac{-}{(\lambda_2) \cdot q \xrightarrow{\lambda_2}_{\langle \textsc{MStep} \rangle} q}\ \textsc{MStep}}{1 < 3 \Rightarrow (\lambda_2) \cdot q \xrightarrow{\lambda_2}_{\langle \textsc{Cond},\textsc{MStep} \rangle} q}\ \textsc{Cond}}{\sum_{n:\{1,2,3\}} n < 3 \Rightarrow (\lambda_2) \cdot q \xrightarrow{\lambda_2}_{\langle \textsc{NSum}(1),\textsc{Cond},\textsc{MStep} \rangle} q}\ \textsc{NSum}(1)}{(\lambda_1) \cdot q + \sum_{n:\{1,2,3\}} n < 3 \Rightarrow (\lambda_2) \cdot q \xrightarrow{\lambda_2}_{\langle \textsc{NChoiceR},\textsc{NSum}(1),\textsc{Cond},\textsc{MStep} \rangle} q}\ \textsc{NChoiceR}$$

Indeed, every node together with its upwards neighbour is an instantiation of an SOS rule (depicted on the right between each pair of nodes). Hence, this tree is a valid proof for the transition that is given as its root node.

Note that the name of each SOS rule that is applied is also mentioned as part of the transition. Hence, the derivation is present twice: on the right-hand side of the proof tree as well as subscripted to the transition. We could have omitted the names on the right of the tree, but chose to keep them to stay closest to the notation of Example 4.8. □

Traditionally, these transitions $p \xrightarrow{\alpha}_{\mathcal{D}} \beta$ generated by the SOS rules are the transitions of the underlying LTS. In our case, however, they are only an intermediate step. They are used to define the sets of interactive and Markovian transitions of the underlying MA, as explained in detail below.

Interactive transition relation. The interactive transition relation of the underlying MA of a MAPA specification is constructed trivially based on the transitions arising from the SOS rules: there is an interactive transition $p \overset{a}{\hookrightarrow} \mu$ for every transition $p \xrightarrow{\alpha}_{\mathcal{D}} \mu$ that can be proven using the SOS rules.

Markovian transition relation. The Markovian transition relation is also constructed based on the transitions arising from the SOS rules. However, now the number of different derivations for transitions between two process terms is taken into account. To see why this is needed, consider a process term like $(5) \cdot p + (5) \cdot p$. The SOS rules provide two different ways of demonstrating that this process term can go to p with rate 5. We can prove both

$$(5) \cdot p + (5) \cdot p \xrightarrow{5}_{\langle \textsc{NChoiceL},\textsc{MStep} \rangle} p \quad \text{and} \quad (5) \cdot p + (5) \cdot p \xrightarrow{5}_{\langle \textsc{NChoiceR},\textsc{MStep} \rangle} p$$

While in traditional process algebra this would just yield a single transition $(5) \cdot p + (5) \cdot p \xrightarrow{5} p$, that is not correct in our situation. Instead of having a rate of 5 to go to p, this process term must act with a rate of 10. After all, the minimum of two exponential distributions is distributed exponentially with the sum of the rates (as discussed in Section 2.2.4). This issue has been recognised before, leading to state-to-function transition systems [LMdV12], rate transition systems [DLLM09], multi-transition systems [Hil05], and derivation-labelled transitions [Pri95], to mention a few. Our approach is based on the latter.

First, we define $\mathsf{MD}(p, p')$ to be the set of Markovian derivations from p to p', given by pairs consisting of a rate and a derivation.

Definition 4.20 (Derivations). *Given two process terms p, p', we define*

$$\mathsf{MD}(p, p') = \{(\lambda, \mathcal{D}) \in \mathbb{R}^{>0} \times \Delta \mid p \xrightarrow{\lambda}_{\mathcal{D}} p'\}$$

to be the set of pairs $(\lambda, \mathcal{D})$ such that there is a transition $p \xrightarrow{\lambda}_{\mathcal{D}} p'$ that can be proven by our SOS rules.

Example 4.21. Consider again the process term

$$p = (\lambda_1) \cdot q + \sum_{n:\{1,2,3\}} n < 3 \Rightarrow (\lambda_2) \cdot q$$

In Example 4.19 we already showed that $p \xrightarrow{\lambda_2}_{\mathcal{D}} q$, with

$$\mathcal{D} = \langle \text{NChoiceR}, \text{NSum}(1), \text{Cond}, \text{MStep} \rangle$$

In the same way, we can find one other derivation $\mathcal{D}'$ with rate λ_2 using NSum(2), and finally $p \xrightarrow{\lambda_1}_{\mathcal{D}''} q$ with $\mathcal{D}'' = \langle \text{NChoiceL}, \text{MStep} \rangle$. Since these are the only Markovian derivations from p to q, we find

$$\mathsf{MD}(p, q) = \{(\lambda_2, \mathcal{D}), (\lambda_2, \mathcal{D}'), (\lambda_1, \mathcal{D}'')\} \qquad \square$$

Now, the total rate from a process term p to a process term q can easily be obtained by summing all rates in $\mathsf{MD}(p, q)$.

Based on the observations above, we now formally define the operational semantics of a well-formed MAPA specification.

Definition 4.22 (Operational semantics). *The semantics of a MAPA specification*

$$M = (\{X_i(\boldsymbol{x_i} : \boldsymbol{D_i}) = p_i\}, X_j(\boldsymbol{t}))$$

is an MA $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \text{AP}, L \rangle$, where

- $s^0 = X_j(\boldsymbol{t})$;
- $\hookrightarrow$ *is the smallest relation such that $p \xhookrightarrow{\alpha} \mu$ if there exists a $\mathcal{D} \in \Delta$ for which $p \xrightarrow{\alpha}_{\mathcal{D}} \mu$ holds;*
- $\rightsquigarrow$ *is the smallest relation such that $p \overset{\lambda}{\rightsquigarrow} p'$ if $\mathsf{MD}(p, p') \neq \varnothing$ and*

$$\lambda = \sum_{(\lambda', \mathcal{D}) \in \mathsf{MD}(p, p')} \lambda'$$

- $L(s) = \{a \in A \setminus \{\tau\} \mid \exists s' \in S \,.\, s \xhookrightarrow{a} s'\}$,

and S, A and AP are the smallest sets such that s^0, $\hookrightarrow$ and $\rightsquigarrow$ are well-defined.

Note that we cannot give S explicitly based on a specification, as its reachable states cannot be determined statically. Therefore, also the set of actions A and hence the set of atomic propositions AP can only be constructed by first computing the reachable state space.

Also note that we may obtain infinite rates, if the MAPA specification is not constructed carefully. We discuss in Remark 4.28 how to check for this.

Remark 4.23. We chose the state labelling in such a way that each state is labelled by the set of actions that it enables. This may seem restrictive, since during model checking we may also be interested in the values of some of the global variables. However, we will see in Section 4.2.4 that any condition over the global variables can easily be encoded by the enabledness of an action (see Remark 4.29).

Note that this implies that none of the proofs showing strong bisimulation for MAPA need to be concerned with the state labelling. After all, since $(s, t) \in R$ implies that all transitions from s can be mimicked by t and vice versa, clearly s and t enable the same actions and hence $L(s) = L(t)$. □

Example 4.24. Continuing with Example 4.21, we find that there is a transition from p to q with rate $\lambda = \lambda_1 + 2\lambda_2$. □

Equivalent MAPA specifications. Given a MAPA specification M and its underlying MA $\mathcal{M}$, two process terms in M are isomorphic if their corresponding states in $\mathcal{M}$ are isomorphic. Two specifications with underlying MAs $\mathcal{M}_1, \mathcal{M}_2$ are isomorphic if $\mathcal{M}_1$ is isomorphic to $\mathcal{M}_2$. Bisimilar process terms and specifications are defined in the same way.

Associativity of nondeterministic sums. It can easily be seen from our operational semantics that nondeterministic sums are associative, i.e., $(p + q) + r$ and $p + (q + r)$ yield the same transitions. The derivations differ, having for instance $\langle \text{NChoiceR}, \text{NChoiceL} \rangle$ instead of $\langle \text{NChoiceL}, \text{NChoiceR} \rangle$ for choosing q, but the resulting MAs will not be any different as the *number* of derivations does not change. Hence, instead of writing $\langle \text{NChoiceL}, \text{NChoiceR} \rangle$ for choosing q in $(p + q) + r$, we could also omit the parentheses and write $\langle \text{NChoice}(2) \rangle$ for choosing q in $p+q+r$. So, we will often just write $p+q+r$ without parentheses.

Congruences. Although not specifically needed for the results in this work, it can be shown that strong bisimulation is a congruence for all MAPA operators, along the same lines as the proof of Proposition 4.35. Just like weak bisimulation is not a congruence for IML [Her02], branching bisimulation is *not* a congruence for MAPA. After all, while $a \cdot p \approx_{\text{b}} \tau \cdot a \cdot p$, we have $a \cdot p + \lambda \cdot p \not\approx_{\text{b}} \tau \cdot a \cdot p + \lambda \cdot p$. We leave it for future work to consider weak congruences for MAPA (as already suggested in [EHZ10b]).

4.2.4 Markovian Linear Process Equations

In the non-probabilistic setting, a restricted version of μCRL is captured by the LPE format [BG94b]. It is well-suited for formal manipulation, state space generation and parallel composition. In the purely functional setting of LTSs, LPEs provided a uniform and simple format for the data-rich process algebra μCRL. As a consequence of this simplicity, the LPE format was essential for theory development and tool construction. It led to elegant proof methods, like the use of invariants for process algebra [BG94b], and the cones and foci method for proof checking process equivalence [GS01, FPvdP06]. It also enabled the application of model checking techniques to process algebra, such as optimisations from static analysis [GL01] (including dead variable reduction [vdPT09a]), data abstraction [VvdP07], distributed model checking [BLvdPW11], symbolic model checking (either with BDDs [BvdP08] or by constructing the product of an LPE and a parameterised μ-calculus formula [GM98, GW05]), and confluence reduction [BvdP02] (a variant of partial order reduction). In all these cases, the LPE format enabled a smooth theoretical development with rigorous correctness proofs (often checked in PVS), and a unifying tool implementation. It also allowed the cross-fertilisation of the various techniques by composing them as LPE to LPE transformations.

A μCRL specification is in LPE format if it consists of only one process and has the following structure:

$$\begin{aligned} X(\boldsymbol{g} : \boldsymbol{G}) = &\textstyle\sum_{\boldsymbol{d_1}:\boldsymbol{D_1}} c_1 \Rightarrow a_1(\boldsymbol{b_1}) \cdot X(\boldsymbol{n_1}) \\ &+ \textstyle\sum_{\boldsymbol{d_2}:\boldsymbol{D_2}} c_2 \Rightarrow a_2(\boldsymbol{b_2}) \cdot X(\boldsymbol{n_2}) \\ &\dots \\ &+ \textstyle\sum_{\boldsymbol{d_k}:\boldsymbol{D_k}} c_k \Rightarrow a_k(\boldsymbol{b_k}) \cdot X(\boldsymbol{n_k}) \end{aligned}$$

Each of the k components is called a *summand*. The conditions c_i, action parameters $\boldsymbol{b_i}$ and next-state vectors $\boldsymbol{n_i}$ may all depend on the process variables $\boldsymbol{g}$ and on $\boldsymbol{d_i}$. The LPE corresponds to the well-known precondition-effect style [LT89].

As each summand contains only one action and then immediately has a recursive call, the values of the process variables $\boldsymbol{g}$ completely characterise the state of an LPE. Hence, a valuation of $\boldsymbol{g}$ is often called a *state vector* (possibly empty), and the process variables are also called the LPE's *global variables*. The variables $\boldsymbol{d_i}$ are often called a summand's *local variables*; these may be absent.

Each summand potentially gives rise to several transitions: it may be enabled for many different valuations of the global variables $\boldsymbol{g}$, and hence produce transitions in multiple states. Additionally, even per state a summand may yield more than one transition, due to its local nondeterministic choice.

Sometimes, the LPE format is more compactly represented as

$$X(\boldsymbol{g} : \boldsymbol{G}) = \sum_{i \in I} \sum_{\boldsymbol{d_i}:\boldsymbol{D_i}} c_i \Rightarrow a_i(\boldsymbol{b_i}) \cdot X(\boldsymbol{n_i})$$

Example 4.25. Consider a system consisting of two buffers, B_1 and B_2. Continuously, buffer B_1 reads a message of type D from the environment, and sends it synchronously to buffer B_2. Then, B_2 writes the message back to the environment. The following LPE has exactly this behaviour when initialised with $a = 1$ and $b = 1$ (x and y can be chosen arbitrarily):

$$\begin{aligned} &X(a : \{1,2\}, b : \{1,2\}, x : D, y : D) = \\ &\quad \textstyle\sum_{d:D} \quad a = 1 \qquad\qquad\quad \Rightarrow \mathrm{read}(d) \cdot X(2, b, d, y) \qquad (1) \\ &\quad + \qquad\quad a = 2 \land b = 1 \Rightarrow \mathrm{comm}(x) \cdot X(1, 2, x, x) \quad (2) \end{aligned}$$

$$+ \qquad b = 2 \qquad \Rightarrow \text{write}(y) \cdot X(a, 1, x, y) \qquad (3)$$

The first summand models B_1's reading, the second the inter-buffer communication, and the third B_2's writing. The global variables a and b are used as program counters for B_1 and B_2, and x and y for their local memory. □

We generalise the LPE format to a restricted format for our MAPA language. As it should easily be mapped onto MAs, it should follow the concept of nondeterministically choosing between rates and actions. Hence, we need Markovian and interactive summands, mirroring the fact that MAs have Markovian and interactive transitions. The rate transitions should have unique target states, whereas the state after an interactive transition should be determined probabilistically. Therefore, a natural generalisation of the LPE to the Markovian world is the format given by the following definition.

Definition 4.26 (MLPEs). *A* Markovian linear process equation *(MLPE) is a MAPA specification of the following format:*

$$\begin{aligned} X(\boldsymbol{g} : \boldsymbol{G}) = & \sum_{i \in I} \sum_{\boldsymbol{d_i} : \boldsymbol{D_i}} c_i \Rightarrow a_i(\boldsymbol{b_i}) \sum_{\boldsymbol{e_i} : \boldsymbol{E_i}}\!\!\!\!\!\bullet\;\; f_i : X(\boldsymbol{n_i}) \\ & + \sum_{j \in J} \sum_{\boldsymbol{d_j} : \boldsymbol{D_j}} c_j \Rightarrow (\lambda_j) \cdot X(\boldsymbol{n_j}) \end{aligned}$$

The first $|I|$ nondeterministic choices are referred to as interactive summands, *the last $|J|$ as* Markovian summands.

The expressions c_i, $\boldsymbol{b_i}$, f_i and $\boldsymbol{n_i}$ may depend on $\boldsymbol{g}$ and $\boldsymbol{d_i}$, and f_i and $\boldsymbol{n_i}$ also on $\boldsymbol{e_i}$. Similarly, c_j, λ_j and $\boldsymbol{n_j}$ may depend on $\boldsymbol{g}$ and $\boldsymbol{d_j}$. Often, we write $c_i(\boldsymbol{g}', \boldsymbol{d}'_{\boldsymbol{i}})$ for $c_i[(\boldsymbol{g}, \boldsymbol{d_i}) := (\boldsymbol{g}', \boldsymbol{d}'_{\boldsymbol{i}})]$, and we use similar shorthands for the other expressions.

As an MLPE consists of only one process, an initial process $X(\boldsymbol{v})$ can be represented by its *initial vector* $\boldsymbol{v}$. Often, we will use the same name for the specification of an MLPE and the single process it contains. Also, we sometimes use $X(\boldsymbol{v})$ to refer to the specification $X = (\{X(\boldsymbol{g} : \boldsymbol{G}) = \dots\}, X(\boldsymbol{v}))$.

Operational semantics. Since a state vector $\boldsymbol{g}' \in \boldsymbol{G}$ completely characterise the state of an MLPE, we can simplify the underlying MA by identifying each state by $\boldsymbol{g}'$ instead of $X(\boldsymbol{g}')$. The initial state is the initial vector.

Since an MLPE is a MAPA specification, its semantics are given by the SOS rules in Table 4.5. However, due to the strict structure of an MLPE, we can simplify to explicit non-recursive semantics. It immediately follows from the SOS rules that for all $\boldsymbol{g}' \in \boldsymbol{G}$, there is a transition $\boldsymbol{g}' \xhookrightarrow{a(\boldsymbol{q})} \mu$ if and only if for at least one summand $i \in I$ there is a local choice $\boldsymbol{d}'_{\boldsymbol{i}} \in \boldsymbol{D_i}$ such that[5]

$$c_i \wedge a_i(\boldsymbol{b_i}) = a(\boldsymbol{q}) \wedge \forall \boldsymbol{e}'_{\boldsymbol{i}} \in \boldsymbol{E_i} \,.\, \mu(\boldsymbol{n_i}[\boldsymbol{e_i} := \boldsymbol{e}'_{\boldsymbol{i}}]) = \sum_{\substack{\boldsymbol{e}''_{\boldsymbol{i}} \in \boldsymbol{E_i} \\ \boldsymbol{n_i}[\boldsymbol{e_i} := \boldsymbol{e}'_{\boldsymbol{i}}] = \boldsymbol{n_i}[\boldsymbol{e_i} := \boldsymbol{e}''_{\boldsymbol{i}}]}} f_i[\boldsymbol{e_i} := \boldsymbol{e}''_{\boldsymbol{i}}]$$

[5]Note that for the variables $\boldsymbol{g}$, $\boldsymbol{d_i}$ and $\boldsymbol{e_i}$ we use the primed notations $\boldsymbol{g}'$, $\boldsymbol{d}'_{\boldsymbol{i}}$, $\boldsymbol{e}'_{\boldsymbol{i}}$ and $\boldsymbol{e}''_{\boldsymbol{i}}$ to denote specific values for these variables. We will often use this convention of having unprimed letters denote variables and primed letters denote possible valuations for them.

where, for readability, the substitution $[(\boldsymbol{g}, \boldsymbol{d_i}) := (\boldsymbol{g'}, \boldsymbol{d'_i})]$ is omitted from c_i, $\boldsymbol{b_i}$, $\boldsymbol{n_i}$ and f_i. Additionally, there is a transition $\boldsymbol{g'} \overset{\lambda}{\rightsquigarrow} \boldsymbol{g''}$ if and only if $\lambda > 0$ and

$$\lambda = \sum_{\substack{(j, \boldsymbol{d'_j}) \in J \times \boldsymbol{D_j} \\ c_j \wedge \boldsymbol{n_j} = \boldsymbol{g''}}} \lambda_j$$

omitting the substitution $[(\boldsymbol{g}, \boldsymbol{d_j}) := (\boldsymbol{g'}, \boldsymbol{d'_j})]$ from c_j, $\boldsymbol{n_j}$ and λ_j.

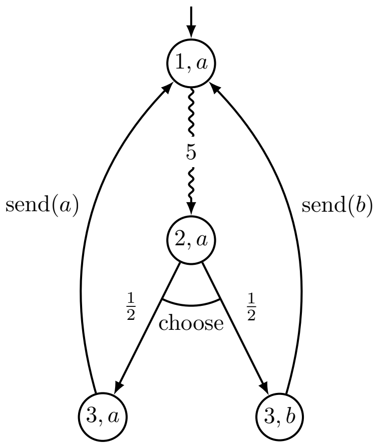
Example 4.27. Consider the following system, continuously sending a random element of a finite type D after waiting for an exponentially distributed time:

$$X = (5) \cdot \text{choose} \sum_{x:D} \frac{1}{|D|} : \text{send}(x) \cdot X$$

Now consider the following MLPE, where $d' \in D$ was chosen arbitrarily. It is easy to see that X is isomorphic to $Y(1, d')$. Note that d' could be chosen arbitrarily, since it is overwritten before used.

$$\begin{aligned} Y(pc : \{1,2,3\}, x : D) &= pc = 1 \Rightarrow (5) \cdot Y(2, x) \\ &+ pc = 2 \Rightarrow \text{choose}\textstyle\sum_{d:D} \frac{1}{|D|} : Y(3, d) \\ &+ pc = 3 \Rightarrow \text{send}(x) \textstyle\sum_{y:\{1\}} 1 : Y(1, d') \end{aligned} \qquad \square$$

The first summand is Markovian, whereas the other two are interactive. Taking $D = \{a, b\}$ and $d' = a$, and applying the semantics defined above, we obtain the following MA:



Obviously, the earlier defined syntactic sugar also applies to MLPEs; we can write $\text{send}(x) \cdot Y(1, d')$ in the second summand. However, as we define linearisation only on the basic operators, we will often keep writing the full form.

Remark 4.28. We already noted in Section 4.2.2 that MAPA specifications are not allowed to have infinite outgoing rates (since this is not allowed by MAs), although at that point we could not yet make this precise. With the MLPE format, though, it is much easier to define and heuristically check this.

For the semantics to be an MA with finite outgoing rates, we can now require that for every Markovian summand $j \in J$ and every reachable state $\boldsymbol{g'} \in \boldsymbol{G}$,

$$\sum_{\substack{\boldsymbol{d'_j} \in \boldsymbol{D_j} \\ c_j(\boldsymbol{g'}, \boldsymbol{d'_j})}} \lambda_j(\boldsymbol{g'}, \boldsymbol{d'_j}) < \infty$$

That is, for each reachable state $\boldsymbol{g'} \in \boldsymbol{G}$ and each Markovian summand j, the Markovian transitions that may be generated in $\boldsymbol{g'} \in \boldsymbol{G}$ by j should add up to a finite number.

One way of enforcing this syntactically is to require all data types in Markovian summands to be finite. For the MLPE in Example 4.27, this is indeed the case. Note, however, that this restriction would disallow an MLPE having a Markovian summand such as

$$\sum_{n:\mathbb{N}} (\tfrac{1}{2^n}) \cdot X(n)$$

even though it is well-formed. □

Remark 4.29. For any MAPA specification in the MLPE format, it is trivial to encode a condition c over its global variables $\boldsymbol{g}$ by the enabledness of an action, as mentioned in Remark 4.23. We just add a single summand to the MLPE, having c as its enabling condition, obtaining

$$\begin{aligned} X(\boldsymbol{g} : \boldsymbol{G}) = {} & \sum_{i \in I} \sum_{\boldsymbol{d_i}:\boldsymbol{D_i}} c_i \Rightarrow a_i(\boldsymbol{b_i}) \sum\nolimits^{\bullet}_{\boldsymbol{e_i}:\boldsymbol{E_i}} f_i : X(\boldsymbol{n_i}) \\ + {} & \sum_{j \in J} \sum_{\boldsymbol{d_j}:\boldsymbol{D_j}} c_j \Rightarrow (\lambda_j) \cdot X(\boldsymbol{n_j}) \\ + {} & \qquad\qquad c \Rightarrow \text{conditionReached} \sum\nolimits^{\bullet}_{x:\{*\}} 1 : X(\boldsymbol{g}) \end{aligned}$$

Clearly, X behaves precisely the same as without the final summand, except that there is a selfloop labelled *conditionReached* in every state such that the global variables satisfy condition c. Of course we should be careful never to hide the action *conditionReached* (renaming it to τ), as in combination with the maximal progress assumption this would disable all Markovian summands. Actually, in our implementation we only use such actions for finding a set of goal states; during state space generation, they are omitted to get precisely the same MA as we would have obtained without the additional summand. □

4.2.5 Probabilistic Common Representation Language

The MAPA specification language we just presented can be used to model systems with nondeterminism, probability and Markovian rates. If a system does not contain any rates, a subset of MAPA suffices. In fact, before defining MAPA in [TKvdPS12a], we had already introduced such a subset under the name *probabilistic Common Representation Language* (prCRL) in [KvdPST10a, KvdPST12]. The syntax of prCRL coincides with the syntax of MAPA, except that it does not contain the $(\lambda) \cdot p$ construct.

Definition 4.30 (Process terms in prCRL). *A* process term *in prCRL is any term that can be generated by the following grammar:*

$$p ::= Y(\boldsymbol{t}) \;\mid\; c \Rightarrow p \;\mid\; p + p \;\mid\; \sum_{\boldsymbol{x}:\boldsymbol{D}} p \;\mid\; a(\boldsymbol{t}) \textstyle\sum\limits^{\bullet}_{\boldsymbol{x}:\boldsymbol{D}} f : p$$

Assuming this restricted syntax, all MAPA concepts defined before (specifications, well-formedness, operational semantics, syntactic sugar) can be applied unchanged. Note that for prCRL the SOS rule MStep is never used, and that the concept of derivations is rendered superfluous.

It is often more convenient to define and prove correct a (reduction) technique on prCRL than on MAPA, due to the absence of Markovian transitions. Therefore, we present an encoding scheme that allows us to encode a MAPA specification M into a prCRL specification $\mathsf{enc}\,(M)$, apply some transformation f on it and then decode back to a MAPA specification $\mathsf{dec}\,(f(\mathsf{enc}\,(M)))$ (see Figure 4.2 on page 59). For a transformation f on prCRL specification to be applicable to MAPA specifications in this manner, we often need to take care that $\mathsf{dec}\,(f(\mathsf{enc}\,(M)))$ is strongly bisimilar to M for every MAPA specification M. To achieve this, we introduce a novel notion of *derivation-preserving bisimulation* on prCRL terms and show that, for any prCRL transformation f that respects this notion, $\mathsf{dec} \circ f \circ \mathsf{enc}$ indeed preserves strong bisimulation.

Encoding and decoding. The encoding of MAPA terms is straightforward. The construct $(\lambda) \cdot p$ of MAPA is the only one that has to be encoded, since the other constructs all are also present in prCRL. We chose to encode each exponential rate λ by a parameterised action $\mathsf{rate}(\lambda)$ (which is assumed not to occur in the original specification). Since actions in prCRL require a probabilistic choice for the next state, we use $\sum^{\bullet}_{x:\{1\}} 1 : p$ such that x is not used in p. Figure 4.6 shows the appropriate encoding and decoding functions.

Remark 4.31. We only provide a decoding rule for rate-actions in a construct of the form $\mathsf{rate}(\lambda)\sum^{\bullet}_{\boldsymbol{x}:\{*\}} 1 : p$, and cannot decode any construct of the form $\mathsf{rate}(\lambda)\sum^{\bullet}_{\boldsymbol{x}:D} f : p$ with $D \neq \{*\}$ or $f \neq 1$. We say that a prCRL specification P is *decodable* if it can be decoded, otherwise it is *non-decodable*.

Note that the encoding of a MAPA specification without any rate actions is always decodable. Also note that transformations on encoded MAPA specifications are not allowed to modify the data types D or probability expressions f of the existing rate-constructs or add additional rate-constructs of a different

$$\begin{array}{llll}
\mathsf{enc}\,(Y(\boldsymbol{t})) & = Y(\boldsymbol{t}) & \mathsf{dec}\,(Y(\boldsymbol{t})) & = Y(\boldsymbol{t}) \\
\mathsf{enc}\,(c \Rightarrow p) & = c \Rightarrow \mathsf{enc}\,(p) & \mathsf{dec}\,(c \Rightarrow p) & = c \Rightarrow \mathsf{dec}\,(p) \\
\mathsf{enc}\,(p + q) & = \mathsf{enc}\,(p) + \mathsf{enc}\,(q) & \mathsf{dec}\,(p + q) & = \mathsf{dec}\,(p) + \mathsf{dec}\,(q) \\
\mathsf{enc}\,\left(\sum_{\boldsymbol{x}:\boldsymbol{D}} p\right) & = \sum_{\boldsymbol{x}:\boldsymbol{D}} \mathsf{enc}\,(p) & \mathsf{dec}\,\left(\sum_{\boldsymbol{x}:\boldsymbol{D}} p\right) & = \sum_{\boldsymbol{x}:\boldsymbol{D}} \mathsf{dec}\,(p) \\
\mathsf{enc}\,\left(a(\boldsymbol{t}) \textstyle\sum\nolimits^{\bullet}_{\boldsymbol{x}:\boldsymbol{D}} f : p\right) & = a(\boldsymbol{t}) \textstyle\sum\nolimits^{\bullet}_{\boldsymbol{x}:\boldsymbol{D}} f : \mathsf{enc}\,(p) & \mathsf{dec}\,\left(a(\boldsymbol{t}) \textstyle\sum\nolimits^{\bullet}_{\boldsymbol{x}:\boldsymbol{D}} f : p\right) & = a(\boldsymbol{t}) \textstyle\sum\nolimits^{\bullet}_{\boldsymbol{x}:\boldsymbol{D}} f : \mathsf{dec}\,(p) \\
 & & & \quad (a \neq \mathsf{rate})
\end{array}$$

$$\begin{array}{lll}
\mathsf{enc}\,((\lambda) \cdot p) & = \mathsf{rate}(\lambda) \textstyle\sum\nolimits^{\bullet}_{x:\{1\}} 1 : \mathsf{enc}\,(p) & (x \text{ does not occur in } p) \\
\mathsf{dec}\,(\mathsf{rate}(\lambda) \textstyle\sum\nolimits^{\bullet}_{\boldsymbol{x}:\{*\}} 1 : p) & = (\lambda) \cdot \mathsf{dec}\,(p) &
\end{array}$$

Figure 4.6: Encoding and decoding rules for process terms.

form, as this would make the specification non-decodable. Indeed, none of our techniques do so. □

Definition 4.32 (Encoding). *Given a MAPA specification M and a decodable prCRL specification P, specified by*

$$M = (\{X_i(\boldsymbol{x_i} : \boldsymbol{D_i}) = p_i\}, X_j(\boldsymbol{t}))$$
$$P = (\{Y_i(\boldsymbol{y_i} : \boldsymbol{E_i}) = q_i\}, Y_j(\boldsymbol{u}))$$

we define

$$\mathsf{enc}\,(M) = (\{X_i(\boldsymbol{x_i} : \boldsymbol{D_i}) = \mathsf{enc}\,(p_i)\}, X_j(\boldsymbol{t}))$$
$$\mathsf{dec}\,(P) = (\{Y_i(\boldsymbol{y_i} : \boldsymbol{E_i}) = \mathsf{dec}\,(q_i)\}, Y_j(\boldsymbol{u}))$$

where the functions enc *and* dec *for process terms are given in Figure 4.6.*

It is easy to see that well-formed MAPA specifications encode to well-formed prCRL specifications. For the reverse, in addition to being well-formed we also need to require prCRL specifications to not contain infinite summations over rate actions that would decode to infinite outgoing rates. Since such constructions can never arise from the encoding of well-formed MAPA specifications, and derivation-preserving bisimulation (as discussed below) does not allow them to be added, this issue does not concern us.

It may appear that, given the above encoding and decoding rules, strongly bisimilar prCRL specifications always decode to strongly bisimilar MAPA specifications. However, this is not the case. Consider the strongly bisimilar prCRL terms $\mathsf{rate}(\lambda) \cdot X + \mathsf{rate}(\lambda) \cdot X$ and $\mathsf{rate}(\lambda) \cdot X$. The decodings of these two terms, $(\lambda) \cdot X + (\lambda) \cdot X$ and $(\lambda) \cdot X$, are clearly not strongly bisimilar in the context of MAPA as they have different rates to go to X.

An obvious solution may seem to encode each rate by a unique action, yielding $\mathsf{rate}_1(\lambda) \cdot X + \mathsf{rate}_2(\lambda) \cdot X$ and preventing the above erroneous reduction. However, this does not work in all occasions either. Take for instance a MAPA specification consisting of the two processes $X = Y + Y$ and $Y = (\lambda) \cdot X$. Encoding this to $X = Y + Y$ and $Y = \mathsf{rate}_1(\lambda) \cdot X$ enables the reduction to $X = Y$ and $Y = \mathsf{rate}_1(\lambda) \cdot X$, which is still incorrect since it halves the rate of X.

Note that an 'encoding scheme' that does yield bisimilar MAPA specifications for bisimilar prCRL specifications exists. We could generate the complete state space of a MAPA specification (in case this is finite), determine the total rate from p to p' for every pair of process terms p, p', and encode each of these as a unique action in the prCRL specification. When decoding, potential copies of this action that may arise when looking at bisimilar specifications can then just be ignored. However, this clearly renders useless the whole idea of reducing a linear specification before generation of the entire state space.

Derivation-preserving bisimulation. The observations above suggest that we need a stronger notion of bisimulation if we want two strongly bisimilar prCRL specifications to decode to strongly bisimilar MAPA specifications: all bisimilar process terms should have an equal number of $\mathsf{rate}(\lambda)$ derivations to every equivalence class (as given by the bisimulation relation). We formalise this by means of a *derivation-preserving bisimulation*[6]. It is defined on prCRL terms instead of a PA, since the number of derivations is not encoded in the PA semantics of prCRL.

Definition 4.33 (Derivation preservation). *Let R be a strong bisimulation relation over decodable prCRL process terms. Then, R is* derivation preserving *if for every pair $(p, q) \in R$, every equivalence class $[r]_R$ and every rate λ:*

$$\begin{aligned} &|\{\mathcal{D} \in \Delta \mid \exists r' \in [r]_R \,.\, p \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mathbb{1}_{r'}\}| = \\ &|\{\mathcal{D} \in \Delta \mid \exists r' \in [r]_R \,.\, q \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mathbb{1}_{r'}\}| \end{aligned}$$

Two prCRL terms p, q are derivation-preserving bisimilar*, denoted by $p \sim_{\text{dp}} q$, if there exists a derivation-preserving bisimulation relation R such that $(p, q) \in R$.*

It can be shown that $\sim_{\text{dp}}$ is an equivalence relation in a way similar to the proof of Proposition 3.33.

Note that—as above for the decoding rules—we assume decodable prCRL process terms. Hence, the definition of derivation preservation does not need to take into account $\mathsf{rate}(\lambda)$-transitions with a non-Dirac target distribution.

Example 4.34. Consider the three prCRL process terms

$$\begin{aligned} p_1 &= \mathsf{rate}(3) \cdot a \cdot X + \mathsf{rate}(3) \cdot a \cdot X \\ p_2 &= \mathsf{rate}(3) \cdot a \cdot X \\ p_3 &= \mathsf{rate}(3) \cdot a \cdot X + \mathsf{rate}(3) \cdot (a \cdot X + a \cdot X) \end{aligned}$$

It can easily be seen that $p_1 \approx_{\text{iso}} p_2$. However, the obvious bisimulation relation obtained by taking the smallest equivalence relation R containing (p_1, p_2) is not derivation preserving. To see why, consider the pair (p_1, p_2), the equivalence class $[a \cdot X]_R = \{a \cdot X\}$ and the rate 3. We find that

$$\begin{aligned} &|\{\mathcal{D} \in \Delta \mid \exists p \in [a \cdot X]_R \,.\, p_1 \xrightarrow{\mathsf{rate}(3)}_{\mathcal{D}} \mathbb{1}_p\}| \\ &= |\{\langle \text{NChoiceL}, \text{MStep}\rangle, \langle \text{NChoiceR}, \text{MStep}\rangle\}| \end{aligned}$$

[6] We could be a bit more liberal—although technically slightly more involved—than the current definition, only requiring equal sums of the λs of all rate-transitions to each equivalence class.

$$
\begin{aligned}
&= 2 \neq 1 \\
&= |\{\langle \text{MStep} \rangle\}| \\
&= |\{\mathcal{D} \in \Delta \mid \exists p \in [a \cdot X]_R \, . \, p_2 \xrightarrow{\mathsf{rate}(3)}_{\mathcal{D}} \mathbb{1}_p\}|
\end{aligned}
$$

No derivation-preserving bisimulation relation can be found for p_1, p_2, and hence isomorphism in the prCRL setting does not imply derivation-preserving bisimulation. This is desirable, since p_1 and p_2 indeed do not decode to bisimilar MAPA process terms.

On the other hand, while p_1 and p_3 are not isomorphic, they are derivation-preserving bisimilar. Consider the bisimulation relation obtained by taking the smallest equivalence relation R containing (p_1, p_3) and $(a \cdot X, a \cdot X + a \cdot X)$. Now, we find that $[a \cdot X]_R = \{a \cdot X, a \cdot X + a \cdot X\}$, and both p_1 and p_3 have two derivations to this class. The fact that $a \cdot X$ and $a \cdot X + a \cdot X$ do not have the same number of derivations to X is irrelevant, as the action is not a rate. □

As expected and desired, derivation-preserving bisimulation is a congruence for every prCRL operator. Hence, prCRL process terms in a specification can be replaced by derivation-preserving bisimilar process terms, retaining a derivation-preserving bisimilar specification.

Theorem 4.35. *Derivation-preserving bisimulation is a congruence for all operators in prCRL.*

Our encoding scheme and notion of derivation-preserving bisimulation allow us to reuse prCRL transformations for MAPA specifications. The next theorem confirms that a function $\mathsf{dec} \circ f \circ \mathsf{enc}$: MAPA $\to$ MAPA respects bisimulation if f: prCRL $\to$ prCRL respects derivation-preserving bisimulation (and does not introduce non-decodable rate-constructs).

Since the full proof (presented in the appendix with all other proofs) is rather complicated, we sketch the main steps.

Theorem 4.36. *Let f: prCRL $\to$ prCRL be a function such that $f(P) \sim_{\mathrm{dp}} P$ for every prCRL specification P, and such that if P is decodable, then so is $f(P)$. Then, $\mathsf{dec}\,(f(\mathsf{enc}\,(M))) \approx_{\mathrm{s}} M$ for every MAPA specification M without any actions labelled by* rate.

Proof (sketch). It can be shown that (a) $p \overset{a}{\hookrightarrow} \mu$ (with $a \neq \mathsf{rate}$) holds for a MAPA process term p if and only if $\mathsf{enc}\,(p) \xrightarrow{a} \mu_{\mathsf{enc}}$ holds for its prCRL encoding $\mathsf{enc}\,(p)$ (recall from Section 2.2.3 that μ_{enc} is the lifting of μ over the function enc), and that (b) every derivation $p \xrightarrow{\lambda}_{\mathcal{D}} p'$ corresponds one-to-one to a derivation $\mathsf{enc}\,(p) \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}'} \mathbb{1}_{\mathsf{enc}(p')}$, with $\mathcal{D}'$ obtained from $\mathcal{D}$ by substituting PSum for MStep. Using these two observations, and taking R as the derivation-preserving bisimulation relation for $f(P) \sim_{\mathrm{dp}} P$ for some arbitrary P, it can be shown that $R' = \{(\mathsf{dec}\,(p), \mathsf{dec}\,(q)) \mid (p, q) \in R\}$ is a bisimulation relation, and hence that $\mathsf{dec}\,(f(P)) \approx_{\mathrm{s}} \mathsf{dec}\,(P)$. As P was arbitrary, this holds for every P. Taking $P = \mathsf{enc}\,(M)$, and noting that $\mathsf{dec}\,(\mathsf{enc}\,(M)) = M$, the theorem follows. □

4.3 Linearisation

Whereas we want to model systems in the full MAPA language as presented in Section 4.2, we prefer to define our reduction techniques and state space generation algorithms on the restricted MLPE format introduced in Section 4.2.4. Hence, we need a way to transform MAPA specifications to an equivalent specification in MLPE format—a transformation called *linearisation*. We note that our algorithms resemble parts of the linearisation procedure developed in [Use02], but that our algorithm is simpler due to the absence of sequential composition. Also, our algorithm will turn out to be derivation-preserving bisimilar, while Usenko's was not (because of the optimisation step that transforms $x + x$ to x).

To simplify matters, we present our linearisation procedure for the prCRL language from Section 4.2.5. We call the resulting object—an MLPE without any Markovian transitions—an *LPPE* (linear probabilistic process equation).

Definition 4.37 (LPPEs). *A* linear probabilistic process equation *(LPPE) is a MAPA specification of the following format:*

$$X(\boldsymbol{g} : \boldsymbol{G}) = \sum_{i \in I} \sum_{\boldsymbol{d_i} : \boldsymbol{D_i}} c_i \Rightarrow a_i(\boldsymbol{b_i}) \sum_{\boldsymbol{e_i} : \boldsymbol{E_i}}\!\!\!\bullet\; f_i : X(\boldsymbol{n_i})$$

Note that, since an LPPE does not contain any rates, it is also a prCRL specification. As we will show that the linearisation procedure for prCRL preserves derivations, Theorem 4.36 tells us that this technique can just as easily be applied to linearise MAPA specifications via encoding and decoding.

Linearisation of a prCRL specification P is performed in two steps. In the first step, a specification P' is created such that $P' \sim_{\text{dp}} P$ and P' is in so-called *intermediate regular form* (IRF). Basically, this form requires every right-hand side to be a summation of process terms, each of which contains exactly one action. This step is performed by Algorithm 1 (page 87), which uses Algorithms 2 and 3 (page 88 and page 90). In the second step, an MLPE X is created from P' such that $X \sim_{\text{dp}} P'$. This step is performed by Algorithm 4 (page 94).

We first illustrate both steps by two examples.

Example 4.38. Consider the specification $P = (\{X = a \cdot b \cdot c \cdot X\}, X)$. The behaviour of P does not change if we introduce a new process $Y = b \cdot c \cdot X$ and let X call Y after its action a. Splitting the new process as well, we obtain the derivation-preserving bisimilar specification

$$P' = (\{X = a \cdot Y, Y = b \cdot Z, Z = c \cdot X\}, X)$$

As required, every right-hand side is a summation of process terms that have precisely one action. Now, an isomorphic LPPE is constructed by introducing a program counter *pc* that keeps track of the subprocess that is currently active, as shown below (similar to Usenko's transformation in [Use02, Section 4.3.2]).

$$\begin{aligned} P''(pc : \{1,2,3\}) = {} & pc = 1 \Rightarrow a \cdot P''(2) \\ & + pc = 2 \Rightarrow b \cdot P''(3) \\ & + pc = 3 \Rightarrow c \cdot P''(1) \end{aligned}$$

It is easy to see that $P''(1)$ is isomorphic to P, and—since it does not contain any rate-actions—also derivation-preserving bisimilar. □

Example 4.39. Now consider the following specification, consisting of two processes with parameters. Let $X(d')$ be the initial process for some arbitrary $d' \in D$. (The types D and E are assumed to be finite, have addition defined on them and contain the element 1—we could for instance take an interval $\{0, 1, 2, \dots, n-1\}$ with addition defined modulo n).

$$\begin{aligned}
&X(d : D)\\
&= \text{choose}\textstyle\sum_{e:E} \frac{1}{|E|} : \text{send}(d+e)\textstyle\sum_{i:\{1,2\}}(\text{if } i = 1 \text{ then } 0.9 \text{ else } 0.1) :\\
&\qquad\qquad\qquad\qquad\qquad\qquad ((i = 1 \Rightarrow Y(d+1)) +\\
&\qquad\qquad\qquad\qquad\qquad\qquad\ (i = 2 \Rightarrow \text{crash}\textstyle\sum_{j:\{*\}} 1 : X(d)))\\
&Y(f : D)\\
&= \text{write}(f)\textstyle\sum_{k:\{*\}} 1 : \sum_{g:D}\text{write}(f+g)\sum_{l:\{*\}} 1 : X(f+g)
\end{aligned}$$

Again, we introduce a new process for each subprocess. They all get the same process variables $(d : D, f : D, e : E, i : \{1, 2\})$, which for lay-out purposes we abbreviate by $(\boldsymbol{p})$. The new initial process is $X_1(d', f', e', i')$, where $f' \in D$, $e' \in E$, and $i' \in \{1, 2\}$ can be chosen arbitrarily (and d' should correspond to the original initial value d'). In the specification below, the values d', f', e', i' correspond to these concrete initial values.

$$\begin{aligned}
X_1(\boldsymbol{p}) &= \text{choose}\textstyle\sum_{e:E} \frac{1}{|E|} : X_2(d, f', e, i')\\
X_2(\boldsymbol{p}) &= \text{send}(d+e)\textstyle\sum_{i:\{1,2\}}(\text{if } i = 1 \text{ then } 0.9 \text{ else } 0.1) : X_3(d, f', e', i)\\
X_3(\boldsymbol{p}) &= (i = 1 \Rightarrow \text{write}(d+1)\textstyle\sum_{k:\{*\}} 1 : X_4(d', d+1, e', i'))\\
&\quad + (i = 2 \Rightarrow \text{crash}\textstyle\sum_{j:\{*\}} 1 : X_1(d, f', e', i'))\\
X_4(\boldsymbol{p}) &= \textstyle\sum_{g:D}\text{write}(f+g)\sum_{l:\{*\}} 1 : X_1(f+g, f', e', i')
\end{aligned}$$

Note that we added process variables to store the values of local variables that were bound by a nondeterministic or probabilistic summation. As the index variables j, k and l are never used, and g is only used directly after the summation that binds it, they are not stored. We reset variables that are not syntactically used in their scope to keep the state space small by changing them to their initial value. The idea of giving all processes the same parameters stems from [Use02, Section 4.3.1]. Again, the LPPE is obtained by introducing a program counter. Its initial vector is $(1, d', f', e', i')$.

$$\begin{aligned}
&X(pc : \{1, 2, 3, 4\}, d : D, f : D, e : E, i : \{1, 2\}) =\\
&\qquad\quad pc = 1 \qquad\quad\ \Rightarrow \text{choose}\textstyle\sum_{e:E} \frac{1}{|E|} : X(2, d, f', e, i')\\
&+ \qquad pc = 2 \qquad\quad\ \Rightarrow \text{send}(d+e)\textstyle\sum_{i:\{1,2\}}(\text{if } i = 1 \text{ then } 0.9 \text{ else } 0.1) :\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad X(3, d, f', e', i)\\
&+ \qquad pc = 3 \wedge i = 1 \Rightarrow \text{write}(d+1)\textstyle\sum_{k:\{*\}} 1 : X(4, d', d+1, e', i')
\end{aligned}$$

$$
\begin{aligned}
&+ \qquad pc = 3 \wedge i = 2 \Rightarrow \text{crash}\textstyle\sum_{j:\{*\}} 1 : X(1, d, f', e', i') \\
&+ \textstyle\sum_{g:D} pc = 4 \qquad\quad \Rightarrow \text{write}(f + g)\textstyle\sum_{l:\{*\}} 1 : X(1, f + g, f', e', i') \qquad \square
\end{aligned}
$$

4.3.1 Transforming from prCRL to IRF

We now formally define the intermediate regular form (IRF), and then discuss the transformation from prCRL to IRF in more detail.

Definition 4.40. *A process term is in* IRF *if it adheres to the following grammar:*

$$p ::= c \Rightarrow p \mid p + p \mid \sum_{\boldsymbol{x}:\boldsymbol{D}} p \mid a(\boldsymbol{t}) \sum_{\boldsymbol{x}:\boldsymbol{D}} f : Y(\boldsymbol{t})$$

A process equation is in IRF if its right-hand side is in IRF, and a specification is in IRF if all its process equations are in IRF and all its processes have the same process variables.

Note that in IRF every probabilistic sum goes to a process instantiation, and that process instantiations do not occur in any other way. Therefore, every process instantiation is preceded by exactly one action.

For every specification P there exists a specification P' in IRF such that $P \sim_{\text{dp}} P'$ (since we provide an algorithm—proven correct in Theorem 4.45—to construct it). However, it is not hard to see that P' is not unique.

Remark 4.41. It is not necessarily true that $P \approx_{\text{iso}} P'$, as we will show in Example 4.46. Still, every specification P representing a finite PA *can* be transformed to an IRF describing an isomorphic PA: define a data type S with an element s_i for every reachable state of the PA underlying P, and create a process $X(s : S)$ consisting of a summation of terms of the form

$$s = s_i \Rightarrow a(\boldsymbol{t})(p_1 : s_1 \oplus p_2 : s_2 \oplus \ldots \oplus p_n : s_n)$$

(one for each transition $s_i \xrightarrow{a(\boldsymbol{t})} \mu$, where $\mu(s_1) = p_1, \mu(s_2) = p_2, \ldots, \mu(s_n) = p_n$). However, this transformation completely defeats its purpose, as the whole idea behind the LPPE is to apply reductions *before* having to compute all states of the original specification. $\square$

Overview of the transformation to IRF. Algorithm 1 (page 87) transforms a specification P to a specification P', in such a way that $P \sim_{\text{dp}} P'$ and P' is in IRF. It requires that all process variables and local variables of P have unique names (which is easily achieved by renaming variables having names that are used more than once). Three important variables are used: (1) *done* is a set of process equations that are already in IRF; (2) *toTransform* is a set of process equations that still have to be transformed to IRF; (3) *bindings* is a set of process equations $\{X'_i(\mathit{pars}) = p_i\}$ such that $X'_i(\mathit{pars})$ is the process in $\mathit{done} \cup \mathit{toTransform}$ representing the process term p_i of the original specification.

Algorithm 1: Transforming a specification to IRF

Input:

- A prCRL specification

$$\begin{aligned} P = (\{&X_1(\boldsymbol{x_1} : \boldsymbol{D_1}) = p_1, \\ &X_2(\boldsymbol{x_2} : \boldsymbol{D_2}) = p_2, \\ &\dots, \\ &X_n(\boldsymbol{x_n} : \boldsymbol{D_n}) = p_n\}, X_1(\boldsymbol{v})) \end{aligned}$$

  in which all variables (either declared as a process variable, or bound by a nondeterministic or probabilistic sum) are named uniquely.

Output:

- A prCRL specification

$$\begin{aligned} P' = (\{&X'_1(\boldsymbol{x_1} : \boldsymbol{D_1}, \boldsymbol{x'} : \boldsymbol{D'}) = p'_1, \dots, \\ &X'_k(\boldsymbol{x_1} : \boldsymbol{D_1}, \boldsymbol{x'} : \boldsymbol{D'}) = p'_k\}, X'_1(\boldsymbol{v'})) \end{aligned}$$

  such that P' is in IRF and $P' \sim_{\text{dp}} P$. Here, $\boldsymbol{x'} : \boldsymbol{D'}$ contains all global variables of $X_2, \dots, X_n$, as well as some additional variables to store nondeterministic and probabilistic choices.

```
Initialisation
1   $[(y_1, E_1), \dots, (y_m, E_m)] = [(y, E) \mid \exists i \,.\, p_i$ binds variable $y$ of type $E$ by
                                      a nondeterministic or probabilistic
                                      sum, and syntactically uses $y$ in the
                                      scope of that operator$]$
2   pars := $(\boldsymbol{x_1} : \boldsymbol{D_1}, \boldsymbol{x_2} : \boldsymbol{D_2}, \dots, \boldsymbol{x_n} : \boldsymbol{D_n}, y_1 : E_1, \dots, y_m : E_m)$
3   $\boldsymbol{v'} := \boldsymbol{v}$ ++ [any constant of type $D \mid D \leftarrow [\boldsymbol{D_2}, \dots, \boldsymbol{D_n}, E_1, \dots, E_m]$]
4   done := $\varnothing$
5   toTransform := $\{X'_1(\text{pars}) = p_1\}$
6   bindings := $\{X'_1(\text{pars}) = p_1\}$
Construction
7   **while** toTransform $\neq \varnothing$ **do**
8       Choose an arbitrary equation $(X'_i(\text{pars}) = p_i) \in$ toTransform
9       $(p'_i, \text{newProcs}) := \text{transform}(p_i, \text{pars}, \text{bindings}, P, \boldsymbol{v'})$
10      done := done $\cup \{X'_i(\text{pars}) = p'_i\}$
11      bindings := bindings $\cup$ newProcs
12      toTransform := (toTransform $\cup$ newProcs) $\setminus \{X'_i(\text{pars}) = p_i\}$
13  **return** (done, $X'_1(\boldsymbol{v'})$)
```

Initially, *pars* is assigned the vector of all variables declared in P, either globally or in a summation (and syntactically used after being bound), to-

Algorithm 2: Transforming process terms to IRF

Input:

- A process term p;
- A list *pars* of typed process variables;
- A set *bindings* of (unchanged) process terms in P that have already been mapped to a new process;
- A specification P;
- An initial vector $\boldsymbol{v}'$.

Output:

- An IRF for p;
- The process equations to add to *toTransform*.

```
   transform($p$, pars, bindings, $P, \boldsymbol{v}'$) =
1    case $p = a(\boldsymbol{t}) \sum_{\boldsymbol{x}:\boldsymbol{D}} f : q$
2      $(q'$, actualPars) := normalForm($q$, pars, $P, \boldsymbol{v}'$)
3      if $\exists j \,.\, (X'_j$(pars) $= q') \in$ bindings then
4        return $(a(\boldsymbol{t}) \sum_{\boldsymbol{x}:\boldsymbol{D}} f : X'_j$(actualPars), $\varnothing)$
5      else
6        return $(a(\boldsymbol{t}) \sum_{\boldsymbol{x}:\boldsymbol{D}} f : X'_k$(actualPars), $\{(X'_k$(pars) $= q')\})$
             where $k = |$bindings$| + 1$
7    case $p = c \Rightarrow q$
8      (newRHS, newProcs) := transform($q$, pars, bindings, $P, \boldsymbol{v}'$)
9      return ($c \Rightarrow$ newRHS, newProcs)
10   case $p = q_1 + q_2$
11     (newRHS$_1$, newProcs$_1$) := transform($q_1$, pars, bindings, $P, \boldsymbol{v}'$)
12     (newRHS$_2$, newProcs$_2$) := transform($q_2$, pars, bindings $\cup$ newProcs$_1$,
                                                     $P, \boldsymbol{v}'$)
13     return (newRHS$_1$ + newRHS$_2$, newProcs$_1 \cup$ newProcs$_2$)
14   case $p = Y(\boldsymbol{t})$
15     (newRHS, newProcs) := transform(RHS($Y$), pars, bindings, $P, \boldsymbol{v}'$)
16     newRHS’ = newRHS, with all free variables substituted by the value
                                  provided for them by $\boldsymbol{t}$
17     return (newRHS’, newProcs)
18   case $p = \sum_{\boldsymbol{x}:\boldsymbol{D}} q$
19     (newRHS, newProcs) := transform($q$, pars, bindings, $P, \boldsymbol{v}'$)
20     return ($\sum_{\boldsymbol{x}:\boldsymbol{D}}$ newRHS, newProcs)
```

gether with the corresponding type. The new initial vector $\boldsymbol{v}'$ is constructed by appending dummy values to the original initial vector for all added variables (denoted by Haskell-like list comprehension). Also, *done* is empty, the right-hand side of the initial process is bound to $X'_1(pars)$, and this equation is added to *toTransform* and *bindings*. Then, we repeatedly take an equation $X_i'(\mathit{pars}) = p_i$ from *toTransform*, transform p_i to a strongly probabilistically bisimilar IRF p_i' using Algorithm 2, add the equation $X_i'(\mathit{pars}) = p_i'$ to *done*, and remove $X_i'(\mathit{pars}) = p_i$ from *toTransform*. The transformation may introduce new processes, which are added to *toTransform*, and *bindings* is updated accordingly.

Remark 4.42. The use of the set *bindings* makes sure that multiple occurrences of the same process term are linearised only once. For instance, when transforming a process term such as $x \cdot b \cdot c \cdot X + y \cdot b \cdot c \cdot X$, we want to make use of the duplication, transforming to $x \cdot X_2' + y \cdot X_2'$, with $X_2' = b \cdot c \cdot X$ (which is initially put in *toTransform* and later transformed itself). To this end, we add the equation $X_2' = b \cdot c \cdot X$ to *bindings* after transforming the left-hand side of the nondeterministic choice to $x \cdot X_2'$. Then, when transforming the right-hand side, this information is used to transform to $y \cdot X_2'$ instead of introducing a new process X_3'.

Additionally, the *bindings* set makes sure that process instantiations are transformed correctly while still terminating. For instance, when transforming a process such as $X = a \cdot b \cdot c \cdot X$, the algorithm starts by putting the equation $X_1' = a \cdot b \cdot c \cdot X$ in both *bindings* and *toTransform*. When transforming the right-hand side of this equation, at some point we will have to transform the process term $c \cdot X$. It is then checked whether the right-hand side of X, in this case $a \cdot b \cdot c \cdot X$, is already present in *bindings*. This is the case (in the form $X_1' = a \cdot b \cdot c \cdot X$), so the instantiation X is changed to X_1'.

In a more complicated situation where X has data parameters, the algorithm still works in the same way. For instance, consider the process $X(n : \mathbb{N}) = a \cdot b \cdot X(n+1) + b \cdot X(5)$. In the initialisation, $X_1'(n : \mathbb{N}) = a \cdot b \cdot X(n+1) + b \cdot X(5)$ is put in *bindings*. When later on $b \cdot X(n+1)$ needs to be transformed, it is recognised that $X(n+1)$ is an instantiation of a process with right-hand side $a \cdot b \cdot X(n+1) + b \cdot X(5)$. This process term is already present in *bindings*, so the algorithm transforms $b \cdot X(n+1)$ to $b \cdot X_1'(n+1)$.

Transforming single process terms to IRF. Algorithm 2 transforms individual process terms to IRF recursively by means of a case distinction over the structure of the terms (using Algorithm 3). Its base case is the action prefix, which is always reached in a finite number of steps due to the requirement of guardedness (Definition 4.16).

For a summation $q_1 + q_2$, the IRF is $q_1' + q_2'$ (with q_i' an IRF of q_i). For the condition $c \Rightarrow q_1$ it is $c \Rightarrow q_1'$, and for $\sum_{\boldsymbol{x}:\boldsymbol{D}} q_1$ it is $\sum_{\boldsymbol{x}:\boldsymbol{D}} q_1'$. Finally, the IRF for $Y(\boldsymbol{t})$ is the IRF for the right-hand side of Y, where the global variables of Y occurring in this term have been substituted by the expressions given by $\boldsymbol{t}$.

The base case is a probabilistic choice $a(\boldsymbol{t}) \textstyle\sum\!\!\!\!\bullet_{\boldsymbol{x}:\boldsymbol{D}} f : q$. The corresponding process term in IRF depends on whether or not there already is a process name X_j' mapped to q (as stored in *bindings*). If this is the case, apparently q has been linearised before and the result simply is $a(\boldsymbol{t}) \textstyle\sum\!\!\!\!\bullet_{\boldsymbol{x}:\boldsymbol{D}} f : X_j'(\mathit{actualPars})$, with *actualPars* as explained below. If q was not linearised before, a new process name X_k' is chosen, the result is $a(\boldsymbol{t}) \textstyle\sum\!\!\!\!\bullet_{\boldsymbol{x}:\boldsymbol{D}} f : X_k'(\mathit{actualPars})$ and X_k' is mapped

Algorithm 3: Normalising process terms

Input:

- A process term p;
- A list *pars* of typed global variables;
- A prCRL specification P;
- An initial vector $\boldsymbol{v}' = (v'_1, v'_2, \dots, v'_k)$.

Output:

- The normal form of p;
- The actual parameters needed to supply to a process which has right-hand side p' to make its behaviour derivation-preserving bisimilar to p.

```
normalForm($p$, pars, $P$, $\boldsymbol{v}'$) =
1  **case** $p = Y(t_1, t_2, \dots, t_n)$
2    **return**(RHS($Y$), [inst($v$) | ($v, D$) ← pars])
       where $\text{inst}(v) = \begin{cases} t_i & \text{if } v \text{ is the } i^{\text{th}} \text{ global variable of } Y \text{ in } P \\ v'_i & \text{if } v \text{ is not a global variable of } Y \text{ in } P, \\ & \text{and } v \text{ is the } i^{\text{th}} \text{ element of pars} \end{cases}$
3  **case** otherwise
4    **return** ($p$, [inst′($v$) | ($v, D$) ← pars])
       where $\text{inst}'(v) = \begin{cases} v & \text{if } v \text{ occurs syntactically in } p \\ v'_i & \text{if } v \text{ does not occur syntactically in } p, \\ & \text{and } v \text{ is the } i^{\text{th}} \text{ element of pars} \end{cases}$
```

to q by adding this information to *bindings*. Since a newly created process X'_k is added to *toTransform*, in a next iteration of Algorithm 1 it will be linearised.

More precisely, instead of q we use its *normal form*, computed by Algorithm 3. The reason behind this is that, when linearising a process in which for instance both the process instantiations $X(n)$ and $X(n+1)$ occur, we do not want to have a distinct term for both of them. We therefore define the normal form of a process instantiation $Y(\boldsymbol{t})$ to be the right-hand side of Y, and of any other process term q to just be q. This way, different process instantiations of the same process and the right-hand side of that process all have the same normal form, and no duplicate terms are generated.

Algorithm 3 is also used to determine the actual parameters that have to be provided to either X'_j (if q was already linearised before) or to X'_k (if q was not linearised before). This depends on whether or not q is a process instantiation. If it is not, the actual parameters for X'_j are just the global variables (possibly resetting variables that are not used in q). If it is, for instance $q = Y(t_1, t_2, \dots, t_n)$, all global variables are reset, except the ones corresponding to the original global variables of Y; for them $t_1, t_2, \dots, t_n$ are used.

Note that in Algorithm 3 we use $(v, D) \leftarrow$ *pars* to denote the list of all pairs

(v_i, D_i), given $pars = (v_1, \dots, v_n) : (D_1 \times \dots \times D_n)$. We use RHS$(Y)$ for the right-hand side of the process equation defining Y.

Example 4.43. We linearise two example specifications:

$$\begin{aligned} P_1 &= (\{X_1 = a \cdot b \cdot c \cdot X_1 + c \cdot X_2, X_2 = a \cdot b \cdot c \cdot X_1\}, X_1) \\ P_2 &= (\{X_3(d : D) = \sum_{e:D} a(d + e) \cdot c(e) \cdot X_3(5)\}, X_3(d')) \end{aligned}$$

Table 4.7 on page 92 shows *done*, *toTransform* and *bindings* at line 7 of Algorithm 1 for every iteration. As *done* and *bindings* only grow, we just list their additions. For layout purposes, we omit the parameters $(d : D, e : D)$ of every X''_i in the lower part of Table 4.7. The results in IRF are $P'_1 = (done_1, X'_1)$ and $P'_2 = (done_2, X''_1(d', e'))$ for an arbitrary $e' \in D$. □

Example 4.44. We illustrate the case $p = Y(\boldsymbol{t})$ of Algorithm 2 by means of the following example:

$$\begin{aligned} P = (\{&X = a \cdot X + Y(5), \\ &Y(n : \mathbb{N}) = a(n) \cdot b(n) \cdot Y(n + 1)\}, X) \end{aligned}$$

For this specification, our algorithm yields

$$\begin{aligned} P' = (\{&X'_1(n : \mathbb{N}) = a \cdot X'_1(0) + a(5) \cdot X'_2(5), \\ &X'_2(n : \mathbb{N}) = b(n) \cdot X'_3(n + 1), \\ &X'_3(n : \mathbb{N}) = a(n) \cdot X'_2(n)\}, X'_1(0)) \end{aligned}$$

First, the right-hand side of Y was transformed to $a(n) \cdot X'_2(n)$. Then, the value 5 provided by the call $Y(5)$ was substituted for n, yielding $a(5) \cdot X'_2(5)$.□

The following theorem states the correctness of our transformation.

Theorem 4.45. *Let P be a decodable prCRL specification such that all variables are named uniquely. Given this input, Algorithm 1 terminates and provides a specification P' such that $P' \sim_{\text{dp}} P$, P' is in IRF and P' is decodable.*

Algorithm 1 does not always compute an isomorphic specification, as illustrated by the following example.

Example 4.46. Let $P = (\{X = \sum_{d:D} a(d) \cdot b(f(d)) \cdot X\}, X)$, with $f(d) = 0$ for all $d \in D$. Then, our procedure will yield the specification

$$\begin{aligned} P' = (\{&X'_1(d : D) = \sum_{d:D} a(d) \cdot X'_2(d), \\ &X'_2(d : D) = b(f(d)) \cdot X'_1(d')\}, X'_1(d')) \end{aligned}$$

for some $d' \in D$. Note that the reachable number of states of P' is $|D| + 1$ for any $d' \in D$. However, the reachable state space of P only consists of the two states X and $b(0) \cdot X$. □

	$done_1$	$toTransform_1$	$bindings_1$
0	$\varnothing$	$X_1' = a \cdot b \cdot c \cdot X_1 + c \cdot X_2$	$X_1' = a \cdot b \cdot c \cdot X_1 + c \cdot X_2$
1	$X_1' = a \cdot X_2' + c \cdot X_3'$	$X_2' = b \cdot c \cdot X_1,\ X_3' = a \cdot b \cdot c \cdot X_1$	$X_2' = b \cdot c \cdot X_1,\ X_3' = a \cdot b \cdot c \cdot X_1$
2	$X_2' = b \cdot X_4'$	$X_3' = a \cdot b \cdot c \cdot X_1,\ X_4' = c \cdot X_1$	$X_4' = c \cdot X_1$
3	$X_3' = a \cdot X_2'$	$X_4' = c \cdot X_1$	
4	$X_4' = c \cdot X_1'$	$\varnothing$	

	$done_2$	$toTransform_2$	$bindings_2$
0	$\varnothing$	$X_1'' = \sum_{e:D} a(d+e) \cdot c(e) \cdot X_3(5)$	$X_1'' = \sum_{e:D} a(d+e) \cdot c(e) \cdot X_3(5)$
1	$X_1'' = \sum_{e:D} a(d+e) \cdot X_2''(d', e)$	$X_2'' = c(e) \cdot X_3(5)$	$X_2'' = c(e) \cdot X_3(5)$
2	$X_2'' = c(e) \cdot X_1''(5, e')$	$\varnothing$	

Table 4.7: Transforming $P_1 = (\{X_1 = a \cdot b \cdot c \cdot X_1 + c \cdot X_2, X_2 = a \cdot b \cdot c \cdot X_1\}, X_1)$ (above) and $P_2 = (\{X_3(d : D) = \sum_{e:D} a(d+e) \cdot c(e) \cdot X_3(5)\}, X_3(d'))$ (below) to IRF.

4.3.2 Transforming from IRF to LPPE

Given a specification P' in IRF, Algorithm 4 (page 94) constructs an LPPE X. The global variables of X are a program counter pc and all global variables of P'. To construct the summands for X, we range over the process equations in P'. For each equation

$$X'_i(\boldsymbol{x} : \boldsymbol{D}) = a(\boldsymbol{t}) \sum_{\boldsymbol{y}:\boldsymbol{E}}\!\!\bullet\, f : X'_j(t'_1, \ldots, t'_k)$$

a summand

$$pc = i \Rightarrow a(\boldsymbol{t}) \sum_{\boldsymbol{y}:\boldsymbol{E}}\!\!\bullet\, f : X(j, t'_1, \ldots, t'_k)$$

is constructed. For an equation $X'_i(\boldsymbol{x} : \boldsymbol{D}) = q_1 + q_2$ the union of the summands produced by $X'_i(\boldsymbol{x} : \boldsymbol{D}) = q_1$ and $X'_i(\boldsymbol{x} : \boldsymbol{D}) = q_2$ is taken. For $X'_i(\boldsymbol{x} : \boldsymbol{D}) = c \Rightarrow q$ the condition c is prefixed to the summands produced by $X'_i(\boldsymbol{x} : \boldsymbol{D}) = q$; nondeterministic sums are handled similarly.

To be precise, the specification produced by the algorithm is not literally an LPPE yet, as there may be several conditions and nondeterministic sums, and their order could still be wrong (we call such specifications semi-LPPEs). An isomorphic LPPE is obtained by moving the nondeterministic sums to the front and merging separate nondeterministic sums (using vectors) and separate conditions (using conjunctions). When moving nondeterministic sums to the front, some variable renaming may be needed to avoid clashes with the conditions.

Example 4.47. Looking at the IRFs obtained in Example 4.43, it is easy to see that $P'_1 \sim_{\text{dp}} X$ and $P'_2 \sim_{\text{dp}} Y$, with

$$\begin{aligned} X = (\{X(pc : \{1,2,3,4\}) &= pc = 1 \Rightarrow a \cdot X(2) \\ &+ pc = 1 \Rightarrow c \cdot X(3) \\ &+ pc = 2 \Rightarrow b \cdot X(4) \\ &+ pc = 3 \Rightarrow a \cdot X(2) \\ &+ pc = 4 \Rightarrow c \cdot X(1)\}, \\ X(1)) & \end{aligned} \qquad \begin{aligned} Y = (\{Y(pc : \{1,2\}, d : D, e : D) & \\ = \sum_{e:D} pc = 1 &\Rightarrow a(d+e) \cdot Y(2, d', e) \\ + \qquad pc = 2 &\Rightarrow c(e) \cdot Y(1, 5, e')\}, \\ Y(1, d', e')) & \end{aligned}$$

where again $e' \in D$ can be chosen arbitrarily in the initial vector of Y. □

Theorem 4.48. *Let P' be a decodable specification in IRF without a variable pc, and let the output of Algorithm 4 applied to P' be the specification X. Then, $P' \sim_{\text{dp}} X$ and X is decodable.*

Let Y be like X, except that for each summand all nondeterministic sums have been moved to the beginning while substituting their variables by fresh names, and all separate nondeterministic sums and separate conditions have been merged (using vectors and conjunctions, respectively). Then, Y is an LPPE, $Y \sim_{\text{dp}} X$ and Y is decodable.

Algorithm 4: Constructing an LPPE from an IRF

Input:

- A specification $P' = (\{X'_1(\boldsymbol{x} : \boldsymbol{D}) = p'_1, \dots, X'_k(\boldsymbol{x} : \boldsymbol{D}) = p'_k\}, X'_1(\boldsymbol{v}))$ in IRF (without variable pc).

Output:

- A semi-LPPE $X = (\{X(pc : \{1, \dots, k\}, \boldsymbol{x} : \boldsymbol{D}) = p''\}, X(1, \boldsymbol{v}))$ such that $P' \sim_{\mathrm{dp}} X$.

```
Construction
1   $S = \varnothing$
2   **forall** $(X'_i(\boldsymbol{x} : \boldsymbol{D}) = p'_i) \in P'$ **do**
3       $S := S \cup \text{makeSummands}(p'_i, i)$
4   **return** $(\{X(pc : \{1, \dots, k\}, \boldsymbol{x} : \boldsymbol{D}) = \sum_{s \in S} s\}, X(1, \boldsymbol{v}))$

where
    makeSummands$(p, i)$ =
5       **case** $p = a(\boldsymbol{t}) \textstyle\sum_{\boldsymbol{y}:\boldsymbol{E}} f : X'_j(t'_1, \dots, t'_k)$
6           **return** $\{pc = i \Rightarrow a(\boldsymbol{t}) \textstyle\sum_{\boldsymbol{y}:\boldsymbol{E}} f : X(j, t'_1, \dots, t'_k)\}$
7       **case** $p = c \Rightarrow q$
8           **return** $\{c \Rightarrow q' \mid q' \in \text{makeSummands}(q, i)\}$
9       **case** $p = q_1 + q_2$
10          **return** $\text{makeSummands}(q_1, i) \cup \text{makeSummands}(q_2, i)$
11      **case** $p = \sum_{\boldsymbol{x}:\boldsymbol{D}} q$
12          **return** $\{\sum_{\boldsymbol{x}:\boldsymbol{D}} q' \mid q' \in \text{makeSummands}(q, i)\}$
```

We are now able to conclude that the linearisation procedure presented above can also be used to transform MAPA specifications to MLPEs. Under the observation that a prCRL specification and its linearisation are derivation-preserving bisimilar (which follows from Theorems 4.45 and 4.48), it is an immediate consequence of Theorem 4.36. The fact that a linearised MAPA specification is indeed an MLPE follows from Theorem 4.48 and the observation that decoding does not change the structure of a specification.

Corollary 4.49. *Let M be a MAPA specification without any* rate *action, and*

$$M' = \mathsf{dec}\,(\mathsf{linearise}(\mathsf{enc}\,(M)))$$

Then, $M \approx_{\mathrm{s}} M'$ and M' is an MLPE.

4.3.3 Complexity

To discuss the complexity of linearisation, we first define the size of prCRL specifications.

Definition 4.50. *The size of a process term is defined as follows:*

$$\begin{aligned} \mathit{size}(Y(\boldsymbol{t})) &= 1 + \mathit{size}(\boldsymbol{t}) \\ \mathit{size}(c \Rightarrow p) &= 1 + \mathit{size}(c) + \mathit{size}(p) \\ \mathit{size}(p + q) &= 1 + \mathit{size}(p) + \mathit{size}(q) \\ \mathit{size}(\textstyle\sum_{\boldsymbol{x}:\boldsymbol{D}} p) &= 1 + |\boldsymbol{x}| + \mathit{size}(p) \\ \mathit{size}\big(a(\boldsymbol{t}) \textstyle\sum_{\boldsymbol{x}:\boldsymbol{D}} f : p\big) &= 1 + \mathit{size}(\boldsymbol{t}) + |\boldsymbol{x}| + \mathit{size}(f) + \mathit{size}(p) \\ \mathit{size}((t_1, t_2, \dots, t_n)) &= \mathit{size}(t_1) + \mathit{size}(t_2) + \cdots + \mathit{size}(t_n) \end{aligned}$$

The size of the expressions f, c and t_i are given by their number of function symbols and constants. Also, $\mathit{size}(X_i(\boldsymbol{x_i} : \boldsymbol{D_i}) = p_i) = |\boldsymbol{x_i}| + \mathit{size}(p_i)$. Given a specification $P = (E, I)$, $\mathit{size}(P) = \sum_{p \in E} \mathit{size}(p) + \mathit{size}(I)$.

Proposition 4.51. *Let P be a prCRL specification such that $\mathit{size}(P) = n$. Then, the worst-case time complexity of linearising P is $O(n^3)$. The worst-case size of the resulting LPPE is in $O(n^2)$.*

To get a more precise time complexity, we first define the notion of *subterms*.

Definition 4.52. *Let p be a process term, then a* subterm *of p is a process term complying to the syntax of prCRL and syntactically occurring in p. The set of all subterms of p is denoted by $\mathit{subterms}(p)$. Let $P = (E, I)$ be a specification, then $\mathit{subterms}(P)$ is the set of all process terms p such that there is an equation $X_i(\boldsymbol{x_i} : \boldsymbol{D_i}) = p_i$ in E such that $p \in \mathit{subterms}(p_i)$.*

We write $\text{subterms}'(P)$ *to denote the* multiset *containing all subterms of P (counting a process term that occurs twice as two subterms, and including nondeterministic and probabilistic choices over a vector of k variables k times).*

Now, we can define $m = |\text{subterms}'(P)|$ and

$$k = |\text{subterms}'(P)| + \sum_{(X_i(\boldsymbol{x_i}:\boldsymbol{D_i}) = p_i) \in E} |\boldsymbol{x_i}|$$

Then, it can be shown that the worst-case time complexity of linearisation is $O(m \cdot k \cdot n)$ (in the same way as we prove the above proposition).

Although the transformation to LPPE increases the size of the specification, it facilitates optimisations to reduce the state space (which is worst-case exponential), as we will see in the next chapters.

4.4 Parallel composition

Using MAPA processes as basic building blocks, we support the modular construction of large systems via top-level parallelism, encapsulation, hiding, and renaming.

Definition 4.53. *A initial process in parallel MAPA is any term that can be generated by the following grammar:*

$$q \ ::= \ Y(\boldsymbol{t}) \ \mid \ q \,||\, q \ \mid \ \partial_E(q) \ \mid \ \tau_H(q) \ \mid \ \rho_R(q)$$

Here, Y is a process name, $\boldsymbol{t}$ is a vector of data expressions, $E, H \subseteq Act$ are sets of actions, and the function $R\colon Act \setminus \{\tau\} \to Act \setminus \{\tau\}$ maps visible actions to visible actions. A parallel MAPA specification *$P = (\{X_i(\boldsymbol{x_i} : \boldsymbol{D_i}) = p_i\}, q)$ is a set of* MAPA process equations *together with an initial process q according to the above grammar*[7]*. The well-formedness criteria of Definition 4.16 are lifted in the obvious way.*

Hence, a parallel MAPA specification is just a normal MAPA specification with a more complex initial process. In such an initial process, $q_1 \,||\, q_2$ is *parallel composition*. Furthermore, $\partial_E(q)$ *encapsulates* the actions in E (i.e., omits all transitions labelled by these actions), $\tau_H(q)$ *hides* the actions in H (renaming them to the internal action τ and removing their parameters), and $\rho_R(q)$ *renames* actions according to the function R. Parallel processes by default interleave all actions. However, we assume a partial function $\gamma\colon Act \times Act \to Act$ that specifies which actions can communicate; more precisely, $\gamma(a, b) = c$ denotes that a and b can communicate if their parameters are equal, resulting in the action c with these parameters (as in ACP [BK89]).

The SOS rules for the initial process in parallel MAPA are shown in Figure 4.8 (relying on the SOS rules for MAPA from Figure 4.5), where for any probability distribution μ, we denote by $\tau_H(\mu)$ the probability distribution μ' such that $\forall p \,.\, \mu'(\tau_H(p)) = \mu(p)$. Similarly, we use $\rho_R(\mu)$ and $\partial_E(\mu)$. Also, we used $\mu \,||\, \mu'$ for the distribution μ'' such that $\mu''(p' \,||\, q') = \mu(p') \cdot \mu'(q')$ for all p', q'. In these rules, λ is a rate, a, b, c are actions and $\boldsymbol{t}$ is a vector of parameters. We use α to denote an action together with its parameters.

The intuition behind the rules is as follows:

ParL / ParR / ParL' / ParR'. The two constituent processes of a parallel composition can still act on their own. Hence, if p or q can do a transition, then so can $p \,||\, q$. For the next state, the acting process changes state as it would do normally, while the other process remains in the same state.

ParC. If a process p has an a-transition and a process q has a b-transitions, and $\gamma(a, b) = c$, then $p \,||\, q$ can synchronise on a and b to perform a c-transition. Both processes choose a next state as they would normally do. As these probabilistic decisions are independent, the probability to go to $p' \,||\, q'$ is the multiplication of the probabilities for p to go to p' and q to go to q'.

HideT / HideF / Hide'. If a process p can do a transition with an action $a(\boldsymbol{t})$, then the hiding process $\tau_H(p)$ can do the same transition, but with $a(\boldsymbol{t})$ substituted by τ if $a \in H$. The next state is chosen as before, except that it is again put within a τ_H operator. Markovian transitions cannot be hidden.

[7]Often, we abuse notation slightly by just writing the initial process when we actually mean to talk about a specification. For instance, given two processes X, Y defined by process equations $X(\boldsymbol{x_1} : \boldsymbol{D_1}) = p_1$ and $Y(\boldsymbol{x_2} : \boldsymbol{D_2}) = p_2$, we write $X(\boldsymbol{v}) \,||\, Y(\boldsymbol{v'})$ when we mean the specification $P = (\{X(\boldsymbol{x_1} : \boldsymbol{D_1}) = p_1, Y(\boldsymbol{x_2} : \boldsymbol{D_2}) = p_2\}, X(\boldsymbol{v}) \,||\, Y(\boldsymbol{v'}))$.

$$\textsc{ParL}\ \frac{p \xrightarrow{\alpha}_{\mathcal{D}} \mu}{p \,||\, q \xrightarrow{\alpha}_{\textsc{ParL}\,\mathcal{D}} \mu'} \text{ where } \forall p' \,.\, \mu'(p' \,||\, q) = \mu(p')$$

$$\textsc{ParR}\ \frac{q \xrightarrow{\alpha}_{\mathcal{D}} \mu}{p \,||\, q \xrightarrow{\alpha}_{\textsc{ParR}\,\mathcal{D}} \mu'} \text{ where } \forall q' \,.\, \mu'(p \,||\, q') = \mu(q')$$

$$\textsc{ParC}\ \frac{p \xrightarrow{a(\boldsymbol{t})}_{\mathcal{D}} \mu \qquad q \xrightarrow{b(\boldsymbol{t})}_{\mathcal{D}'} \mu'}{p \,||\, q \xrightarrow{c(\boldsymbol{t})}_{\textsc{ParC}\,\mathcal{D}\,\mathcal{D}'} \mu \,||\, \mu'} \text{ if } \gamma(a,b) = c$$

$$\textsc{HideT}\ \frac{p \xrightarrow{a(\boldsymbol{t})}_{\mathcal{D}} \mu}{\tau_H(p) \xrightarrow{\tau}_{\textsc{HideT}\,\mathcal{D}} \tau_H(\mu)} \text{ if } a \in H \qquad \textsc{HideF}\ \frac{p \xrightarrow{a(\boldsymbol{t})}_{\mathcal{D}} \mu}{\tau_H(p) \xrightarrow{a(\boldsymbol{t})}_{\textsc{HideF}\,\mathcal{D}} \tau_H(\mu)} \text{ if } a \notin H$$

$$\textsc{Ren}\ \frac{p \xrightarrow{a(\boldsymbol{t})}_{\mathcal{D}} \mu}{\rho_R(p) \xrightarrow{R(a)(\boldsymbol{t})}_{\textsc{Ren}\,\mathcal{D}} \rho_R(\mu)} \qquad \textsc{EncapF}\ \frac{p \xrightarrow{a(\boldsymbol{t})}_{\mathcal{D}} \mu}{\partial_E(p) \xrightarrow{a(\boldsymbol{t})}_{\textsc{EncapF}\,\mathcal{D}} \partial_E(\mu)} \text{ if } a \notin E$$

$$\textsc{ParL'}\ \frac{p \xrightarrow{\lambda}_{\mathcal{D}} p'}{p \,||\, q \xrightarrow{\lambda}_{\textsc{ParL'}\,\mathcal{D}} p' \,||\, q} \qquad \textsc{ParR'}\ \frac{q \xrightarrow{\lambda}_{\mathcal{D}} q'}{p \,||\, q \xrightarrow{\lambda}_{\textsc{ParR'}\,\mathcal{D}} p \,||\, q'}$$

$$\textsc{Hide'}\ \frac{p \xrightarrow{\lambda}_{\mathcal{D}} p'}{\tau_H(p) \xrightarrow{\lambda}_{\textsc{Hide'}\,\mathcal{D}} \tau_H(p')} \qquad \textsc{Ren'}\ \frac{p \xrightarrow{\lambda}_{\mathcal{D}} p'}{\rho_R(p) \xrightarrow{\lambda}_{\textsc{Ren'}\,\mathcal{D}} \rho_R(p')}$$

$$\textsc{Encap'}\ \frac{p \xrightarrow{\lambda}_{\mathcal{D}} p'}{\partial_E(p) \xrightarrow{\lambda}_{\textsc{Encap'}\,\mathcal{D}} \partial_E(p')}$$

Figure 4.8: SOS rules for parallel MAPA.

REN / REN'. If a process p can do a transition with an action $a(\boldsymbol{t})$, then the renaming process $\rho_R(p)$ can do the same transition, but with $a(\boldsymbol{t})$ substituted by $R(a)(\boldsymbol{t})$. The next state is chosen as before, except that it is again put within a ρ_R operator. Markovian transitions cannot be renamed.

ENCAPF / ENCAP'. If a process p can do a transition with an action $a(\boldsymbol{t})$, then the encapsulating process $\partial_E(p)$ can do the same transition only if $a \notin E$. The next state is chosen as before, except that it is again put within a ∂_R operator. Note that there is no ENCAP-T rule, to remove transitions labelled by an encapsulated action. Markovian transitions cannot be encapsulated; hence, they are always undisturbed by the ∂_R operator.

Note that, if $p \xrightarrow{\lambda_1} p$ and $q \xrightarrow{\lambda_2} q$, we obtain both a derivation for $p \,||\, q \xrightarrow{\lambda_1} p \,||\, q$ and one for $p \,||\, q \xrightarrow{\lambda_2} p \,||\, q$. By the operational semantics, as defined in Section 4.2.3, this means that the underlying MA has a transition $p \,||\, q \overset{\lambda}{\rightsquigarrow} p \,||\, q$ with $\lambda = \lambda_1 + \lambda_2$. The derivation-based semantics makes sure that this even works if $\lambda_1 = \lambda_2$. Hence, no side condition is needed for this scenario (as was the case for the formal definition of parallel composition of MAs in Section 3.2.3).

Example 4.54. Figure 4.9 shows the specification for a slightly more involved variant of the system explained in Example 1.1. Instead of having just one type

constant $\mathit{queueSize} = 10, \mathit{nrOfJobTypes} = 3$
type $\mathit{Stations} = \{1, 2\},\ \ \mathit{Jobs} = \{1, \dots, \mathit{nrOfJobTypes}\}$

$$\begin{aligned}
&\mathit{Station}(i : \mathit{Stations}, q : \text{Queue}\})\\
&\qquad = \mathit{size}(q) < \mathit{queueSize} \Rightarrow (2i+1) \cdot \textstyle\sum_{j:Jobs} \mathit{arrive}(j) \cdot \mathit{Station}(i, \text{enqueue}(q, j))\\
&\qquad + \mathit{size}(q) > 0 \qquad\quad \Rightarrow \mathit{deliver}(i, \text{head}(q)) \displaystyle\sum^{\bullet}_{k \in \{1,9\}} \tfrac{k}{10} : k = 1 \Rightarrow \mathit{Station}(i, q)\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\ \ + k = 9 \Rightarrow \mathit{Station}(i, \text{tail}(q))
\end{aligned}$$

$$\mathit{Server} = \textstyle\sum_{n:Stations} \sum_{j:Jobs} \mathit{poll}(n, j) \cdot (5 * j) \cdot \mathit{complete}(j) \cdot \mathit{Server}$$

$$\gamma(\mathit{poll}, \mathit{deliver}) = \mathit{copy}$$

init $\tau_{\{copy, arrive, complete\}}(\partial_{\{poll, deliver\}}(\mathit{Station}(1, \text{empty}) \mid\mid \mathit{Server} \mid\mid \mathit{Station}(2, \text{empty})))$

Figure 4.9: Specification of a polling system.

of job, as was the case there, we now allow a number of different kinds of jobs (with different service rates). Also, we allow the stations to have larger buffers.

The specification uses three data types: a set *Stations* with identifiers for the two stations, a set *Jobs* with the possible incoming jobs, and a built-in type *Queue*. The arrival rate for station i is set to $2i + 1$, so in terms of the rates in Figure 1.2 we have $\lambda_1 = 3$ and $\lambda_2 = 5$. Each job j is served with rate $5j$.

The stations receive jobs if their queue is not full, and are able to deliver jobs if their queue is not empty. As explained before, removal of jobs from the queue fails with probability $\frac{1}{10}$. The server continuously polls the stations and works on their jobs. The system is composed of the server and two stations, communicating via the *poll* and *deliver* actions. □

4.4.1 Linearisation of parallel processes

The MLPE format allows a parallel composition of processes to be linearised quite easily. Since strong bisimulation is a congruence for parallel composition (Proposition 3.27), we can first linearise the components of the parallel composition and then compose the resulting MLPEs.

Assume the following two MLPEs (omitting the initial states):

$$\begin{aligned}
X(\boldsymbol{g} : \boldsymbol{G}) &= \sum_{i \in I} \sum_{\boldsymbol{d_i} : \boldsymbol{D_i}} c_i \ \Rightarrow a_i(\boldsymbol{b_i}) \sum^{\bullet}_{\boldsymbol{e_i} : \boldsymbol{E_i}} f_i : X(\boldsymbol{n_i})\\
&+ \sum_{j \in J} \sum_{\boldsymbol{d_j} : \boldsymbol{D_j}} c_j \ \Rightarrow (\lambda_j) \cdot X(\boldsymbol{n_j})\\
Y(\boldsymbol{g'} : \boldsymbol{G'}) &= \sum_{i \in I'} \sum_{\boldsymbol{d'_i} : \boldsymbol{D'_i}} c'_i \ \Rightarrow a'_i(\boldsymbol{b'_i}) \sum^{\bullet}_{\boldsymbol{e'_i} : \boldsymbol{E'_i}} f'_i : Y(\boldsymbol{n'_i})\\
&+ \sum_{j \in J'} \sum_{\boldsymbol{d'_j} : \boldsymbol{D'_j}} c'_j \Rightarrow (\lambda'_j) \cdot Y(\boldsymbol{n'_j})
\end{aligned}$$

Also assuming (without loss of generality) that all global and local variables are named uniquely, the product $Z(\boldsymbol{g} : \boldsymbol{G}, \boldsymbol{g'} : \boldsymbol{G'}) = X(\boldsymbol{g}) \mid\mid Y(\boldsymbol{g'})$ is constructed as follows (based on the construction introduced by Usenko for traditional LPEs [Use02]):

$$\begin{aligned}
Z(\boldsymbol{g} : \boldsymbol{G}, \boldsymbol{g'} : \boldsymbol{G'}) = & \sum_{i \in I} \sum_{\boldsymbol{d_i} : \boldsymbol{D_i}} c_i \Rightarrow a_i(\boldsymbol{b_i}) \sum\nolimits^{\bullet}_{\boldsymbol{e_i} : \boldsymbol{E_i}} f_i : Z(\boldsymbol{n_i}, \boldsymbol{g'}) \\
& + \sum_{i \in I'} \sum_{\boldsymbol{d'_i} : \boldsymbol{D'_i}} c'_i \Rightarrow a'_i(\boldsymbol{b'_i}) \sum\nolimits^{\bullet}_{\boldsymbol{e'_i} : \boldsymbol{E'_i}} f'_i : Z(\boldsymbol{g}, \boldsymbol{n'_i}) \\
& + \sum_{(k,l) \in I \gamma I'} \sum_{(\boldsymbol{d_k}, \boldsymbol{d'_l}) : \boldsymbol{D_k} \times \boldsymbol{D'_l}} c_k \wedge c'_l \wedge \boldsymbol{b_k} = \boldsymbol{b'_l} \Rightarrow \\
& \qquad\qquad \gamma(a_k, a'_l)(\boldsymbol{b_k}) \sum\nolimits^{\bullet}_{(\boldsymbol{e_k}, \boldsymbol{e'_l}) : \boldsymbol{E_k} \times \boldsymbol{E'_l}} f_k \cdot f'_l : Z(\boldsymbol{n_k}, \boldsymbol{n'_l}) \\
& + \sum_{j \in J} \sum_{\boldsymbol{d_j} : \boldsymbol{D_j}} c_j \Rightarrow (\lambda_j) \cdot X(\boldsymbol{n_j}, \boldsymbol{g'}) \\
& + \sum_{j \in J'} \sum_{\boldsymbol{d'_j} : \boldsymbol{D'_j}} c'_j \Rightarrow (\lambda'_j) \cdot Y(\boldsymbol{g}, \boldsymbol{n'_j})
\end{aligned}$$

Here, $I \gamma I'$ is the set of all combinations of summands $(k, l) \in I \times I'$ such that the action a_k of summand k and the action a'_l of summand l can communicate. Formally, $I \gamma I' = \{(k, l) \in I \times I' \mid (a_k, a'_l) \in \mathrm{dom}(\gamma)\}$.

The first set of summands represents X doing a transition independent from Y, and the second set of summands represents Y doing a transition independent from X. The third set corresponds to their communications. The fourth and fifth interleave all Markovian summands of X and Y. As there is no synchronisation on Markovian transitions, no Markovian variant of the third set of summands needs to be added.

Example 4.55. Consider the following two MLPEs:

$$\begin{aligned}
& X(pc_1 : \{1,2,3\}, m : \{1,2\}) = \\
& \qquad\qquad pc_1 = 1 \Rightarrow \text{choose} \sum\nolimits^{\bullet}_{n:\{1,2\}} \tfrac{1}{2} : X(2, n) \\
& + \qquad\quad pc_1 = 2 \Rightarrow (5) \cdot X(3, m) \\
& + \sum\nolimits_{x:\{0,1\}} pc_1 = 3 \Rightarrow \text{transmit}(m + x) \sum\nolimits^{\bullet}_{l:\{*\}} 1 : X(1, m)
\end{aligned}$$

$$\begin{aligned}
& Y(pc_2 : \{1,2\}, m' : \{1,2\}) = \\
& \quad \sum\nolimits_{p:\{1,2,3\}} pc_2 = 1 \Rightarrow \text{retrieve}(p) \sum\nolimits^{\bullet}_{1:\{*\}} 1 : X(2, p) \\
& + \qquad\quad pc_2 = 2 \Rightarrow \text{send}(m') \sum\nolimits^{\bullet}_{1:\{*\}} 1 : X(1, m')
\end{aligned}$$

The first system randomly chooses either the value 1 or the value 2. Then, a delay governed by an exponential rate 5 happens. Finally, the chosen value, possibly increased by 1, is transmitted to the environment. The second system retrieves a value from the environment and then sends is out again.

The parallel composition of these two MLPEs, assuming the communication function γ given by $\gamma(\text{transmit}, \text{retrieve}) = \text{communicate}$, is

$$\begin{array}{ll}
\multicolumn{2}{l}{Z(pc_1 : \{1,2,3\}, m : \{1,2\}, pc_2 : \{1,2\}, m' : \{1,2\}) =} \\
 & pc_1 = 1 \Rightarrow \text{choose}\textstyle\sum_{n:\{1,2\}} \frac{1}{2} : Z(2, n, pc_2, m') \\
+ \sum_{x:\{0,1\}} & pc_1 = 3 \Rightarrow \text{transmit}(m + x)\textstyle\sum_{l:\{*\}} 1 : Z(1, m, pc_2, m') \\
+ \sum_{p:\{1,2,3\}} & pc_2 = 1 \Rightarrow \text{retrieve}(p)\textstyle\sum_{1:\{*\}} 1 : Z(pc_1, m, 2, p) \\
+ & pc_2 = 2 \Rightarrow \text{send}(m')\textstyle\sum_{1:\{*\}} 1 : Z(pc_1, m, 1, m') \\
+ \sum_{(x,p):\{0,1\}\times\{1,2,3\}} & pc_1 = 3 \wedge pc_2 = 1 \wedge m + x = p \Rightarrow \\
 & \text{communicate}(m + x)\textstyle\sum_{(l,l'):\{*\}\times\{*\}} 1 \cdot 1 : Z(1, m, 2, p) \\
+ & pc_1 = 2 \Rightarrow (5) \cdot Z(3, m, pc_2, m')
\end{array} \qquad \square$$

The following proposition states that our construction of an MLPE for the parallel composition of two MLPEs is correct.

Proposition 4.56. *For all $\boldsymbol{v} \in \boldsymbol{G}, \boldsymbol{v}' \in \boldsymbol{G}'$, it holds that*

$$Z(\boldsymbol{v}, \boldsymbol{v}') \approx_{\text{iso}} X(\boldsymbol{v}) \,||\, Y(\boldsymbol{v}')$$

Although the MLPE size is worst-case exponential in the number of parallel processes (when all summands have different actions and all of them communicate), in practice we see only linear growth (having only some actions communicate).

4.4.2 Linearisation of hiding, encapsulation and renaming

For hiding, renaming, and encapsulation, again we can first linearise and then manipulate the MLPE.

Remark 4.57. In our case, it is easy to see that strong bisimulation is a congruence for hiding: if each transition can be mimicked by a transition with the same label, this will still be the case after renaming some actions to τ. Maximal progress may disable some transitions due to this renaming, but at most this could make systems bisimilar that were not bisimilar before; this is no problem for congruence, though. A similar argument can be made for encapsulation and renaming (note that since it is not allowed to rename τ to a visible action, renaming can never enable transitions that were disabled by the maximal progress assumption). $\square$

For the MLPE

$$\begin{aligned}
X(\boldsymbol{g} : \boldsymbol{G}) = & \sum_{i \in I} \sum_{\boldsymbol{d_i}:\boldsymbol{D_i}} c_i \Rightarrow a_i(\boldsymbol{b_i}) \sum_{\boldsymbol{e_i}:\boldsymbol{E_i}} f_i : X(\boldsymbol{n_i}) \\
+ & \sum_{j \in J} \sum_{\boldsymbol{d_j}:\boldsymbol{D_j}} c_j \Rightarrow (\lambda_j) \cdot X(\boldsymbol{n_j})
\end{aligned}$$

we define the MLPEs $U(\boldsymbol{g})$ for $\tau_H(X(\boldsymbol{g}))$, $V(\boldsymbol{g})$ for $\rho_R(X(\boldsymbol{g}))$, and $W(\boldsymbol{g})$ for $\partial_E(X(\boldsymbol{g}))$, by

$$\begin{aligned}
U(\boldsymbol{g} : \boldsymbol{G}) &= \sum_{i \in I} \sum_{\boldsymbol{d_i} : \boldsymbol{D_i}} c_i \Rightarrow a_i'(\boldsymbol{b_i'}) \sum_{\boldsymbol{e_i} : \boldsymbol{E_i}} f_i : U(\boldsymbol{n_i}) \\
&+ \sum_{j \in J} \sum_{\boldsymbol{d_j} : \boldsymbol{D_j}} c_j \Rightarrow (\lambda_j) \cdot U(\boldsymbol{n_j}) \\
V(\boldsymbol{g} : \boldsymbol{G}) &= \sum_{i \in I} \sum_{\boldsymbol{d_i} : \boldsymbol{D_i}} c_i \Rightarrow a_i''(\boldsymbol{b_i}) \sum_{\boldsymbol{e_i} : \boldsymbol{E_i}} f_i : V(\boldsymbol{n_i}) \\
&+ \sum_{j \in J} \sum_{\boldsymbol{d_j} : \boldsymbol{D_j}} c_j \Rightarrow (\lambda_j) \cdot V(\boldsymbol{n_j}) \\
W(\boldsymbol{g} : \boldsymbol{G}) &= \sum_{i \in I'} \sum_{\boldsymbol{d_i} : \boldsymbol{D_i}} c_i \Rightarrow a_i(\boldsymbol{b_i}) \sum_{\boldsymbol{e_i} : \boldsymbol{E_i}} f_i : W(\boldsymbol{n_i}) \\
&+ \sum_{j \in J} \sum_{\boldsymbol{d_j} : \boldsymbol{D_j}} c_j \Rightarrow (\lambda_j) \cdot W(\boldsymbol{n_j})
\end{aligned}$$

where

$$a_i' = \begin{cases} \tau & \text{if } a_i \in H \\ a_i & \text{otherwise} \end{cases} \qquad a_i'' = R(a_i)$$

$$\boldsymbol{b_i'} = \begin{cases} (\,) & \text{if } a_i \in H \\ \boldsymbol{b_i} & \text{otherwise} \end{cases} \qquad I' = \{i \in I \mid a_i \notin E\}$$

Since hiding, renaming and encapsulation only affect the interactive transitions, the Markovian summands can remain unchanged. For hiding an action a, every occurrence of this action is just replaced by τ and its parameters are removed. Similarly, renaming is applied. For encapsulation of a set of actions, we just omit all interactive summands that have an action from this set.

Example 4.58. For the parallel composition of the systems X and Y of Example 4.55, we do not want the *transmit* or *retrieve* action to happen anymore; the systems are always assumed to communicate on these actions. Moreover, we are not interested in the internal behaviour, and so want to hide the *communicate* action. We can thus specify the system by means of the following initial process (choosing some initial values for the parameters of the two constituent processes):

$$\tau_{\{\text{communicate}\}}(\partial_{\{\text{transmit,retrieve}\}}(X(1,1) \,||\, Y(1,1)))$$

which is linearised to

$$\begin{aligned}
S(pc_1 : \{1,2,3\}, m : \{1,2\}, pc_2 : \{1,2\}, m' : \{1,2\}) = \\
& pc_1 = 1 \Rightarrow \text{choose} \textstyle\sum_{n:\{1,2\}} \frac{1}{2} : S(2, n, pc_2, m') \\
+ \quad & pc_2 = 2 \Rightarrow \text{send}(m') \textstyle\sum_{1:\{*\}} 1 : S(pc_1, m, 1, m') \\
+ \textstyle\sum_{(x,p):\{0,1\}\times\{1,2,3\}} & pc_1 = 3 \wedge pc_2 = 1 \wedge m + x = p \Rightarrow \\
& \tau(m+x) \textstyle\sum_{(l,l'):\{*\}\times\{*\}} 1 \cdot 1 : S(1, m, 2, p)
\end{aligned}$$

$$+ \qquad pc_1 = 2 \Rightarrow (5) \cdot S(3, m, pc_2, m') \qquad \square$$

Again, we prove our method correct.

Proposition 4.59. *For all $\boldsymbol{v} \in \boldsymbol{G}$, $U(\boldsymbol{v}) \approx_{\text{iso}} \tau_H(X(\boldsymbol{v}))$, $V(\boldsymbol{v}) \approx_{\text{iso}} \rho_R(X(\boldsymbol{v}))$, and $W(\boldsymbol{v}) \approx_{\text{iso}} \partial_E(X(\boldsymbol{v}))$.*

4.5 Basic reduction techniques

Using the MLPE, several reduction techniques can now easily be applied to MAPA specifications. Even simpler, some reductions can be defined on the LPPE (i.e., an MLPE without any Markovian summands) in case they preserve derivations, due to Theorem 4.36.

We introduce three basic reduction techniques: (1) *Constant elimination* detects parameters of an MLPE that never change value. Then, these parameters are omitted and all references to them are replaced by their initial values. (2) *Expression simplification* evaluates functions for which all parameters are constants and applies basic laws from logic. (3) *Summation elimination* aims to remove unnecessary summations that often arise from communications. These techniques do not change the actual state space, but simplify the specification and hence speed up state space generation. All our techniques work on the syntactic level, i.e., they do not unfold the data types at all, or only locally to avoid a data explosion—this makes them easy to apply.

We note that not all ideas presented in this section are original; these simplification techniques already exist for the non-quantitative LPE format [GL01]. However, their generalisation to the Markovian setting is new. We would also like to stress again that, since MAs generalise LTSs, CTMCs, DTMCs, PAs and IMCs, all our reduction techniques are also applicable to these subclasses.

4.5.1 Constant elimination

If a parameter of an MLPE never changes value, we can clearly just omit it and replace every reference to it by its initial value. Basically, we detect a parameter p to be constant if in every summand it is either unchanged, or 'changed' to its initial value.

More precisely, we do a greatest fixed-point computation to find all non-constants, initially assuming all parameters to be constant. If no new non-constants are found (which happens after a finite number of iterations as there are finitely many parameters), the procedure terminates and the remaining parameters are constant. In every iteration, we check for each parameter x that is still assumed constant whether there exists a summand s (with an enabling condition that cannot be shown to be always unsatisfied) that may change it. This is the case if either x is bound by a probabilistic or nondeterministic summation in s, or if its next state is determined by an expression that is syntactically different from x, different from the initial value of x, and different from the name of another parameter that is still assumed constant and has the same initial value as x.

Example 4.60. Consider the specification

$$P = (\{X(id : \{\text{one}, \text{two}\}) = \text{say}(id) \cdot X(id)\}, X(\text{one}))$$

Clearly, the process variable *id* never changes value, so the specification can be simplified to $P' = (\{X = \text{say}(\text{one}) \cdot X\}, X)$. □

Proposition 4.61. *The underlying MAs of an MLPE before and after constant elimination are isomorphic.*

Actually, we initially implemented constant elimination for prCRL by having it operate on the LPPE. It is easy to see that constant elimination is derivation preserving. Hence, by Theorem 4.36 we can just use this implementation for optimising MAPA specifications as well: first encode as prCRL, then linearise to an LPPE, then apply constant elimination, and then decode back to an MLPE (as depicted earlier in Figure 4.2).

Since every iteration goes through the MLPE once, and the number of iterations is limited by the number of parameters m, the worst-case time-complexity is $O(m \cdot N)$ (with N the size of the MLPE).

4.5.2 Expression simplification

Expressions occurring as enabling conditions, action parameters or next state parameters can often be simplified. We apply two kinds of simplifications (recursively): (1) functions for which all parameters are constants are evaluated, and (2) basic laws from logic are applied.

Additionally, summands for which the enabling condition simplified to *false* are removed, as they cannot contribute to the behaviour of an LPPE anyway. More thoroughly, we also check for each summand whether every local and global parameter with a finite type has at least one possible value for which the enabling condition does not simplify to *false*. If there exists a parameter without at least one such a value, the summand apparently can never be taken and hence is removed.

Example 4.62. Consider the expression $3 = 1+2 \vee x > 5$. As all parameters of the addition function are given, the expression is first simplified to $3 = 3 \vee x > 5$. Then, the equality function can be evaluated, obtaining $\mathit{true} \vee x > 5$. Finally, logic tells us that we can simplify once more, obtaining the expression *true*. □

The following proposition is trivial: replacing expressions by equivalent ones does not change anything.

Proposition 4.63. *The underlying MAs of an MLPE before and after expression simplification are isomorphic.*

Again, we actually employ an implementation on LPPEs via the encoding discussed earlier. After all, it is immediately clear that expression simplification does not influence a process's derivations.

Expression simplification is linear in the size of the MLPE, as it just processes all expressions one by one to simplify them.

4.5.3 Summation elimination

When composing parallel components that communicate based on message passing actions, often this results in summations that can be eliminated. For instance, summands with a nondeterministic choice of the form $\sum_{d:D}$ and a condition $d = e$ may arise, which can obviously be simplified by omitting the summation and substituting e for every occurrence of d.

More precisely, to eliminate a sum $\sum_{d:D}$ in a summand that has the enabling condition c, we compute the set S of possible values that c allows for d (and use the empty set if we cannot establish specific values). When c is given by $d = e$ or $e = d$, where e is a value or an expression (in which d does not occur freely) that can be evaluated, we take S to be the singleton set containing this value. When c is a conjunction $e_1 \wedge e_2$, we take $S = S_1 \cap S_2$, where S_i is the set of possible values for c given by e_i. For a disjunction, we take a union (unless $S_1 = \varnothing$ or $S_2 = \varnothing$; in that case also $S = \varnothing$). If it turns out that S is a singleton set $\{d'\}$, we omit the summation and substitute every free occurrence of d by d'.

Example 4.64. Consider the specification $X = \sum_{d:\{1,2,3\}} d = 2 \Rightarrow \text{send}(d) \cdot X$. As the summand is only enabled for $d = 2$, the specification can be simplified to $X = 2 = 2 \Rightarrow \text{send}(2) \cdot X$. Expression simplification may further reduce this to $X = \text{send}(2) \cdot X$. □

The concept of summation elimination described above, as well as the notions of constant elimination and expression simplification, could be generalised to the Markovian setting without any trouble. They all could be implemented for LPPEs and applied to MAPA by means of our encoding. Their procedures as described here are basically no different from the way they were defined earlier for the non-quantitative LPEs [GL01].

For the second part of summation elimination, which we define next, this is not the case anymore: we have to be slightly more careful. In the context of LPEs and LPPEs we reduced summands such as $\sum_{d:\{1,2\}} a \cdot X$ to $a \cdot X$, as the summation variable is not used anyway. This, however, does influence the number of derivations to take the action a. Hence, if this were an encoded rate, the reduction would be erroneous. Therefore, we have to define summation elimination directly on MLPEs. Interactive summands can be handled as before, but for Markovian summands the second kind of reduction is altered. Instead of reducing $\sum_{d:D} (\lambda) \cdot X$ to $(\lambda) \cdot X$, we now reduce to $(|D| \times \lambda) \cdot X$. That way, the total rate to X remains the same.

In fact, we can even reduce summations over a variable that also appears in the condition and the rate of a Markovian summand, as long as it does not occur in the probabilities and next state expressions and there are no other variables that occur in the condition. For instance, we automatically reduce $\sum_{d:\{2..5\}} d < 4 \Rightarrow (d^i) \cdot X$ to $(2^i + 3^i) \cdot X$. To be precise, we reduce

$$\sum_{d:D} c \Rightarrow (\lambda) \cdot X \quad \text{to} \quad \Big(\sum_{\substack{d' \in D \\ c[d := d']}} \lambda[d := d'] \Big) \cdot X$$

where the second summation is meant in the mathematical sense. If the summation variable d does not occur in the condition, we can just sum over all its values and leave the condition intact.

Proposition 4.65. *The underlying MAs of an MLPE before and after summation elimination are isomorphic.*

Summation elimination is linear in the size of the MLPE.

4.6 Contributions

We introduced a new process-algebraic framework—called MAPA (Markov Automata Process Algebra)—for modelling and generating MAs. It can be used to specify systems incorporating both nondeterminism, probability and Markovian rates. Key ingredient is the combined treatment of data and data-dependent probabilistic choice in a fully symbolic manner. Since MAs generalise LTSs, DTMCs, CTMCs, IMCs and PAs, the MAPA framework is applicable to specifying systems in any of these domains.

We defined a restricted format of MAPA, the MLPE, that allows easy state space generation and parallel composition. Also, its basic structure enables much simpler definitions of many reduction techniques, as will become apparent in the next three chapters. We showed how MAPA specifications can be encoded in a restriction of itself, prCRL, and defined the novel concept of derivation-preserving bisimulation to prove under which circumstances MAPA transformations can safely—that is, while preserving strong bisimulation—be defined solely on prCRL.

We showed how to linearise prCRL specifications to a restricted variant of the MLPE, while indeed obtaining derivation-preserving bisimilar results. Hence, the encoding scheme allows this procedure to be lifted to transform MAPA specifications to MLPEs without any effort. That way, techniques defined on the MLPE can be automatically applied to the full spectrum of MAPA.

The results show that the treatment of probabilities and Markovian rates is simple and elegant, and rather orthogonal to the traditional setting [Use02]. This is very desirable, as it simplifies the generalisation of existing techniques to the Markovian setting. We did this for three existing reduction techniques: constant elimination, expression simplification and summation elimination. The case studies in Chapter 9 will demonstrate their significance, sometimes speeding up state space generation by more than an order of magnitude.

Although the CADP [CGH$^+$10, GLMS13] and MODEST [BDHK06] frameworks are also able to model rates and probabilistic choices in the presence of data, the MAPA framework is much simpler. None of the other quantitative approaches features a restricted format such as the MLPE, which allows us to easily define reduction techniques that are not available in the other toolsets. CADP does implement bisimulation minimisation, but not for systems that feature both discrete probabilistic choice and stochastic timing—hence, it is not applicable to MAs. As already discussed in Section 1.3 and in the introduction of this chapter, all other related specification languages cannot handle the full spectrum of MAs and/or are not able to deal with data.

CHAPTER 5

Dead Variable Reduction

"Everything should be made as simple as possible, but no simpler."

Albert Einstein

SECTION 4.5 introduced three techniques for MAPA that simplify the MLPE representation of specifications, without influencing the state space. The current chapter provides a different type of technique: dead variable reduction. This technique does not necessarily simplify MLPEs, but rather reduces the number of states of the underlying MA while preserving strong bisimulation. We define this reduction technique on LPPEs, show that it yields derivation-preserving bisimilar systems, and apply Theorem 4.36 to show that it can be used on MLPEs just as well. The linearisation procedure introduced in Section 4.3 makes the technique applicable to all MAPA specifications. The idea of our technique is to analyse when certain global variables of an LPPE are *dead*, also called *irrelevant*. Basically, this means that for all continuations of the process, these variables are overwritten before they are used again. Hence, they can just as well be reset to their initial value, reducing the number of states.

Our approach. Due to the structure of an LPPE, it is not obvious when variables are irrelevant; this relies on the order in which summands can be executed. The explicit control flows of the original parallel processes have been lost, though, since they were merged into one linear form. Therefore, it is not immediately clear which values a variable may get in the future, or when it will be overwritten. Moreover, some control flow could already have been encoded in the state parameters of the original specification. To solve this problem, we first present a technique to (re)construct the *control flow graphs* of an LPPE. This technique is based on detecting which state parameters act as program counters for the underlying parallel processes; we call these counters *control flow parameters* (CFPs). We analyse which summands of the LPPE correspond to each CFP, similar to the ideas of *program slicing* [Wei84].

Using the reconstructed control flow, we define a parameter to be *relevant* in a given state if, before being overwritten, it may be used by an enabling condition, action parameter or probability distribution, or by an expression to determine the value of another parameter that will be relevant in the next state. Based on this notion of relevance, we present a syntactic reduction technique that resets variables that are *not* relevant to their initial value. This is justified, because these variables will anyway be overwritten before ever being read. We show that our technique may only shrink the state space and that it never enlarges it, and discuss a couple of our technical design choices. A case study on a handshake register shows its applicability in practice.

Related work. Liveness analysis techniques are well-known in compiler theory [ASU86]. However, their focus is often not on handling the multiple control flows arising from parallelism. Moreover, these techniques generally work only locally for each block of program code, and aim at reducing execution time instead of state space size.

The concept of resetting dead variables for state space reduction was first formalised by Bozga et al. [BFG99, FBG03], but in their analysis relevance of variables was only dealt with locally, such that a variable that is passed to a queue or written to another variable was considered relevant, even if it is never used afterwards. A similar technique was presented in [YG04], using analysis of control flow graphs. It suffers from the same locality restriction as [BFG99]. Most recent is [GS06], which applies data flow analysis to value-passing process algebras. It uses Petri nets as its intermediate format, featuring concurrency and taking into account global liveness information. We improve on this work by providing a thorough formal foundation including bisimulation preservation proofs, and by showing that our transformation never increases the state space. Additionally, none of the aforementioned techniques works in the context of MAs. Most importantly, they do not attempt to reconstruct control flow information that is hidden in the state variables, missing opportunities for reduction—they can only reduce based on control flow resulting from the structure of the parallel components, whereas we can exploit user-induced control flow as well.

The μCRL toolkit already contained a tool `parelm`, implementing a basic variant of our methods. Instead of resetting state parameters that are dead given some context, it simply removes parameters that are dead in all contexts [GL01]. That is, it marks all parameters that either occur in some condition or action argument, and also (recursively and iteratively) all parameters that are used in some summand to determine one of the marked parameters. Unmarked parameters are then deleted. As it does not take into account the control flow, parameters that are sometimes relevant and sometimes not will never be reset.

Organisation of the chapter. We discuss the construction of control flow graphs in Section 5.1, the data flow analysis in Section 5.2, and the LPPE transformation in Section 5.3. Some potential adaptations to the framework, including examples illustrating our choice for the current variant, are discussed in Section 5.4. In Section 5.5 we demonstrate our technique on a larger example. Finally, Section 5.6 concludes by summarising the contributions of this chapter.

Origins of the chapter. The ideas of dead variable reduction for non-probabilistic LPEs were developed by Jaco van de Pol, and published together with the author in the proceedings of the *7th International Symposium on Automated Technology* *for Verification and Analysis* (ATVA) [vdPT09a] and a corresponding technical report [vdPT09b]. For this chapter, the author generalised the framework to work with LPPEs, and added correctness proofs for derivation preservation to be able to apply the technique to MAPA specifications as well.

5.1 Reconstructing the control flow graphs

The techniques introduced in this chapter all work on LPPEs. As discussed in Chapter 4, these are MLPEs without any Markovian summands (and hence they are also part of the prCRL language):

Definition 4.37 (LPPEs). *A* linear probabilistic process equation *(LPPE) is a MAPA specification of the following format:*

$$X(\boldsymbol{g} : \boldsymbol{G}) = \sum_{i \in I} \sum_{\boldsymbol{d_i} : \boldsymbol{D_i}} c_i \Rightarrow a_i(\boldsymbol{b_i}) \sum_{\boldsymbol{e_i} : \boldsymbol{E_i}} f_i : X(\boldsymbol{n_i})$$

Throughout this chapter we assume the existence of such an LPPE, with variable names as indicated and an initial state vector $\boldsymbol{init}$, and we assume that it is decodable (see Remark 4.31). We use the indices $i \in I$ to refer to its summands. Without loss of generality, we assume that all variables in $\boldsymbol{g}$ (the *parameters* or *global variables*), $\boldsymbol{d_i}$ (the *local variables*) and $\boldsymbol{e_i}$ (the *probabilistic variables*) are unique—this can easily be achieved by renaming duplicates.

Given a vector of formal state parameters $\boldsymbol{g}$, we use g_j to refer to its j^{th} parameter. An actual state is a vector of values, which we often denote by $\boldsymbol{v}$; we use v_j to refer to its j^{th} value. We use G_j to denote the type of g_j, and J for the set of all parameters g_j. Furthermore, $n_{i,j}$ denotes the j^{th} element of $\boldsymbol{n_i}$, and pars(t) the set of all parameters g_j that syntactically occur in the expression t.

5.1.1 Basic control flow analysis

To construct the control flow of an LPPE, we detect which of its global parameters are acting as program counters for the system or a part of it. We take this to be the case for a parameter $g_j \in J$ if each summand either leaves it unchanged (since that summand deals with a part of the system not governed by g_j) or changes it from a specific value to another specific value (since that summand does deal with a part of the system governed by g_j). Of course, we could just remember which parameters were introduced as program counters during the linearisation procedure of Section 4.3. However, the approach we take here is more general: it does not only detect these parameters, but may additionally detect user-defined parameters to also act as program counters for (a part of) the system.

First, we define a parameter to be *changed* in a summand i if its value after taking i may be different from its current value. Clearly, this can only be the case if the expression $n_{i,j}$ is different from g_j.

Definition 5.1 (Changed parameters). *Given a summand i, a parameter $g_j \in J$ is* changed *in i if $n_{i,j} \neq g_j$, otherwise it is* unchanged *in i.*

Note that parameters that are changed according to this definition may still keep the same value. However, since this is not easily decidable in practice, we chose to take a safe overapproximation. Parameters that are unchanged in a summand i according to this definition indeed will never be changed by i.

For a summand i to change a parameter g_j from one specific value to another, there should be one value s such that the enabling condition of i can only hold if g_j has that value. We call this value the *source* of g_j for i.

Definition 5.2 (Source functions). *A function $f \colon I \times J \to \bigcup_{j \in J} G_j \cup \{\bot\}$ is a* source function *if, for every $i \in I$, $g_j \in J$, and $s \in G_j$, $f(i, g_j) = s$ implies that*

$$\forall \boldsymbol{v} \in \boldsymbol{G}, \boldsymbol{d'_i} \in \boldsymbol{D_i} \,.\, c_i(\boldsymbol{v}, \boldsymbol{d'_i}) \implies v_j = s$$

Furthermore, $f(i, g_j) = \bot$ is always allowed; it indicates that no unique value s complying to the above could be found.

From now on we assume a given source function source.

We note that $\mathit{source}(i, g_j)$ is allowed to be $\bot$ even though there may be some source s, as computing the source is in general undecidable—so, in practice heuristics are used that sometimes yield $\bot$ when in fact a source is present. However, we will see that this does not result in any errors. It may limit the effects of the reduction, though. The same holds for the destination functions defined below.

The heuristics we apply in our implementation (see Chapter 9) to find a source can handle equations, disjunctions and conjunctions, as well as negations of boolean expressions. For an equational condition $g_j = c$ the source of g_j is obviously c, for a disjunction of such terms we apply set union, and for conjunction intersection. If for some summand i a *set* of sources is obtained, it can be split into multiple summands, such that each again has a unique source.

Example 5.3. Let c_i be given by $(g_j = 3 \lor g_j = 5) \land g_j = 3 \land g_k = 10$, then obviously $\mathit{source}(i, g_j) = 3$ is valid (because $(\{3\} \cup \{5\}) \cap \{3\} = \{3\}$), but also (as always) $\mathit{source}(i, g_j) = \bot$ is allowed. $\square$

For a summand to change a parameter from one specific value to another, in addition to having a unique source it also should have a unique *destination.* We define the destination of a parameter g_j for a summand i to be the unique value g_j obtains after taking i, regardless of the values of the global, local and probabilistic variables. Again, we only specify a minimal requirement, since such a value may not always easily be found.

Definition 5.4 (Destination functions). *A function $f \colon I \times J \to \bigcup_{g_j \in J} G_j \cup \{\bot\}$ is a* destination function *if, for every $i \in I$, $g_j \in J$, and $s \in G_j$, $f(i, g_j) = s$ implies that*

$$\forall \boldsymbol{v} \in \boldsymbol{G}, \boldsymbol{d'_i} \in \boldsymbol{D_i}, \boldsymbol{e'_i} \in \boldsymbol{E_i} \,.\, c_i(\boldsymbol{v}, \boldsymbol{d'_i}) \implies n_{i,j}(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e'_i}) = s$$

Furthermore, $f(i, g_j) = \bot$ is always allowed, indicating that no unique destination value could be derived.

From now on we assume a given destination function dest.[1]

The heuristics in our implementation for computing $\mathit{dest}(i, g_j)$ substitute $\mathit{source}(i, g_j)$ for g_j in $n_{i,j}$, and try to rewrite the result to a closed term.

Example 5.5. If c_i is given by $g_j = 8$ and $n_{i,j}$ by $g_j + 5$, then $\mathit{dest}(i, g_j) = 13$ is valid, but also (as always) $\mathit{dest}(i, g_j) = \perp$ is allowed. If for instance c_i is $g_j = 5$ and $n_{i,j}$ equals e_3, then $\mathit{dest}(i, g_j)$ can only yield $\perp$, since the value of g_j after taking i is not fixed (it depends on the value of e_3). □

We say that a parameter g_j *rules* a summand i if both its source and its destination for that summand can be computed, i.e., are different from $\perp$. This implies that i indeed changes g_j from one specific value to another, and hence that i belongs to the part of the system governed by g_j.

Definition 5.6 (The rules relation). *A parameter $g_j \in J$* rules *a summand i if*

$$\mathit{source}(i, g_j) \neq \perp \quad \text{and} \quad \mathit{dest}(i, g_j) \neq \perp$$

The set of all summands ruled by g_j is denoted by $R_{g_j} = \{i \in I \mid g_j \text{ rules } i\}$. Furthermore, V_{g_j} denotes the set of all possible values that g_j can take before and after taking one of the summands that it rules, plus its initial value. Formally,

$$V_{g_j} = \{\mathit{source}(i, g_j) \mid i \in R_{g_j}\} \cup \{\mathit{dest}(i, g_j) \mid i \in R_{g_j}\} \cup \{\mathit{init}_j\}$$

Examples will show that summands can be ruled by several parameters.

5.1.2 Control flow parameters

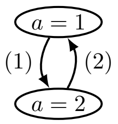
We now define a parameter to be a *control flow parameter* if it rules all summands in which it is changed. Stated differently, in every summand a control flow parameter is either left alone or we know what happens to it. As discussed before, such a parameter can be seen as a *program counter* for the summands it rules, and therefore its values can be seen as *locations.* All other parameters are called *data parameters*.

Definition 5.7 (Control flow and data parameters). *A parameter $g_j \in J$ is a* control flow parameter (CFP) *if for all $i \in I$, either g_j rules i or g_j is unchanged in i. A parameter that is not a CFP is called a* data parameter (DP).

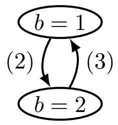
We call the set of all summands $i \in I$ ruled by g_j its cluster *(denoted above by R_{g_j}). The set of all CFPs is denoted by $\mathcal{C}$, the set of all DPs by $\mathcal{D}$.*

Example 5.8. Consider the following LPPE, based on the specification in Example 4.25. It still consists of two buffers that continuously read and write

[1]Note that we could be a bit more liberal than the current definition, only requiring $n_{i,j}(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e'_i}) = s$ for values $\boldsymbol{e'_i} \in \boldsymbol{E_i}$ such that $f_i(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e'_i}) > 0$. We chose not to do so, to keep the presentation simple. However, clearly all our results still hold for this more liberal definition.



(a) Control flow graph for a.



(b) Control flow graph for b.

Figure 5.1: Control flow graphs for the LPPE of Example 5.8.

messages from and to the environment, but now also contains some probabilistic erroneous behaviour. It is used as a running example throughout this chapter.

$$
\begin{array}{llll}
X(a : \{1,2\}, b : \{1,2\}, x : \mathbb{N}, y : \mathbb{N}) = & & & \\
\quad \sum_{d:\mathbb{N}} & a = 1 & \Rightarrow \mathrm{read}(d) \sum_{c:\{0,1\}} \frac{1}{2} : X(2, b, d + c, y) & (1) \\
\quad + & a = 2 \wedge b = 1 & \Rightarrow \mathrm{comm}(x) \cdot X(1, 2, x, x) & (2) \\
\quad + & b = 2 & \Rightarrow \mathrm{write}(y) \cdot X(a, 1, x, y) & (3)
\end{array}
$$

For the first summand we may define $\mathit{source}(1, a) = 1$, since this summand is clearly only enabled if $a = 1$. Also, we can define $\mathit{dest}(1, a) = 2$, since $n_{1,1} = 2$ for all valuations of the global, local and probabilistic variables. Therefore, parameter a rules the first summand. Similarly, it rules the second summand. As a is unchanged in the third summand, it is a CFP (with summands 1 and 2 in its cluster). In the same way, we can show that parameter b is a CFP ruling summands 2 and 3. Parameter x is a DP, as it is changed in summand 1 while both its source and its destination are not unique. From summand 2 it follows that y is a DP. □

Based on CFPs, we can define *control flow graphs*. The nodes of the control flow graph of a CFP g_j are the values g_j can take, and the edges denote possible transitions between these values. Specifically, an edge labelled i from value s to t denotes that summand i may be taken if g_j has the value s, resulting in it getting the value t.

Definition 5.9 (Control flow graphs). *Let g_j be a CFP, then the* control flow graph *for g_j is the tuple (V_{g_j}, E_{g_j}), where V_{g_j} is as given in Definition 5.6, and*

$$E_{g_j} = \{(s, i, t) \mid i \in R_{g_j} \wedge s = \mathit{source}(i, g_j) \wedge t = \mathit{dest}(i, g_j)\}$$

Example 5.10. Figure 5.1 shows the control flow graphs for the LPPE of Example 5.8. □

Note that we construct a local control flow graph per CFP, rather than a global control flow graph. Although global control flow may be useful, its graph could grow larger than the complete state space, completely defeating its purpose.

Algorithm 5: Reachable values of a CFP g_j.

```
$Reach_j = \{init_j\}$;
$Prev = \varnothing$;
while $Reach_j \neq Prev$ do
    $Explore := Reach_j \setminus Prev$;
    $Prev := Reach_j$;
    forall the $s \in Explore$ do
        forall the $i \in R_{g_j}$ such that $s = source(i, g_j)$ do
            $Reach_j := Reach_j \cup \{dest(i, g_j)\}$;
        end
    end
end
```

5.1.3 Removing dead code using CFPs

Although our main purpose of control flow analysis is to reset dead variables, it has an additional application: removal of unreachable summands. After all, for each CFP g_j we can easily compute the set of all its reachable values $Reach_j$ using Algorithm 5. The algorithm provides an overapproximation, since conditions containing other parameters may prevent some summands from being executed.

Proposition 5.11. *After executing Algorithm 5, the set $Reach_j$ contains all values that a CFP g_j may obtain.*

Now, it immediately follows that a summand $i \in I$ can be removed if, for some CFP g_j that rules i, $source(i, g_j) \notin Reach_j$. After all, in this case the enabling condition of i will never be satisfied. From the operational semantics it then follows that i does not contribute to the behaviour of the system. Although removal of such summands does not reduce the state space, it does clean up the LPPE and hence may speed up state space generation. This extension has not been included in our implementation yet, though.

Note that unreachable summands may arise naturally if behaviour can be disabled by the context of a process.

5.2 Simultaneous data flow analysis

Using the notion of CFPs, we analyse to which CFPs each DP *belongs*. We say that a DP g_k belongs to a CFP g_j if g_j rules all summands that change or use g_k. After all, in that case we can say that g_k belongs to the part of the system governed by the summands in the cluster of g_j. Therefore, all decisions on resetting g_k can be made based on these summands.

First, we formalise what we mean by a parameter being used by a summand. This is split into two parts: a parameter is *directly used* in i if it occurs in its enabling condition c_i, action parameters $\boldsymbol{b_i}$ or probability expression f_i, and *used* if it is either directly used or needed to compute the next state. The idea is that *directly used* parameters are immediately needed for the summand to behave as it should. The relevance of parameters that are just *used* depends on whether or not the next-state parameters they influence will be of use in the next state.

Definition 5.12 (Used parameters). *A parameter $g_j \in J$ is* directly used *in a summand i if $g_j \in \text{pars}(c_i) \cup \text{pars}(\boldsymbol{b_i}) \cup \text{pars}(f_i)$, and* used *in i if it is directly used in i or $g_j \in \text{pars}(n_{i,k})$ for some k such that g_k is changed in i.*

Note that the requirement for g_k to be changed in i just means that $k \neq j$ or that $k = j$ and $\text{pars}(n_{i,j}) \neq g_j$.

Definition 5.13 (The belongs-to relation). *Let g_k be a DP and g_j a CFP, then g_k* belongs to *g_j if all summands $i \in I$ that use or change g_k are ruled by g_j. For technical reasons, we assume that each DP belongs to at least one CFP, and we do not allow CFPs to belong to anything.*

The assumption of each DP belonging to at least one CFP can always be satisfied by adding a fresh global variable of some dummy type $\{*\}$ to the LPPE and leaving it unused and unchanged in all summands—trivially, it is a CFP and all DPs belong to it. Clearly, this adapted LPPE has the same behaviour as the original system.

Example 5.14. In our running example, x belongs to a and y belongs to b. □

5.2.1 Data relevance analysis

Having settled the control flow and knowing to which CFPs each DP belongs, we continue our analysis. For each DP g_k and each CFP g_j such that g_k belongs to g_j, we check during which part of g_j's cluster the value of g_k is irrelevant.

Basically, g_k is irrelevant at a certain point if its value will always be overwritten before being needed again; otherwise, it is relevant. More precisely, the relevance of g_k is divided into three conditions, stating whether g_k is relevant if a CFP g_j that it belongs to has some value s (denoted by $R(g_k, g_j, s)$). We illustrate all three conditions based on examples.

Direct usage. Consider the following LPPE.

$$\begin{aligned} &X(a : \{1,2\}, x : \mathbb{N}) = \\ &\qquad a = 1 \Rightarrow \text{send}(x) \cdot X(1, x+1) \end{aligned}$$

It is easy to see that a is a CFP and that x belongs to a. Note that the value of x is relevant if a has value 1, since that enables a summand that directly uses x. Hence, we write $R(x, a, 1)$. This motivates our first condition of relevance:

1. If a DP $g_k \in \mathcal{D}$ is *directly used* in some $i \in I$, then for every $g_j \in \mathcal{C}$ to which g_k belongs we require

$$R(g_k, g_j, \mathit{source}(i, g_j))$$

Note that indeed $source(i, g_j) \neq \bot$, since the facts that g_k is directly used in i and that g_k belongs to g_j together imply that g_j rules i and hence that $source(i, g_j) \neq \bot$.

Indirect usage within one cluster. Now consider the following LPPE.

$$\begin{aligned}
&X(a : \{1, 2\}, x : \mathbb{N}, y : \mathbb{N}) = \\
&\qquad a = 1 \Rightarrow \tau \cdot X(2, y, y + 1) && (1)\\
&\qquad a = 2 \Rightarrow \mathrm{send}(x) \cdot X(1, x + 1, y) && (2)
\end{aligned}$$

Clearly a is a CFP, and x and y both belong to a. By the previous condition, we immediately find $R(x, a, 2)$ (due to the second summand). Additionally, we observe that $R(y, a, 1)$ should hold, since y is used in the first summand (enabled when a has value 1) to determine the value that x will have when a becomes 2. This motivates our second condition of relevance:

2. If a DP $g_k \in \mathcal{D}$ is *used* in some $i \in I$ to determine the value of another DP $g_l \in \mathcal{D}$, then for every $g_j \in \mathcal{C}$ such that g_k belongs to g_j and $R(g_l, g_j, dest(i, g_j))$ we require

$$R(g_k, g_j, source(i, g_j))$$

Again, $source(i, g_j)$ and $dest(i, g_j)$ are well-defined since g_k is used in i and belongs to g_j, and hence g_j has to rule i.

Indirect usage between clusters. While the previous condition takes into account the use of DPs to set other DPs that will be relevant later within the same cluster, data may also be copied between different clusters. Consider for instance the following LPPE.

$$\begin{aligned}
&X(a : \{1, 2\}, b : \{1, 2\}, x : \mathbb{N}, y : \mathbb{N}) = \\
&\qquad a = 1 &&\Rightarrow \mathrm{send}(x) \cdot X(2, b, x, y) && (1)\\
&\qquad a = 2 \wedge b = 1 &&\Rightarrow \tau \cdot X(1, 2, y, y) && (2)\\
&\qquad b = 2 &&\Rightarrow \tau \cdot X(a, 1, x, y + 1) && (3)
\end{aligned}$$

We find that a and b are CFPs, that x belongs to a and that y belongs to b. Clearly, $R(x, a, 1)$ due to the first summand. Moreover, we observe that y is used in the second summand to determine the value of x. Since this summand sets a to 1, the value of x may be relevant in the next state (due to $R(x, a, 1)$), and hence y may be relevant in the second summand (so $R(y, b, 1)$). However, there is no $g_j \in \mathcal{C}$ such that y belongs to g_j and $R(x, g_j, dest(i, g_j))$, so the second condition is not applicable. Hence, we need the following third condition, dealing with a situation like this when a value is copied between clusters of two CFPs:

3. If a DP $g_k \in \mathcal{D}$ is used in some $i \in I$ to determine the value of another DP $g_l \in \mathcal{D}$ such that $R(g_l, g_p, dest(i, g_p))$ for some $g_p \in \mathcal{C}$, then for every

$g_j \in \mathcal{C}$ such that g_k belongs to g_j and g_l does *not* belong to g_j we require

$$R(g_k, g_j, \mathit{source}(i, g_j))$$

Now, for the example LPPE we already knew that $R(x, a, 1)$, that y belongs to b and that x does not belong to b. Hence, we obtain $R(y, b, 1)$ via this condition.

By adding the restriction that g_l does *not* belong to g_j, we exclude the situation that $R(g_l, g_p, \mathit{dest}(i, g_p))$ for some $g_p \in \mathcal{C}$ and that both g_k and g_l belong to g_j. This restriction is indeed safe, because if both g_k and g_l belong to g_j, all of their behaviour happens within the cluster of g_j. Hence, we don't have to consider the cluster of any other CFP g_p to decide for which values of g_j the value of g_k is relevant for setting the value of g_l. Our second condition suffices to find all the relevant source values for g_j. The following example shows that we may even unnecessarily denote DPs to be relevant if we lift the restriction.

Example 5.15. Consider the following updated variant of our third condition.

(3') If a DP $g_k \in \mathcal{D}$ is used in some $i \in I$ to determine the value of another DP $g_l \in \mathcal{D}$ such that $R(g_l, g_p, \mathit{dest}(i, g_p))$ for some $g_p \in \mathcal{C}$, then

$$R(g_k, g_j, \mathit{source}(i, g_j))$$

for every $g_j \in \mathcal{C}$ such that g_k belongs to g_j.

Note that this updated condition subsumes our second condition, since it includes the case that $g_p = g_j$. As the adapted definition is more inclined to consider a DP relevant, the lemma relying on this definition (Lemma A.20) will still hold. However, this potentially yields more relevance than necessary, decreasing the number of reductions that can be made. As an example, observe the following LPPE.

$$\begin{array}{lll}
X(p\colon \{1,2\}, q\colon \{1,2\}, x\colon \mathbb{N}) = & & \\
\quad\;\; p = 1 \wedge q = 1 \Rightarrow a(x) \cdot X(2,1,x) & & (1) \\
\quad + \;\; p = 2 \wedge q = 1 \Rightarrow \tau \cdot X(1,1,2) & & (2) \\
\quad + \;\; p = 2 \wedge q = 2 \Rightarrow \tau \cdot X(2,1,x) & & (3)
\end{array}$$

Clearly, p and q are CFPs and x belongs to both p and q. Using our initial three condition, we find $R(x, p, 1)$ and $R(x, q, 1)$ when applying the first condition on the first summand of X. The second condition then yields $R(x, q, 2)$ by looking at the third summand. Now, no clause applies anymore, resulting in the observation that $\neg R(x, p, 2)$, and hence—as we will discuss in more detail later—x can be reset in the next-state expression of the first and third summand.

Using condition (3') instead of (3), on the other hand, the third summand combined with $R(x, q, 1)$ *would* yield $R(x, p, 2)$, so no reductions can be made anymore. The problem here is that x seems to be relevant when $p = 2$, since in that case the third summand can be taken, after which $q = 1$ (and we already knew that $R(x, q, 1)$). However, after taking the third summand p will still be 2, preventing the first summand from being taken. □

We are now able to give a smallest fixed-point definition of the relevance relation R, using three clauses based on the three conditions discussed above. We phrase the clauses slightly differently to accommodate the fixed-point way of defining R, but it can easily be seen that they precisely correspond to the three conditions described above.

Definition 5.16 (Relevance). *Let $g_k \in \mathcal{D}$ and $g_j \in \mathcal{C}$, such that g_k belongs to g_j. Given some $s \in G_j$, we write $(g_k, g_j, s) \in R$ (or $R(g_k, g_j, s)$) to denote that the value of g_k is* relevant *when g_j has value s.*

Formally, R is the smallest relation such that

1. *If $g_k \in \mathcal{D}$ is directly used in some $i \in I$, g_k belongs to some $g_j \in \mathcal{C}$ and $s = source(i, g_j)$, then $R(g_k, g_j, s)$;*
2. *If $R(g_l, g_j, t)$, and there exists an $i \in I$ such that $(s, i, t) \in E_{g_j}$, and g_k belongs to g_j, and $g_k \in \mathrm{pars}(n_{i,l})$, then $R(g_k, g_j, s)$;*
3. *If $R(g_l, g_p, t)$, and there exists an $i \in I$ and an r such that $(r, i, t) \in E_{g_p}$, and $g_k \in \mathrm{pars}(n_{i,l})$, and g_k belongs to some CFP g_j to which g_l does not belong, and $s = source(i, g_j)$, then $R(g_k, g_j, s)$.*

If $(g_k, g_j, s) \notin R$, we write $\neg R(g_k, g_j, s)$ and say that g_k is irrelevant *when $g_j = s$.*

Example 5.17. We now apply the definition of relevance to our running example. The first condition immediately yields $R(x, a, 2)$ and $R(y, b, 2)$. Then, no clauses apply anymore, so $\neg R(x, a, 1)$ and $\neg R(y, b, 1)$. Now, to makes things more interesting we hide the action *comm*, obtaining

$$\begin{array}{llll}
\multicolumn{4}{l}{X(a : \{1,2\}, b : \{1,2\}, x : \mathbb{N}, y : \mathbb{N}) =} \\
\sum_{d:\mathbb{N}} & a = 1 & \Rightarrow \mathrm{read}(d) \sum_{c:\{0,1\}} \frac{1}{2} : X(2, b, d + c, y) & (1) \\
+ & a = 2 \wedge b = 1 & \Rightarrow \tau \cdot X(1, 2, x, x) & (2) \\
+ & b = 2 & \Rightarrow \mathrm{write}(y) \cdot X(a, 1, x, y) & (3)
\end{array}$$

In this case, the first clause of relevance only yields $R(y, b, 2)$. Moreover, since x is used in summand 2 to determine the value that y will have when b becomes 2, also $R(x, a, 2)$. Formally, this can be found using the third clause, substituting $g_l = y$, $g_p = b$, $t = 2$, $i = 2$, $r = 1$, $g_k = x$, $g_j = a$, and $s = 2$. □

Remark 5.18. The computation of relevance is by far the most complex part of our approach. It depends on the number of data parameters $|\mathcal{D}|$, the number of control flow parameters $|\mathcal{C}|$ and the number of summands $|I|$.

To construct R, first all directly relevant pairs are computed using the first clause of Definition 5.16. For each summand i we check for each DP g_k whether it is directly used. If so, for all CFPs g_j to which g_k belongs the pair (g_k, g_j, s) is added to R, with s the source of g_j for summand i. Hence, in the worst case $O(|I| \times |\mathcal{D}| \times |\mathcal{C}|)$ operations are performed. The fixed-point computation continues, and each iteration adds at least one relevant pair to R. Since each CFP can have at most $|I|$ sources, the number of iterations is bound by $O(|I| \times |\mathcal{D}| \times |\mathcal{C}|)$ as well. Per iteration, for each pair that is already in R

(which worst case has size $O(|I| \times |\mathcal{D}| \times |\mathcal{C}|)$), all $|I|$ summands are traversed to check if there is a data parameter to satisfy the second clause of Definition 5.16. For the third clause, a DP and a CFP are sought after. Hence, this is worst-case in $O(|I|^2 \times |\mathcal{D}|^2 \times |\mathcal{C}|^2)$. Combining these observations, the worst-case time complexity of dead variable reduction is in $O(|I|^3 \times |\mathcal{D}|^3 \times |\mathcal{C}|^3)$.

In practice, however, the number of iterations is very unlikely to be so large. For instance, one of our case studies that will be discussed in Chapter 9 (`leader-3`) has 100 summands, 13 CFPs and 14 DPs. Instead of the worst-case $100 \cdot 13 \cdot 14 = 18{,}200$ iterations, only 3 were needed. Other case studies required 2 instead of the maximal 360 iterations (`polling-4`), and 4 instead of the maximum of 7,350 (`hesselink-3`). The final case study had no CFPs (`grid-3`). Hence, in practice the number of iterations seems not to depend on $|I|$, $|\mathcal{D}|$ and $|\mathcal{C}|$. Additionally, we rarely come across summands that are ruled by more than two CFPs. As CFPs can only change value in summands they rule, it therefore seems realistic to assume that the number of pairs (c, s), with c a CFP and s a source value for c, is in $O(|I|)$. This implies that the size of R is in $O(|I| \times |\mathcal{D}|)$ instead of $O(|I| \times |\mathcal{D}| \times |\mathcal{C}|)$. Assuming the number of iterations to be in $O(1)$, and the size of R in $O(|I| \times |\mathcal{D}|)$, this brings our total complexity down to $O(|I|^2 \times |\mathcal{D}|^2 \times |\mathcal{C}|)$. □

Since clusters have only limited information, they do not always detect a DP's irrelevance. However, they always have sufficient information to never erroneously declare a DP irrelevant. Therefore, we define a DP g_k to be relevant given a state vector $\boldsymbol{v}$, if and only if it is relevant for the valuations of *all* CFPs g_j it belongs to.

Definition 5.19 (Relevance in state vectors). *The* relevance *of a parameter $g_k \in J$ given a state vector $\boldsymbol{v}$, denoted by $Relevant(g_k, \boldsymbol{v})$, is defined by*

$$Relevant(g_k, \boldsymbol{v}) = \bigwedge_{\substack{g_j \in \mathcal{C} \\ g_k \text{ belongs to } g_j}} R(g_k, g_j, v_j)$$

Note that, since a CFP belongs to no parameters, it is always relevant.

Example 5.20. For our running example we derived that x belongs to a, and that it is irrelevant when $a = 1$. Therefore, the valuation $x = 5$ is not relevant in the state vector $\boldsymbol{v} = (1, 2, 5, 3)$, so we write $\neg\, Relevant(x, \boldsymbol{v})$. □

Intuitively, the value of a DP that is irrelevant in a state vector does not matter. For instance, the two state vectors $\boldsymbol{v} = (1, 2, 3)$ and $\boldsymbol{v}' = (1, 5, 3)$ are equivalent if $\neg\, Relevant(g_2, \boldsymbol{v})$. To formalise this, we introduce a relation $\cong$ on state vectors, given by

$$\boldsymbol{v} \cong \boldsymbol{v}' \iff \forall g_k \in J\colon\ (Relevant(g_k, \boldsymbol{v}) \implies v_k = v'_k)$$

and prove that it is a derivation-preserving bisimulation (considering each vector $\boldsymbol{v}$ to behave as the process $X(\boldsymbol{v})$. This is an important result: it shows us that irrelevant DPs can indeed safely be modified without influencing an LPPE's behaviour.

Theorem 5.21. *The relation $\cong$ is a derivation-preserving bisimulation.*

5.2.2 Changing the initial state

Linearisation chooses dummy values for parameters whose initial value does not matter. These values are chosen locally per component. However, after generating an LPPE (or MLPE) global information is available, making it possible to choose more intelligent values.

Although the main purpose of Theorem 5.21 is to reset dead variables (as discussed in the next section), it can also be used to see that the initial value of a DP g_k, belonging to a CFP g_j, can be modified if $\neg R(g_k, g_j, init_j)$. If possible, the initial value of a parameter should be chosen such that it is not changed by any summand, as it can then be removed by constant elimination (see Section 4.5). This extension has not been included in our implementation yet.

5.3 State space reduction using data flow analysis

The most important application of the data flow analysis described in the previous section is to reduce the number of reachable states of the PA underlying an LPPE. We need to do this in a careful manner, as modifications to irrelevant parameters in an arbitrary way could even increase this number. We present a syntactic transformation of LPPEs, and prove that it yields a derivation-preserving bisimilar system and can never increase the number of reachable states. In several practical examples, it yields a decrease.

5.3.1 Syntactic transformation

Our transformation uses the idea that a data parameter g_k that is irrelevant in all possible states after taking a summand i, can just as well be reset by i to its initial value. Resetting to a different value could result in an increase of the number of states, in case some variable is irrelevant in the initial state. The reason for this is that no resets have yet taken place for the initial state itself. Hence, if at some point the initial state is revisited in the original system, an additional state may be generated by the transformed system due to a variable reset.

Definition 5.22 (LPPE transformation). *Given an LPPE X*

$$X(\boldsymbol{g} : \boldsymbol{G}) = \sum_{i \in I} \sum_{\boldsymbol{d_i} : \boldsymbol{D_i}} c_i \Rightarrow a_i(\boldsymbol{b_i}) \sum_{\boldsymbol{e_i} : \boldsymbol{E_i}}\!\!\!\!\!\!\bullet\;\; f_i : X(\boldsymbol{n_i})$$

its transform *is the LPPE X' given by*

$$X'(\boldsymbol{g} : \boldsymbol{G}) = \sum_{i \in I} \sum_{\boldsymbol{d_i} : \boldsymbol{D_i}} c_i \Rightarrow a_i(\boldsymbol{b_i}) \sum_{\boldsymbol{e_i} : \boldsymbol{E_i}}\!\!\!\!\!\!\bullet\;\; f_i : X'(\boldsymbol{n'_i})$$

with

$$n'_{i,k} = \begin{cases} n_{i,k} & , \textit{if} \quad \bigwedge\limits_{\substack{g_j \in \mathcal{C} \\ g_j \textit{ rules } i \\ g_k \textit{ belongs to } g_j}} R(g_k, g_j, dest(i, g_j)) \\ init_k & , \textit{otherwise} \end{cases}$$

We will use the notation $X(\boldsymbol{v})$ to denote state $\boldsymbol{v}$ in the underlying PA of X, and $X'(\boldsymbol{v})$ to denote state $\boldsymbol{v}$ in the underlying PA of X'.

Note that n'_i only deviates from n_i for parameters g_k that are irrelevant after taking i. The following theorem states that the transformation satisfies derivation-preserving bisimilation for decodable LPPEs.

Theorem 5.23. *Let X be a decodable LPPE, X' its transform, and $\boldsymbol{v}$ a state vector for X. Then, $X(\boldsymbol{v}) \sim_{\text{dp}} X'(\boldsymbol{v})$.*

Since our transformation additionally does not alter any of the action-prefix constructs, it can never make an LPPE non-decodable. Hence, by Theorem 4.36 this transformation can also safely be applied to MLPE specifications.

We now show that our choice for redefining each n'_i ensures that the state space of X' is at most as large as the state space of X. This is based on the following invariant, stating that if a parameter is irrelevant, it is equal to its initial value.

Proposition 5.24. *For any state vector $\boldsymbol{v}$ reachable by the process $X'(\boldsymbol{init})$, invariably $\neg\, Relevant(g_k, \boldsymbol{v})$ implies that $v_k = init_k$.*

Theorem 5.25. *The number of reachable states of $X'(\boldsymbol{init})$ is at most the number of reachable states of $X(\boldsymbol{init})$.*

Since we prove the above theorem by providing a derivation-preserving functional bisimulation from X to X' (see Appendix A.3), the result also holds if our technique is applied to MAPA specifications by linearising, encoding to LPPE, resetting dead variables and decoding back to MLPE.

Example 5.26. Using the above transformation, the LPPE of our running example becomes

$$\begin{array}{llll}
\multicolumn{3}{l}{X(a : \{1,2\}, b : \{1,2\}, x : \mathbb{N}, y : \mathbb{N}) =} & \\
\quad \sum_{d:\mathbb{N}} & a = 1 & \Rightarrow \text{read}(d) \sum_{c:\{0,1\}} \frac{1}{2} : X(2, b, d + c, y) & (1) \\
\quad + & a = 2 \wedge b = 1 & \Rightarrow \text{comm}(x) \cdot X(1, 2, \underline{1}, x) & (2) \\
\quad + & b = 2 & \Rightarrow \text{write}(y) \cdot X(a, 1, x, \underline{1}) & (3)
\end{array}$$

assuming that the initial state vector is $(1, 1, 1, 1)$. The variable resets have been underlined for comparison.

Note that for X' the state $(1, 1, x', y')$ is only reachable for $x' = y' = 1$, whereas in the original specification X it is reachable for all $x', y' \in \mathbb{N}$ such that $x' = y'$. □

5.3.2 Lack of idempotency

Generally it is considered desirable for a reduction technique to be idempotent; applying it once yields the maximum effect it can possibly achieve, and applying it more often does not have any additional effect. Unfortunately, our method is not idempotent in general, although experiments show that in practice it often is. As an example of the lack of idempotency, consider the following LPPE.

$$\begin{array}{rlll} \multicolumn{3}{l}{X(p\colon \{1,2,3\}, q\colon \{1,2\}, x\colon \mathbb{N}) =} & \\ & p = 2 \wedge q = 1 & \Rightarrow a(x) \cdot X(1,1,x) & (1) \\ + & p = 1 \wedge q = 1 & \Rightarrow \tau \cdot X(2,2,x+1) & (2) \\ + & p = 3 & \Rightarrow \tau \cdot X(1,q,x) & (3) \end{array}$$

For this specification, p and q are both CFPs. Furthermore, p rules all summands, whereas q rules only the first two. Since x is unchanged in the third summand, however, it still belongs to both.

Clearly, $R(x,p,2)$ and $R(x,q,1)$, since in that case the first summand (which directly uses x) can be taken. Moreover, $R(x,p,1)$, since the second summand can be taken if $p = 1$, which uses x to set x in the next state and updates p to 2. Finally, $R(x,p,3)$ due to the third summand.

However, because the second summand changes q to 2 and x is not relevant when $q = 2$, its next-state vector can be changed to $(2,2,0)$, obtaining

$$\begin{array}{rlll} \multicolumn{3}{l}{X(p\colon \{1,2,3\}, q\colon \{1,2\}, x\colon \mathbb{N}) =} & \\ & p = 2 \wedge q = 1 & \Rightarrow a(x) \cdot X(1,1,x) & (1) \\ + & p = 1 \wedge q = 1 & \Rightarrow \tau \cdot X(2,2,\underline{0}) & (2) \\ + & p = 3 & \Rightarrow \tau \cdot X(1,q,x) & (3) \end{array}$$

The next-state vector of the third summand could not (yet) be transformed, since it changes p to 1 and it was established that x is relevant when $p = 1$.

However, starting over based on the transformed LPPE, it can be seen that x is not relevant anymore when $p = 1$, since it is not used in the next-state vector. Therefore, x is also not relevant anymore for $p = 3$, so that now the next-state vector of the third summand can be changed to $(1,q,0)$.

This example shows that the lack of idempotency is due to our control flow analysis being local; at first sight it seemed that the first summand would be reachable after the second. As discussed before, we do not construct a control flow graph of the complete system for efficiency reasons.

5.4 Failing alternatives

We discuss two potential adaptations to the theory: (1) allowing CFPs to belong to other CFPs, and (2) relaxing the definition of the belongs-to relation. While both adaptations may intuitively seem like an improvement, they can also cause difficulties. Hence, we provide examples to motivate the current state of the theory.

5.4.1 Allowing CFPs to belong to other CFPs

A possible adaptation to the theory may seem to allow CFPs to belong to other CFPs. This, however, would raise problems if cycles occur in the belongs-to relation. Consider for example the following LPPE.

$$\begin{aligned} & X(p\colon \{1,2\}, q\colon \{1,2\}) = \\ & \quad p = 1 \wedge q = 1 \Rightarrow \tau \cdot X(2,2) \end{aligned}$$

with $(2,2)$ as the initial state vector. Clearly, this system can perform no actions.

If CFPs could belong to CFPs, in this case p would belong to q and q would belong to p. Furthermore, we obtain $\neg R(p,q,2)$ and $\neg R(q,p,2)$. Therefore, the initial condition seems to be allowed to change to $(1,1)$. However, in that case we obtain a specification that is not strongly bisimilar to the original anymore, since it can perform a τ-action.

5.4.2 Relaxing the definition of belongs-to

The definition of belongs-to could be relaxed to also allow DPs to be changed to a constant value in summands that are not ruled by the CFPs they belong to. That is, a DP g_k belongs to a CFP g_j if all summands $i \in I$ that use or change g_k to a *non-constant value* are ruled by g_j. In this case, the technique is still correct; our proofs can easily be adapted. As an example of a useful application of this change, observe the LPPE

$$\begin{aligned} & X(p\colon \{1,2\}, x\colon \mathbb{N}) = \\ & \quad\ \ p = 1 \Rightarrow \tau \cdot X(1, x+1) \quad (1) \\ & + \ p = 2 \Rightarrow a(x) \cdot X(1, x) \quad (2) \\ & + \qquad\qquad \tau \cdot X(p, 0) \quad (3) \end{aligned}$$

For this infinite-state system parameter p rules the first two summands, but x is changed in the last so according to the original definition x would not belong to p and no reductions can be made. However, using the new definition, x does belong to p, and we observe that x is only relevant when $p = 2$. Therefore, we can reduce to the following LPPE, obtaining a finite-state system.

$$\begin{aligned} & X(p\colon \{1,2\}, x\colon \mathbb{N}) = \\ & \quad\ \ p = 1 \Rightarrow \tau \cdot X(1, 0) \quad (1) \\ & + \ p = 2 \Rightarrow a(x) \cdot X(1, 0) \quad (2) \\ & + \qquad\qquad \tau \cdot X(p, 0) \quad (3) \end{aligned}$$

However, with the adapted definition it is also possible to increase the state space due to a 'reduction'; consider for instance the following LPPE.

$$\begin{aligned} & X(p\colon \{1,2\}, q\colon \{1,2\}, x\colon \{1,2\}) = \\ & \quad\ \ p = 1 \Rightarrow \tau \cdot X(2, q, 2) \quad (1) \\ & + \ q = 1 \Rightarrow a(x) \cdot X(p, 2, x) \quad (2) \end{aligned}$$

Using the initial state vector $(1,1,1)$, this system has a state space consisting of four states. Using the adapted definition of belongs-to we find that x belongs to q, and that x is not relevant when $q = 2$, so that reduction results in

$$\begin{aligned} &X(p\colon \{1,2\}, q\colon \{1,2\}, x\colon \{1,2\}) = \\ &\qquad p = 1 \Rightarrow \tau \cdot X(2, q, 2) \qquad (1) \\ &+\quad q = 1 \Rightarrow a(x) \cdot X(p, 2, 1) \qquad (2) \end{aligned}$$

This specification results in a state space with five states, so Corollary 5.25 does not apply anymore.

5.5 Case study

A restricted version of the techniques described in this chapter was implemented by Jaco van de Pol in the non-quantitative context of the μCRL toolset. Later, the author implemented them in his quantitative tool SCOOP. Chapter 9 will discuss our implementation and several case studies in detail, but we already present one of the larger case studies here to show our technique in practice. It is a model of a *handshake register*, modelled and verified by Hesselink [Hes98].

A handshake register is a data structure that is used for communication between a single reader and a single writer. It guarantees *recentness* and *sequentiality*; any value that is read was at some point during the read action the last value written, and the values of sequential reads occur in the same order as they were written. Also, it is *waitfree*; both the reader and the writer can complete their actions within a bounded number of steps, independent of the other process. Hesselink provides a method to implement a handshake register of a certain data type based on four so-called safe registers and four atomic boolean registers.

We modelled both the specification of the handshake register and its implementation using the registers, aiming at demonstrating their equality. The model of the implementation, though, is rather complex and hence has a large state space (quickly reaching millions of states for larger variants of the data type stored by the handshake register). We will show in Chapter 9 that our dead variable reduction technique provides a substantial reduction of this state space: at best, it is reduced to less than 0.5% of its original size. To see why, observe the LPPE of each of the four safe registers:

$$\begin{aligned} &Y(i\colon \text{Bool}, j\colon \text{Bool}, r\colon \{1,2,3\}, w\colon \{1,2,3\}, v\colon D, vw\colon D, vr\colon D) = \\ &\qquad\qquad r = 1 \Rightarrow \text{beginRead}(i,j) \cdot Y(i,j,2,w,v,vw,\underline{vr}) && (1) \\ &+ \qquad r = 2 \land w = 1 \Rightarrow \tau \cdot Y(i,j,3,w,v,\underline{vw},v) && (2) \\ &+ \textstyle\sum_{x\colon D}\ r = 2 \land w \neq 1 \Rightarrow \tau \cdot Y(i,j,3,w,v,vw,x) && (3) \\ &+ \qquad r = 3 \Rightarrow \text{endRead}(i,j,vr) \cdot Y(i,j,1,w,v,vw,\underline{vr}) && (4) \\ &+ \textstyle\sum_{x\colon D}\ w = 1 \Rightarrow \text{beginWrite}(i,j,x) \cdot Y(i,j,r,2,\underline{v},x,vr) && (5) \\ &+ \qquad w = 2 \Rightarrow \tau \cdot Y(i,j,r,3,vw,\underline{vw},vr) && (6) \\ &+ \qquad w = 3 \Rightarrow \text{endWrite}(i,j) \cdot Y(i,j,r,1,v,\underline{vw},vr) && (7) \end{aligned}$$

The boolean parameters i and j are process identifiers to distinguish the four components (Y is instantiated four times, once for each combination of valuations for i and j). The parameter r denotes the read status, and w the write status.

Reading consists of a *beginRead* action, a τ step, and an *endRead* action. During the τ step either the contents of v is copied into vr (summand 2), or, when writing is taking place at the same time, a random value is copied to vr (summand 3). In the final step, the *endRead* action is taken with the value of vr as a parameter (summand 4). Writing works by first storing the value to be written in vw (summand 5), and then copying vw to v (summand 6).

Our dead variable technique automatically detects several possibilities for variable resets, indicated by the underlined variables in the specification. These are due to r and w being recognised as manually encoded control flow parameters. Based on these CFPs, our implementation detects that the value of vr is irrelevant after summand 4, since it will not be used before summand 4 is reached again. This is always preceded by summand 2 or 3, both overwriting vr. Thus, vr can be reset to its initial value in the next-state expression of summand 4. These resets drastically decrease the size of the state space, as our experiments will show in Section 9.3.1. The CADP tool [GS06] was not able to make these reductions, since it does not contain control flow reconstruction.

Note that the use of parallel processes for the reader and the writer instead of our solution of encoding control flow in the data parameters—which would limit the need for control flow reconstruction—would be difficult, because of the shared variables. Additionally, although the example may seem artificial, it is an almost one-to-one formalisation of its description in [Hes98]. Without our method for control flow reconstruction, useful variable resets could not be found automatically. Manual optimisation would be hard and error-prone.

5.6 Contributions

We presented a novel method for reconstructing the local control flow of linear processes. The reconstruction process enables us to interpret some variables as program counters. Especially when specifications are translated between languages, their control flow may be hidden in the state parameters (as showed by the case study presented in this chapter). To the best of our knowledge, no such reconstruction method appeared in literature before, and hence other tools are not able to do this analysis. As an exception, our techniques have recently been used and adapted [KWW13] to the context of PBESs (parameterised Boolean equation systems). This now allows dead variable reduction to also be applied in the mCRL2 toolset [CGK$^+$13].

The reconstructed control flow is used for data flow analysis, aiming at state space reduction by resetting variables that are irrelevant given a certain state. We introduced a transformation and proved its correctness with regard to derivation-preserving bisimilarity, as well as its property to never increase the state space.

By finding useful variable resets automatically, users can focus on modelling systems in an intuitive way, instead of formulating models such that a toolset can handle them best. This idea of automatic syntactic transformations for improving the efficiency of formal verification (not relying on users to make their models as efficient as possible) already proved to be a fruitful concept in earlier work [WH00]. Our case studies in Chapter 9 will show that significant reductions can indeed be obtained, sometimes shrinking state spaces to less than 1% of their original size.

Part III

Confluence Reduction

CHAPTER 6

Confluence Reduction

"*Art is the elimination of the unnecessary.*"

Pablo Picasso

The previous chapters presented techniques for reducing MAPA specifications while preserving strong bisimulation. Although these are all very useful, we can do much better in many cases: we can abstract from internal behaviour, reducing more while preserving divergence-sensitive branching bisimulation[1].

This chapter therefore generalises *confluence reduction* from LTSs to MAs. It is a powerful state space reduction technique based on commutativity of transitions, removing spurious nondeterminism often arising from the parallel composition of largely independent components. The core is a *confluent set* of invisible transitions; these are chosen in such a way that they always connect bisimilar states. Confluence therefore paves the way for state space reductions, by giving confluent transitions priority over their neighbouring transitions.

The aim of our analysis is to efficiently approximate which invisible transitions are confluent, and hence do not influence the observable behaviour. Although it may be beneficial to know *all* confluent transitions, this is often infeasible in practice as it may require the entire state space—however, generating the unreduced system is precisely what we want to prevent. Hence, we settle for an underapproximation. To the best of our knowledge, it is the first technique of this kind for MAs.

Our approach. In the probabilistic setting, transitions do not necessarily have a unique target state. Using an example, we argue that only τ-transitions with a unique target state can be considered confluent. Additionally, we generalise the concept of commutativity to the probabilistic realm, taking into account the probability of commuting paths. Based on these ideas, we define our novel notion of confluence for MAs. As in the non-probabilistic case, it specifies sufficient conditions for invisible transitions to not alter the behaviour of an MA; i.e., if a transition is found to be confluent in a state s, it may be given priority over all other transitions emanating from s.

We prove that confluent transitions connect divergence-sensitive branching bisimilar states, and present a mapping of states to representatives to efficiently

[1]Since this chapter is only concerned with divergence-sensitive branching bisimulation, we will often abbreviate this concept by just the term *bisimulation*.

generate a reduced MA. We explain how confluence can be detected symbolically on specifications in the MAPA language, and provide a detailed exposition of our heuristics to do so efficiently. Case studies in Chapter 9 applying these heuristics will demonstrate state space reduction up to more than 90%, with decreases in analysis time of sometimes more than 98%. We conclude with a discussion on possible alternatives to our notions and show why these fail.

Related work. Compared to the earlier approach on confluence reduction for process algebras [BvdP02], the notion of confluence we present in this chapter is different in three important ways:

- We can handle MAs, lifting confluence reduction to a significantly larger class of systems (including the subclasses of PAs and IMCs).
- We discuss a subtle flaw in the earlier work [BvdP02], and solve it by introducing an underlying classification of the interactive transitions. This way we guarantee closure under unions, something that was not guaranteed before[2]. It is key to the way we detect confluence on MAPA specifications.
- We preserve divergence and hence minimal reachability probabilities.

Outside the domain of confluence reduction, our technique is most related to *partial order reduction* (POR). Several types of POR have been defined, both for non-probabilistic [Val90, Val93, Pel93, GP93, God96] and probabilistic systems [DN04, BGC04, BDG06]. These techniques are based on ideas similar to confluence, choosing a subset of the outgoing transitions per state to reduce the state space while preserving a certain notion of bisimulation or trace equivalence. None of these techniques is capable of reducing MAs. For the subclass of PAs, we provide a detailed comparison in Chapters 7 and 8, showing confluence reduction to be more powerful in theory as well as practice when restricting to the preservation of branching-time properties.

Since none of the existing techniques is able to deal with MAs, we believe that our generalisation—the first on-the-fly reduction technique for MAs abstracting from internal transitions—is a major step forward in efficient quantitative verification.

Organisation of the chapter. We start in Section 6.1 by presenting an informal introduction to the concept of confluence reduction. Since the additional technical difficulties due to probabilities and Markovian rates may distract from the underlying ideas, we just consider confluence for LTSs in this introduction. Section 6.2 then discusses the additional difficulties when defining confluence for probabilistic systems, introduces our novel notion of confluence for MAs,

[2]Actually, the approach taken in [BvdP02] resembles our approach to a large degree (except that they consider a less expressive model and do not preserve divergence). While their claim of being able to take the union of confluent sets is faulty in general, their implementation works correctly. We show in this chapter that an additional technical restriction (the confluence classification) is needed to remedy the theoretical mistake, and this restriction happens to be satisfied in their implementation (in the same way as in ours). Since the confluence reduction technique described in [BvdP02] is a restricted variant of our notion, it could be fixed by introducing a confluence classification precisely in the same way as we do in this chapter.

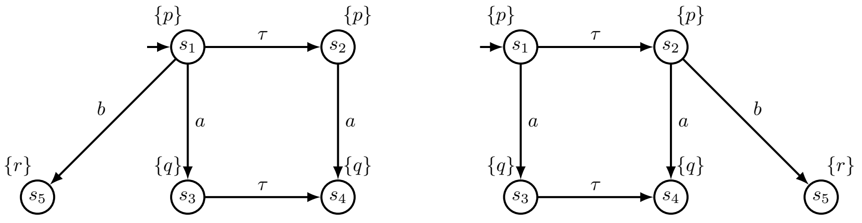
and proves closure under unions as well as the fact that confluent transitions connect divergence-sensitive branching bisimilar states. Then, we present our state space reduction technique based on confluence and representation maps in Section 6.3. Section 6.4 provides a logical characterisation and heuristics for detecting confluence on the MAPA language, and Section 6.5 discusses the disadvantages of some variations on our concept. Finally, Section 6.6 concludes by summarising the contributions of this chapter.

Origins of the chapter. The results in this chapter on confluence reduction were first published for the context of probabilistic automata and prCRL, in the proceedings of the *17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (TACAS) [TSvdP11] and a corresponding technical report [TSvdP10]. The generalisation to MAs and MAPA was published in the proceedings of the *11th International Conference on Formal Modeling and Analysis of Timed Systems* (FORMATS) [TSvdP13a] and a corresponding technical report [TSvdP13b].

6.1 Informal introduction

The concept of confluence reduction is based on the idea that some transitions do not influence the observable behaviour of a system—assuming that only visible actions (i.e., actions different from τ) and changes in the validity of atomic propositions can be observed. To this end, they at least have to be invisible themselves: their action should be τ and they should not change the state labelling, i.e., they should be stuttering. Still, invisible transitions *may* influence the observable behaviour of an MA, even though they are not observable themselves. Figure 6.1 illustrates this phenomenon.

Example 6.1. While the transition $s_1 \xrightarrow{\tau} s_2$ in Figure 6.1(a) cannot be observed itself, it does disable the b-transition. Hence, this transition influences the observable behaviour of the system—if it was always taken from s_1 while omitting the other two transitions emanating from this state, no b-action would ever be observed and the atomic proposition r would never hold. Therefore, states s_1 and s_2 are not branching bisimilar, and hence neither are the original system and the proposed reduced system.



(a) Observable invisible transition $s_1 \xrightarrow{\tau} s_2$. (b) Unobservable invisible transition $s_1 \xrightarrow{\tau} s_2$.

Figure 6.1: Observable versus unobservable invisible transitions.

The invisible transition $s_1 \xrightarrow{\tau} s_2$ in Figure 6.1(b) is of a different type: we could always take it from s_1 while ignoring the transition $s_1 \xrightarrow{a} s_3$, without losing any observable behaviour. Both the original system and the system reduced in this way can eventually see an a or a b-action and may end up in a state satisfying either q or r. Actually, states s_1 and s_2 are branching bisimilar, and so are the original system and the proposed reduced system. □

The example illustrates the most important property of confluent transitions: they connect branching bisimilar states, and hence in principle could be given priority over their neighbouring transitions without losing any behaviour. To verify that a transition is confluent and hence has this property, it should be invisible *and* still allow all behaviour enabled from its source state to occur from its target state as well. Stated otherwise, all other transitions from its source state should be *mimicked* from its target state.

6.1.1 Checking for mimicking behaviour

To check whether all behaviour from a transition's source state is also enabled from its target state, confluence employs a coinductive approach similar to the common definitions of bisimulation. For an invisible transition $s \xrightarrow{\tau} s'$ to be confluent, clearly the existence of a transition $s \xrightarrow{a} t$ should imply the existence of a transition $s' \xrightarrow{a} t'$ for some t'. Additionally, for all behaviour from s to be present at s', also all behaviour from t should be present at t'. To achieve this, we coinductively require s' to have a confluent transition to t'. We note that a coinductive approach such as the one just described requires a set of transitions to be defined upfront. Then, we can validate whether or not this set indeed satisfies the conditions for it to be confluent. In principle, we are always interested in finding the *largest* set for which this is the case.

Example 6.2. In Figure 6.1(a), the set containing both τ-transitions is *not* confluent. After all, for $s_1 \xrightarrow{\tau} s_2$ it is not the case that every action enabled from its source state is also enabled from its target state.

In Figure 6.1(b), the set containing both τ-transitions *is* confluent. For $s_3 \xrightarrow{\tau} s_4$ the mimicking condition is satisfied trivially, since it does not have any neighbouring transitions from s_3. For $s_1 \xrightarrow{\tau} s_2$ the condition is also satisfied, since the transition $s_1 \xrightarrow{a} s_3$ is mimicked by $s_2 \xrightarrow{a} s_4$. As required, s_3 and s_4 are indeed connected by a confluent transition. □

6.1.2 State space reduction based on confluence

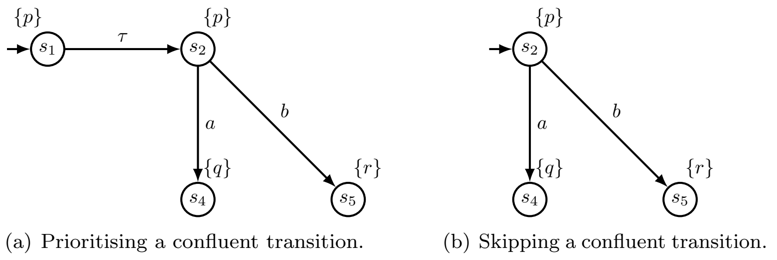
When a transition is confluent, it can in principle be given priority, in the sense that all other transitions emanating from the same state are omitted. This may yield significant state space reductions, as many states may become unreachable. Although a system obtained due to prioritisation of confluent transitions is indeed branching bisimilar to the original system (under some assumptions that we come back to later), it often contains states that could just as well be omitted. They only have one outgoing invisible transition, and hence do not contribute to the system's observable behaviour in any way. Therefore, instead of prioritising confluent transitions, we rather skip over them.

Example 6.3. Consider again the toy example presented in Figure 6.1(b). We already discussed in the previous example that both invisible transitions are confluent. Figure 6.2(a) demonstrates the reduced state space when giving these transitions priority over their neighbours. Although this is a valid reduction, state s_1 has little purpose and can be skipped over. That way, we obtain the system illustrated in Figure 6.2(b). □

Both the idea of prioritising transitions and the idea of skipping over them only work in the absence of cycles of confluent transitions. To see why, consider the system depicted in Figure 6.3(a). All invisible transitions are confluent, as can easily be checked. However, when continuously omitting all non-confluent transitions (as in Figure 6.3(b)), the a-transition is postponed forever and the atomic proposition q will never hold. Clearly, such a reduction does not preserve all properties and hence the reduced system is not branching bisimilar to the original. This problem is well known in partial order reduction as the *ignoring problem* [Val90, EP10], and dealt with by requiring the reduction to be acyclic. That is, no cycle should be present of states that are all omitting some of their transitions. Indeed, this requirement is violated in Figure 6.3(b). As a solution, we could also require reductions to be acyclic, forcing at least one state of a cycle to be fully explored.

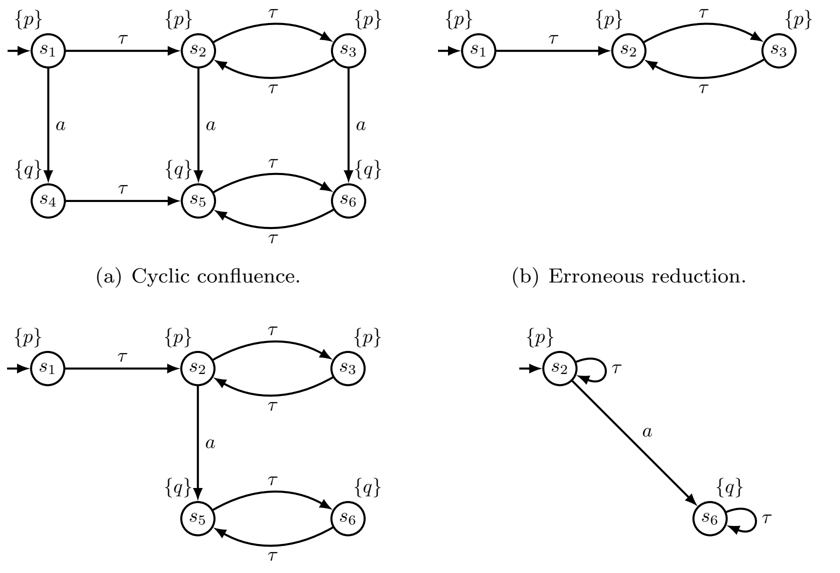
Example 6.4. Figure 6.3(c) shows the result of reducing Figure 6.3(a) based on the idea of prioritisation while forcing at least one state on a cycle to be fully explored (in this case, state s_2 was chosen as the fully explored state). □

The technique of skipping over confluent transitions can also be extended to work in the presence of cycles of confluent transitions. In the absence of such cycles, the approach simply boils down to skipping over confluent transitions until reaching a state without any outgoing confluent transitions (state s_2 in Figure 6.2(b)). In the presence of cycles, we just continue until reaching the bottom strongly connected component (BSCC) of the subgraph when considering only the confluent transitions. Due to some technical requirements on confluent transitions, there always is a unique BSCC reachable from every state (BSCC $\{s_2, s_3\}$ for s_1 in Figure 6.3(a)). In this BSCC, we select one state to be the *representative* for all states that can reach it by skipping over



(a) Prioritising a confluent transition. (b) Skipping a confluent transition.

Figure 6.2: State space reduction based on confluence.



(a) Cyclic confluence.

(b) Erroneous reduction.

(c) Acyclic reduction using prioritisation.

(d) Acyclic reduction using representatives.

Figure 6.3: Confluence reduction in the presence of cyclic confluent transitions.

confluent transitions. Since confluent transitions never disable behaviour, such a representative state has all behaviour of the states that it represents. The representative state is fully explored, and all transitions to states that can reach that representative by confluent transitions are redirected towards the representative.

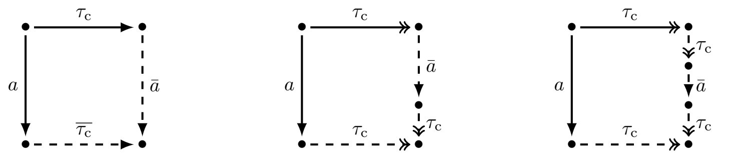
Example 6.5. Figure 6.3(d) illustrates the result of the approach with representative states. Here, the state s_2 was selected as representative of s_1, s_2 and s_3, and s_6 was selected as representative of s_5 and s_6.

Note that both reduction approaches yield systems that are branching bisimilar to the original system, but that the representatives approach allows for much more reduction. □

6.1.3 Traditional notions of confluence

For non-probabilistic systems, several notions of confluence exist [Blo01, BvdP02]. Basically, as discussed above, they all require that if an action a is enabled from a state that also enables a confluent τ-transition, then both

1. The action a is still enabled after taking the confluent τ-transition (possibly allowing some additional confluent τ-transitions first), and
2. We can always end up in the same state traversing only confluent τ-steps and the a-step, no matter whether we started by the τ- or the a-transition.



(a) Strong confluence. (b) Confluence. (c) Weak confluence.

Figure 6.4: Three variants of confluence.

Figure 6.4 depicts three notions of confluence from [Blo01], ordered from strong to weak—the strength denoting their ability to distinguish systems, similar to the terminology for notions of bisimulation. The stronger notions are easier to detect, but less powerful in their reductions. In these diagrams, the notation τ_c is used for confluent τ-transitions, and a can either be τ or an observable action. The diagrams should be interpreted as follows: for any state from which the solid transitions are enabled (universally quantified), there should be a matching for the dashed transitions (existentially quantified). A double-headed arrow denotes a path of zero of more transitions with the corresponding label, and an arrow with label $\overline{a}$ denotes a step labelled with a that is optional in case $a = \tau$ (i.e., its source and target state may then coincide). Additionally, confluent transitions are required to be mimicked by confluent transitions; i.e., if the solid a-transition in these diagrams is confluent (and hence $a = \tau$), then so should the dashed one be.

Note that we always first need to find a subset of τ-transitions that we believe are confluent; then, the diagrams are checked.

6.2 Confluence for Markov automata

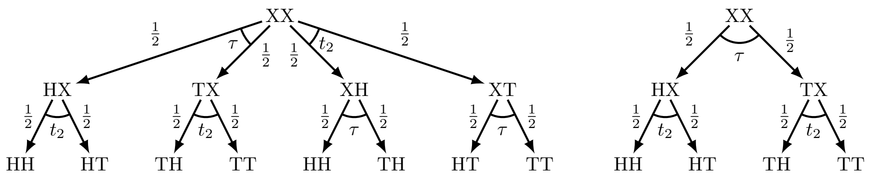
In [TSvdP11] we generalised the three variants of non-probabilistic confluence depicted in Figure 6.4 to the setting of probabilistic automata. In a process-algebraic context such as the MAPA framework, where confluence is detected heuristically over a syntactic description of a system, it is most practical to apply strong confluence. Therefore, we only generalise strong confluence to the Markovian realm and from now on focus on this notion.

First, we discuss some important limitations that arise when defining confluence for systems incorporating probabilities. Since this includes MAs, these limitations also hold for the notion of confluence we define later.

6.2.1 Limitations of probabilistic confluence

For probabilistic systems, the situation is more involved than before, since the target of each transition is a probability distribution, rather than a single state. This yields two limitations to the generalisation of confluence:

1. Only τ-transitions with a unique target state can be considered confluent;
2. A more complicated notion of commutativity is needed.



(a) The original specification. (b) A wrong reduction.

Figure 6.5: Two players throwing dice.

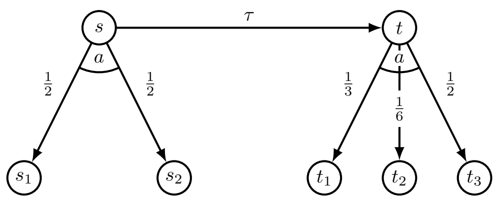
We discuss both issues in detail.

1. Dirac restriction on confluent transitions. We illustrate the first limitation using an example of two players each throwing a die. The PA in Figure 6.5(a) models this behaviour under the assumption that it is unknown who throws first. The first character of each state name indicates whether the first player has not thrown yet (X), or threw heads (H) or tails (T); the second character indicates the same for the second player. For lay-out purposes, some states were drawn twice.

We hid the first player's throw action, and kept the other one visible. Now, it may appear that the order in which the t_2- and the τ-transition occur does not influence the behaviour. However, the τ-step from state XX does not connect bisimilar states (assuming HH, HT, TH, and TT to be distinct). After all, from state XX it is possible to reach a state (XH) from where HH is reached with probability 0.5 and TH with probability 0.5. From HX and TX no such state is reachable anymore. Giving the τ-transition priority in XX, resulting in the PA in Figure 6.5(b), therefore yields a reduced system that is *not* branching bisimilar to the original system. If the τ-transitions were non-probabilistic, this would not have been the case (as we will see in this chapter).

We note that this limitation is due to our goal of preserving branching bisimulation. Future work may focus on preserving different types of bisimulation, that may allow confluent probabilistic transitions.

2. More complicated notion of commutativity. In the non-probabilistic setting of LTSs it is clear that a path $a\tau$ should reach the same state as its corresponding path τa. For probabilistic systems, however, this is more involved as the a-step leads us to a distribution over states. So, how should the partial model shown below be completed for the τ-steps to be confluent?



Since we want confluent τ-transitions to connect bisimilar states, we must assure that s and t are bisimilar. Therefore, the distributions

$$\mu = \{s_1 \mapsto \tfrac{1}{2}, s_2 \mapsto \tfrac{1}{2}\} \qquad \text{and} \qquad \nu = \{t_1 \mapsto \tfrac{1}{3}, t_2 \mapsto \tfrac{1}{6}, t_3 \mapsto \tfrac{1}{2}\}$$

must assign equal probabilities to each *class* of bisimilar states. Basically, given the inductive assumption that all other confluent τ-transitions already connect bisimilar states, this may seem to be the case if $\mu \equiv_R \nu$ for

$$R = \{(s, s') \mid s \twoheadleftarrow\!\!\twoheadrightarrow s' \text{ using only confluent } \tau\text{-transitions}\}$$

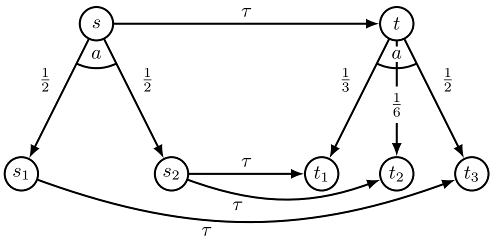
However, due to the fact that the $\twoheadleftarrow\!\!\twoheadrightarrow$ relation allows us to traverse transitions backwards this would go wrong, as explained in more detail in Section 6.5.1. Hence, we need to take a smaller equivalence relation R.

For non-probabilistic confluence (Figure 6.4), given $s \xrightarrow{a} s'$ and $t \xrightarrow{a} t'$, a confluent τ-transition $s' \xrightarrow{\tau_c} t'$ was required. Hence, no transitions from t' to s' or longer paths of transitions were taken into account. We generalise this idea to the probabilistic setting, only considering confluent τ-transitions from the support of μ to the support of ν. Hence, this yields

$$R = \{(s, t) \in \mathit{supp}(\mu) \times \mathit{supp}(\nu) \mid (s \xrightarrow{\tau} t) \in \mathcal{T}\}$$

where $\mathcal{T}$ is the set of confluent transitions[3]. Since the operator $\equiv$ can only be applied to equivalence relations, we actually take the smallest equivalence relation containing the last set R just defined. Note that this implies that overlapping states in the supports of μ and ν do not require a confluent transition.

Now, assuming all τ-transitions to be in $\mathcal{T}$, one way of completing the model above is as follows.



We find that R yields three equivalence classes: $C_1 = \{s, t\}$, $C_2 = \{s_1, t_3\}$ and $C_3 = \{s_2, t_1, t_2\}$. Indeed, μ and ν coincide for these equivalence classes:

$$\mu(C_1) = 0 = \nu(C_1) \qquad \mu(C_2) = \tfrac{1}{2} = \nu(C_2) \qquad \mu(C_3) = \tfrac{1}{2} = \nu(C_3)$$

[3]We could have also chosen to be a bit more liberal, allowing a *path* of $\mathcal{T}$-transitions from s to t. However, we decided to stay closer to the non-probabilistic notion of strong confluence and take the current approach. In addition to simplifying this definition and some proofs later on, it also corresponds more directly to the way we detect confluence heuristically in practice.

6.2.2 Confluence classifications and confluent sets

In the non-probabilistic case, the notion of confluence reduction had a subtle flaw: it relied on the assumption that confluent sets are closed under unions [BvdP02]. In practical applications this is indeed a valid assumption (so the implementation of confluence reduction as described in [BvdP02] was not erroneous), but technically closure under unions of confluent set could not be assumed just like that. This was due to the requirement that confluent transitions have to be mimicked by confluent transitions. When taking the union of two valid sets of confluent transitions, this requirement was possibly invalidated (as explained in more detail in Section 6.5.4).

To solve this problem, we slightly adjust the approach. Before, we just took any subset of the invisible transitions and showed that it was confluent. Now, we impose some more structure, classifying the interactive transitions of an MA into groups upfront—allowing overlap and not requiring all interactive transitions to be in at least one group. We will see that this is natural in the context of MAPA, where the transitions of each summand will form a group together.

At this point, the set of interactive transitions as well as the classification are still allowed to be countably infinite. However, for the representation map approach later on, finiteness is required.

Definition 6.6 (Confluence classification). *Given an MA $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L\rangle$, a* confluence classification *P is a set of sets of interactive transitions:*

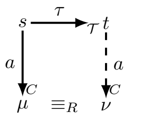
$$P = \{C_1, C_2, \ldots, C_n\} \subseteq \mathscr{P}(\hookrightarrow)$$

Given a set $\mathcal{T} \subseteq P$ (possibly P itself) of groups, we slightly abuse notation by writing $(s \xrightarrow{a} \mu) \in \mathcal{T}$ to denote that $(s \xrightarrow{a} \mu) \in C$ for some $C \in \mathcal{T}$.

Additionally, we use $s \xrightarrow{a}_{C_i} \mu$ to denote that $(s \xrightarrow{a} \mu) \in C_i$ and $s \xrightarrow{a}_{\mathcal{T}} \mu$ to denote that $(s \xrightarrow{a} \mu) \in \mathcal{T}$. Similarly, we subscript reachability, joinability and convertibility arrows (e.g., $s \twoheadrightarrow\!\twoheadleftarrow_{\mathcal{T}} t$) to indicate that they only traverse transitions from a certain group or set of groups of transitions.

We define confluence on such a classification: we designate a *set of groups* $\mathcal{T} \subseteq P$ to be confluent (now called *Markovian confluent*). As discussed in the previous section and just like in probabilistic partial order reduction [BDG06], only invisible transitions with a Dirac distribution are allowed to be confluent. (Still, prioritising such transitions may very well reduce probabilistic transitions as well, as we will see in Section 6.3.) For a set $\mathcal{T}$ to be Markovian confluent, it is therefore not allowed to contain any visible or probabilistic transitions.

Additionally, each transition $s \xrightarrow{a} \mu$ (allowing $a = \tau$) enabled together with a transition $s \xrightarrow{\tau}_{\mathcal{T}} t$ should have a mimicking transition $t \xrightarrow{a} \nu$ such that μ and ν are connected by $\mathcal{T}$-transitions (as discussed in Section 6.2.1). The previous requirement for confluent transitions to be mimicked by confluent transitions is strengthened: we require for each group in the classification that transitions from that group are mimicked by transitions from the same group. This turns out to be essential for the closure of confluence under unions. No restrictions are imposed on transitions that are not in any group, since they cannot be chosen to be confluent anyway.


(a) $(s \xrightarrow{a} \mu) \in P$.

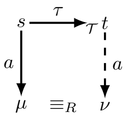

(b) $(s \xrightarrow{a} \mu) \notin P$.

Figure 6.6: The confluence diagrams for $s \xrightarrow{\tau}_{\mathcal{T}} t$. If the solid transitions are present, then so should the dashed ones be (such that $\mu \equiv_R \nu$).

All is formalised in the definition below, and illustrated in Figure 6.6.

Definition 6.7 (Markovian confluence). *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L\rangle$ be an MA and $P \subseteq \mathscr{P}(\hookrightarrow)$ a confluence classification. Then, a set $\mathcal{T} \subseteq P$ is* Markovian confluent for P *if it only contains sets of invisible transitions with Dirac distributions, and for all $s \xrightarrow{\tau}_{\mathcal{T}} t$ and all transitions $(s \xrightarrow{a} \mu) \neq (s \xrightarrow{\tau} t)$:*

$$\begin{cases} \forall C \in P \,.\, s \xrightarrow{a}_C \mu \implies \exists \nu \in \mathrm{Distr}(S) \,.\, t \xrightarrow{a}_C \nu \wedge \mu \equiv_{R^{\mathcal{T}}_{\mu,\nu}} \nu & \text{, if } (s \xrightarrow{a} \mu) \in P \\ \exists \nu \in \mathrm{Distr}(S) \,.\, t \xrightarrow{a} \nu \wedge \mu \equiv_{R^{\mathcal{T}}_{\mu,\nu}} \nu & \text{, if } (s \xrightarrow{a} \mu) \notin P \end{cases}$$

with $R^{\mathcal{T}}_{\mu,\nu}$ the smallest equivalence relation such that

$$R^{\mathcal{T}}_{\mu,\nu} \supseteq \{(s,t) \in \mathit{supp}(\mu) \times \mathit{supp}(\nu) \mid (s \xrightarrow{\tau} t) \in \mathcal{T}\}$$

A transition $s \xrightarrow{\tau} t$ is Markovian confluent *if there exists a Markovian confluent set $\mathcal{T}$ such that $s \xrightarrow{\tau}_{\mathcal{T}} t$. Often, we omit the adjective 'Markovian'.*

We remark several important aspects about this definition:

- Recall that a transition $s \xrightarrow{a} \mu$ is *invisible* if both $a = \tau$ and $L(s) = L(t)$ for every $t \in \mathit{supp}(\mu)$.
- Each set $\mathcal{T}$ and pair of transitions $s \xrightarrow{a} \mu$ and $t \xrightarrow{a} \nu$ yields a relation $R^{\mathcal{T}}_{\mu,\nu}$ to show that μ and ν are equivalent; this relation depends on $\mathcal{T}$, μ and ν. However, as it will always be clear from the context on which distributions this relation depends (and often on which set $\mathcal{T}$), we will from now on often omit the subscripts μ, ν and the superscript $\mathcal{T}$.
- For $\mu \equiv_{R^{\mathcal{T}}_{\mu,\nu}} \nu$, we require $\mathcal{T}$-transitions from the support of μ to the support of ν. Even though a (symmetric and transitive) equivalence relation is used, transitions from the support of ν to the support of μ do not influence $R^{\mathcal{T}}_{\mu,\nu}$, and neither do confluent paths from μ to ν of length more than one.
- Markovian transitions are irrelevant for confluence. After all, states having a τ-transition can never execute a Markovian transition due to the maximal progress assumption. Hence, if $s \xrightarrow{\tau} t$ and $s \xrightarrow{a} \mu$, then by definition of extended transitions $s \xrightarrow{a} \mu$ corresponds to an interactive transition $s \xhookrightarrow{a} \mu$.

Remark 6.8. Due to the confluence classification, confluent transitions are always mimicked by confluent transitions. After all, transitions from a group

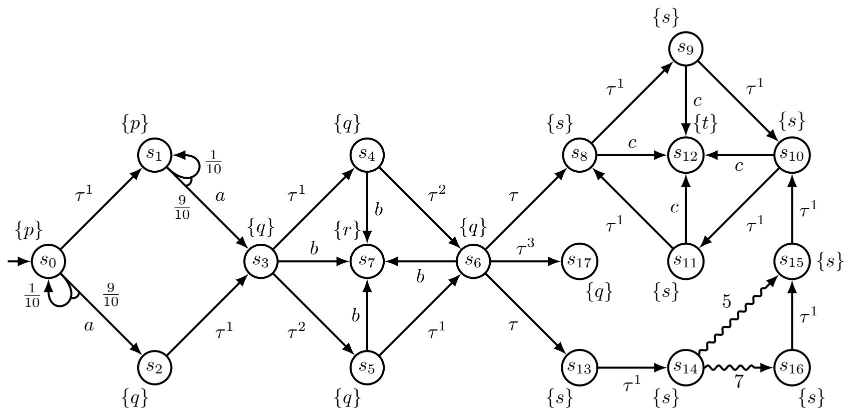


Figure 6.7: An MA $\mathcal{M}$.

$C \in P$ are mimicked by transitions from C. So, if C is designated confluent by $\mathcal{T}$, then all these confluent transitions are indeed mimicked by confluent transitions. □

Although the confluence classification may appear restrictive, we will see that in practice it is obtained naturally. Transitions are often instantiations of higher-level constructs, and are therefore easily grouped together. Then, it makes sense to detect the confluence of such a higher-level construct. Additionally, to show that a certain set of invisible transitions is confluent, we can just take P to consist of one group containing precisely all those transitions. Then, the requirement for P-transitions to be mimicked by the same group reduces to the old requirement that confluent transitions are mimicked by confluent transitions.

Example 6.9. Figure 6.7 provides an MA $\mathcal{M}$ with nondeterminism, probability, Markovian rates and state labels. It is used throughout this chapter as a running example to illustrate the various concepts related to confluence.

We depict a possible confluence classification P for $\mathcal{M}$ by adding superscripts to the τ-labels of some of the transitions (these are not part of the actual label). Hence, P contains three groups:

$$\begin{aligned}
C_1 &= \{(s_0,\tau,\mathbb{1}_{s_1}),(s_2,\tau,\mathbb{1}_{s_3}),(s_3,\tau,\mathbb{1}_{s_4}),(s_5,\tau,\mathbb{1}_{s_6}),(s_8,\tau,\mathbb{1}_{s_9}),(s_9,\tau,\mathbb{1}_{s_{10}}),\\
&\qquad (s_{10},\tau,\mathbb{1}_{s_{11}}),(s_{11},\tau,\mathbb{1}_{s_8}),(s_{13},\tau,\mathbb{1}_{s_{14}}),(s_{16},\tau,\mathbb{1}_{s_{15}}),(s_{15},\tau,\mathbb{1}_{s_{10}})\}\\
C_2 &= \{(s_3,\tau,\mathbb{1}_{s_5}),(s_4,\tau,\mathbb{1}_{s_6})\}\\
C_3 &= \{(s_6,\tau,\mathbb{1}_{s_{17}})\}
\end{aligned}$$

All transitions in P are labelled by τ, have a Dirac distribution and do not change the state labelling. Hence, they potentially may be confluent, if they additionally commute with all neighbouring transitions. Note that no other transitions can be confluent, as they all are either labelled by a visible action or change the state labelling. So, there is no use for them in P.

Now, we take $\mathcal{T} = \{C_1\}$ and show that it is a Markovian confluent set. We have to show that each transition in $\mathcal{T}$ is confluent as defined above. We illustrate this for a couple of transitions.

First, consider $s_0 \xrightarrow{\tau}_{\mathcal{T}} s_1$. There is one other transition from s_0, namely $s_0 \xrightarrow{a} \mu$ with $\mu(s_2) = \frac{9}{10}$ and $\mu(s_0) = \frac{1}{10}$. Since $s_0 \xrightarrow{a} \mu \notin P$, we need to show that $\exists \nu \in \mathrm{Distr}(S) \;.\; s_1 \xrightarrow{a} \nu \wedge \mu \equiv_R \nu$. We take $s_1 \xrightarrow{a} \nu$ with $\nu(s_3) = \frac{9}{10}$ and $\nu(s_1) = \frac{1}{10}$. This yields

$$R = \mathit{Id} \cup \{(s_0, s_1), (s_1, s_0), (s_2, s_3), (s_3, s_2)\}$$

with Id the identity relation. Indeed, μ and ν assign the same probability to each equivalence class of R, so $\mu \equiv_R \nu$.

Second, consider $s_2 \xrightarrow{\tau}_{\mathcal{T}} s_3$. Since there are no other transitions from s_2, there is nothing to check. The same holds for several other transitions in $\mathcal{T}$.

Finally, consider $s_3 \xrightarrow{\tau}_{\mathcal{T}} s_4$. There are two other transitions from s_3, namely $s_3 \xrightarrow{b} \mathbb{1}_{s_7}$ and $s_3 \xrightarrow{\tau} \mathbb{1}_{s_5}$. The first one can be mimicked by $s_4 \xrightarrow{b} \mathbb{1}_{s_7}$. Clearly $\mathbb{1}_{s_7} \equiv_R \mathbb{1}_{s_7}$, due to reflexivity. The second can be mimicked by $s_4 \xrightarrow{\tau} \mathbb{1}_{s_6}$. Then,

$$R = \mathit{Id} \cup \{(s_5, s_6), (s_6, s_5)\}$$

and hence indeed $\mathbb{1}_{s_5} \equiv_R \mathbb{1}_{s_6}$. Since $s_3 \xrightarrow{\tau} \mathbb{1}_{s_5} \in C_2$, we additionally need to check that also $s_4 \xrightarrow{\tau} \mathbb{1}_{s_6} \in C_2$. This is indeed the case.

The remaining transitions can be shown to satisfy the requirements in the same manner. $\square$

6.2.3 Properties of confluent sets

Since confluent transitions are always mimicked by confluent transitions, confluent paths (i.e., paths following only transitions from a confluent set) are always joinable. This is captured by the following proposition.

Proposition 6.10. *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L\rangle$ be an MA, $P \subseteq \mathscr{P}(\hookrightarrow)$ a confluence classification for $\mathcal{M}$ and $\mathcal{T}$ a Markovian confluent set for P. Then,*

$$s \twoheadrightarrow\twoheadleftarrow_{\mathcal{T}} t \qquad \textit{if and only if} \qquad s \twoheadleftarrow\!\!\twoheadrightarrow_{\mathcal{T}} t$$

Due to the confluence classification, we now also do have a closure result. Closure under union tells us that it is safe to show confluence of multiple sets of transitions in isolation, and then just take their union as one confluent set. Also, it implies that there exists a unique maximal confluent set.

Theorem 6.11. *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L\rangle$ be an MA, $P \subseteq \mathscr{P}(\hookrightarrow)$ a confluence classification for $\mathcal{M}$ and $\mathcal{T}_1, \mathcal{T}_2$ two Markovian confluent sets for P. Then, $\mathcal{T}_1 \cup \mathcal{T}_2$ is also a Markovian confluent set for P.*

Example 6.12. Example 6.9 demonstrated that $\mathcal{T} = \{C_1\}$ is confluent for our running example. In the same way, it can be shown that $\mathcal{T}' = \{C_2\}$ is confluent. Hence, $\mathcal{T}'' = \{C_1, C_2\}$ is also confluent. $\square$

The final result of this section shows that confluent transitions indeed connect divergence-sensitive branching bisimilar states. This is a key result; it implies that confluent transitions can be given priority over other transitions without losing behaviour—when being careful not to indefinitely ignore any behaviour.

Theorem 6.13. *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L \rangle$ be an MA, $s, s' \in S$ two of its states, $P \subseteq \mathscr{P}(\hookrightarrow)$ a confluence classification for $\mathcal{M}$ and $\mathcal{T}$ a Markovian confluent set for P. Then,*

$$s \twoheadleftarrow\!\!\twoheadrightarrow_{\mathcal{T}} s' \textit{ implies } s \approx_{\mathrm{b}}^{\mathrm{div}} s'$$

6.3 State space reduction using confluence

We can reduce state spaces by giving priority to confluent transitions, i.e., omitting all other transitions from a state that also enables a confluent transition (as long as no behaviour is ignored indefinitely). Better still, we aim at omitting all intermediate states on a confluent path altogether; after all, they are all bisimilar anyway by Theorem 6.13. Confluence even dictates that all visible transitions and divergences enabled from a state s can directly be mimicked from another state t if $s \twoheadrightarrow_{\mathcal{T}} t$. Hence, during state space generation we can just keep following a confluent path and only retain the last state. To avoid getting stuck in an infinite confluent loop, we detect entering a bottom strongly connected component (BSCC) of confluent transitions and choose a unique *representative* from this BSCC for all states that can reach it. This technique was proposed first in [Blo01], and later used in [BvdP02] for the non-probabilistic setting. A highly similar construction was used in [DJJL02] for representing sets of states for the so-called *essential state* abstraction for probabilistic transition systems.

Since confluent joinability is transitive (as implied by Proposition 6.10), it follows immediately that all confluent paths emanating from a certain state s always end up in a unique BSCC (as long as the system is finite).

6.3.1 Representation maps

Formally, we use the notion of a *representation map*, assigning a representative state $\varphi(s)$ to every state s. We make sure that $\varphi(s)$ indeed exhibits all behaviour of s due to being in a BSCC reachable from s.

Definition 6.14 (Representation map). *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L \rangle$ be an MA and $\mathcal{T}$ a Markovian confluent set for $\mathcal{M}$. Then, a function $\varphi_{\mathcal{T}} \colon S \to S$ is a* representation map *for $\mathcal{M}$ under $\mathcal{T}$ if for all $s, s' \in S$*

- $s \twoheadrightarrow_{\mathcal{T}} \varphi_{\mathcal{T}}(s)$;
- $s \to_{\mathcal{T}} s' \implies \varphi_{\mathcal{T}}(s) = \varphi_{\mathcal{T}}(s')$.

Note that the first requirement ensures that every representative is reachable from all states it represents, while the second takes care that all $\mathcal{T}$-related states have the same representative. Together, they imply that every representative is in a BSCC. Since all $\mathcal{T}$-related states have the same BSCC, as discussed above, it is indeed always possible to find a representation map. We refer to [Blo01]

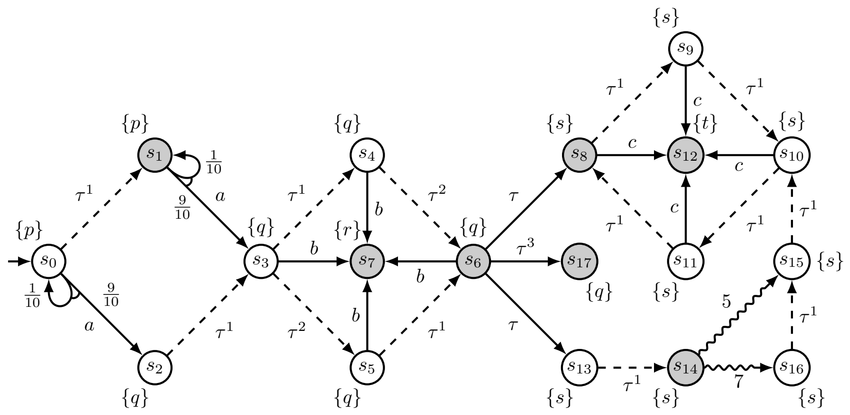


Figure 6.8: An MA $\mathcal{M}$ with confluent transitions and representatives.

for the algorithm we use to construct it in our implementation—a variation on Tarjan's algorithm for finding strongly connected components [Tar72].

Example 6.15. For our running example $\mathcal{M}$, we saw that $\mathcal{T} = \{C_1, C_2\}$ is confluent. A possible representation map for $\mathcal{M}$ under $\mathcal{T}$ is given by

$$
\begin{aligned}
\varphi(s_0) &= \varphi(s_1) = s_1 \\
\varphi(s_2) &= \varphi(s_3) = \varphi(s_4) = \varphi(s_5) = \varphi(s_6) = s_6 \\
\varphi(s_7) &= s_7 \\
\varphi(s_8) &= \varphi(s_9) = \varphi(s_{10}) = \varphi(s_{11}) = \varphi(s_{15}) = \varphi(s_{16}) = s_8 \\
\varphi(s_{12}) &= s_{12} \\
\varphi(s_{13}) &= \varphi(s_{14}) = s_{14} \\
\varphi(s_{17}) &= s_{17}
\end{aligned}
$$

Figure 6.8 shows our running example again, depicting all confluent transitions by dashed arrows and all representatives by means of a grey background.

Note that, indeed, each state can reach its representative via confluent transitions, and that confluently connected states all share the same representative. Due to these requirements, it is for instance not valid to use $\varphi(s_1) = s_0$ or $\varphi(s_0) = s_0$. □

Basically, the representation map is obtained by continuously following confluent transitions until ending up in either a state without any outgoing confluent transitions, or a cycle of confluent transitions.

6.3.2 Quotienting using a representation map

Since representatives exhibit all behaviour of the states they represent, they can be used for state space reduction. More precisely, it is possible to define the quotient of an MA modulo a representation map. This model does not have any

$\mathcal{T}$-transitions anymore, except for self-loops on representatives that have outgoing $\mathcal{T}$-transitions in the original system. These ensure preservation of divergences.

In the non-probabilistic case [BvdP02], these self-loops were not yet added and confluence reduction was not divergence sensitive. For MAs, omitting these self-loops could even erroneously enable Markovian transitions that were disabled in the presence of divergence due to the maximal progress assumption. Hence, we would not even preserve Markovian divergence-*in*sensitive branching bisimulation. Our current definition does not only make the theory work for MAs, it even yields preservation of divergence-*sensitive* branching bisimulation (and hence, according to our conjecture in Section 3.3.3, of minimal reachability probabilities as well).

Definition 6.16 (Quotient). *Given an MA $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L\rangle$, a confluent set $\mathcal{T}$ for $\mathcal{M}$, and a representation map $\varphi\colon S \to S$ for $\mathcal{M}$ under $\mathcal{T}$, the* quotient of $\mathcal{M}$ modulo φ *is the smallest system*

$$\mathcal{M}/\varphi = \langle \varphi(S), \varphi(s^0), A, \hookrightarrow_\varphi, \rightsquigarrow_\varphi, \mathrm{AP}, L_\varphi\rangle$$

such that

- $\varphi(S) = \{\varphi(s) \mid s \in S\}$;
- $\varphi(s) \xhookrightarrow{a}_\varphi \mu_\varphi$ *if* $\varphi(s) \xhookrightarrow{a} \mu$;
- $\varphi(s) \overset{\lambda}{\rightsquigarrow}_\varphi \varphi(s')$ *if* $\lambda = \sum_{\lambda' \in \Lambda(s,s')} \lambda'$ *and* $\lambda > 0$;
- $L_\varphi(\varphi(s)) = L(s)$ *for every* $\varphi(s) \in \varphi(S)$.

where $\Lambda(s, s')$ *is the multiset* $\{\!|\lambda' \in \mathbb{R} \mid \exists s^* \in S \,.\, \varphi(s) \overset{\lambda'}{\rightsquigarrow} s^* \wedge \varphi(s^*) = \varphi(s')|\!\}$.

Note that each interactive transition from $\varphi(s)$ in $\mathcal{M}$ is lifted to $\mathcal{M}/\varphi$ by changing all states in the support of its target distribution to their representatives. Additionally, each pair $\varphi(s), \varphi(s')$ of representative states in $\mathcal{M}/\varphi$ has a connecting Markovian transition with rate equal to the total outgoing rate of $\varphi(s)$ in $\mathcal{M}$ to states s^* that have $\varphi(s')$ as their representative (unless this sum is 0). It is easy to see that this implies $\varphi(s) \xrightarrow{\chi(\lambda)}_\varphi \mu_\varphi$ if and only if $\varphi(s) \xrightarrow{\chi(\lambda)} \mu$.

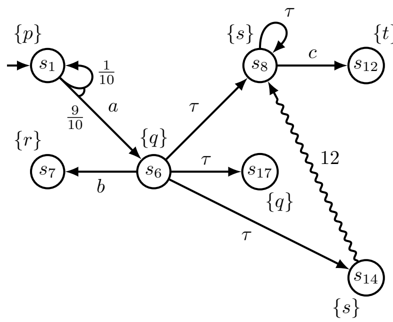


Figure 6.9: The quotient $\mathcal{M}/\varphi$ of $\mathcal{M}$ of the MA $\mathcal{M}$ from Figure 6.7.

Example 6.17. Based on the representation map given in Example 6.15, Figure 6.9 shows the quotient of our running example $\mathcal{M}$. The set of states is obtained easily by applying the function φ to all states of $\mathcal{M}$, resulting in $\varphi(S) = \{s_1, s_6, s_7, s_8, s_{12}, s_{14}, s_{17}\}$ (the grey states from Figure 6.8). The initial state of $\mathcal{M}/\varphi$ is the representative of s_0, which is s_1.

To understand the construction of $\hookrightarrow$, consider the original transition $s_1 \xhookrightarrow{a} \nu$ with $\nu(s_3) = \frac{9}{10}$ and $\nu(s_1) = \frac{1}{10}$. It yields the transition $s_1 \xhookrightarrow{a} \mu_\varphi$ in the quotient. Since $\varphi(s_3) = s_6$ and $\varphi(s_1) = s_1$, it follows that μ_φ is the distribution assigning probability $\frac{9}{10}$ to s_6 and $\frac{1}{10}$ to s_1. Note that, in the same way, the confluent transition from s_8 yields a self-loop due to the fact that $\varphi(s_9) = s_8$.

To understand the construction of $\rightsquigarrow$, we discuss the motivation for the transition $s_{14} \overset{12}{\rightsquigarrow} s_8$ in the quotient. Note that

$$\begin{aligned}\Lambda(s_{14}, s_8) &= \{\!|\lambda' \in \mathbb{R} \mid \exists s^* \in S \,.\, s_{14} \overset{\lambda'}{\rightsquigarrow} s^* \wedge \varphi(s^*) = s_8|\!\} \\ &= \{\!|5, 7|\!\}\end{aligned}$$

Hence, in $\mathcal{M}$ there is a total outgoing rate of $5 + 7 = 12$ from s_{14} to states having s_8 as their representative. Therefore, the quotient has a transition $s_{14} \overset{12}{\rightsquigarrow} s_8$. Note that the multiset is necessary due to the possibility of having several transitions with the same rate to states having the same representative. □

Since $\mathcal{T}$-transitions connect bisimilar states, and representatives exhibit all behaviour of the states they represent, we can prove the following theorem. It shows that we indeed reached our goal of providing a reduction that is safe with respect to divergence-sensitive branching bisimulation.

Theorem 6.18. *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L\rangle$ be an MA, $\mathcal{T}$ a Markovian confluent set for $\mathcal{M}$, and $\varphi\colon S \to S$ a representation map for $\mathcal{M}$ under $\mathcal{T}$. Then,*

$$\mathcal{M}/\varphi \approx_{\mathrm{b}}^{\mathrm{div}} \mathcal{M}$$

6.4 Symbolic detection of Markovian confluence

Although the MA-based definition of confluence in Section 6.2 is useful to show the correctness of our approach, it is often not feasible to check in practice. After all, we want to reduce *on-the-fly* to obtain a smaller state space without first generating the unreduced one. Therefore, we propose a set of heuristics to detect Markovian confluence in the context of the process-algebraic modelling language MAPA (as defined in Chapter 4).

Since every MAPA specification can efficiently be linearised to an MLPE, it suffices to define our heuristics only on this subset of the language. For convenience, we restate the definition of the MLPE here.

Definition 4.26 (MLPEs). *A* Markovian linear process equation *(MLPE) is a MAPA specification of the following format:*

$$X(\boldsymbol{g} : \boldsymbol{G}) = \sum_{i \in I} \sum_{\boldsymbol{d_i} : \boldsymbol{D_i}} c_i \Rightarrow a_i(\boldsymbol{b_i}) \sum_{\boldsymbol{e_i} : \boldsymbol{E_i}} f_i : X(\boldsymbol{n_i})$$

$$+ \sum_{j \in J} \sum_{\boldsymbol{d_j} : \boldsymbol{D_j}} c_j \Rightarrow (\lambda_j) \cdot X(\boldsymbol{n_j})$$

The first $|I|$ *nondeterministic choices are referred to as* interactive summands*, the last* $|J|$ *as* Markovian summands*.*

Recall that the expressions c_i, $\boldsymbol{b_i}$, f_i and $\boldsymbol{n_i}$ may depend on $\boldsymbol{g}$ and $\boldsymbol{d_i}$, and f_i and $\boldsymbol{n_i}$ also on $\boldsymbol{e_i}$. Similarly, c_j, λ_j and $\boldsymbol{n_j}$ may depend on $\boldsymbol{g}$ and $\boldsymbol{d_j}$. Often, we use the shorthand notation $c_i(\boldsymbol{g'}, \boldsymbol{d'_i})$ for $c_i[(\boldsymbol{g}, \boldsymbol{d_i}) := (\boldsymbol{g'}, \boldsymbol{d'_i})]$, and similar notations for the other expressions.

Confluent summands. As discussed in Chapter 4, each interactive summand of an MLPE yields a set of interactive transitions, whereas each Markovian summand yields a set of Markovian transitions. Instead of detecting *individual* confluent transitions, we detect *confluent summands.* A summand is considered confluent if the set of *all* transitions it may generate (according to the operational semantics as discussed in Section 4.2.4) is guaranteed to be confluent. Since only interactive transitions can be confluent, only interactive summands can be confluent.

Hence, we assume an implicit confluence classification $P = \{C_1, C_2, \ldots, C_k\}$ that, for each interactive summand $i \in I = \{1, \ldots, k\}$, contains a group C_i consisting of precisely all transitions generated by that summand. For each interactive summand i we try to show that the set $\mathcal{T} = \{C_i\}$ is confluent. Then, by Theorem 6.11, the set of transitions generated by all confluent summands together is also confluent. Whenever during state space generation a confluent summand is enabled, all other summands can be ignored (continuing until reaching a representative in a BSCC, as explained in the previous section).

6.4.1 Characterisation of confluent summands

Let C_i be the set of all transitions generated by a summand i. By Definition 6.7, for confluence we need to check for every transition $s \xrightarrow{a'} \mu'$ in C_i whether

1. The action is invisible: $a' = \tau$.
2. The probability distribution is Dirac: $\mu' = \mathbb{1}_t$ for some state t.
3. For every transition $s \xrightarrow{a} \mu$ different from $s \xrightarrow{\tau} \mathbb{1}_t$ it holds that

$$\begin{cases} \forall C \in P \,.\, s \xrightarrow{a}_C \mu \implies \exists \nu \in \mathrm{Distr}(S) \,.\, t \xrightarrow{a}_C \nu \wedge \mu \equiv_R \nu & \text{,if } (s \xrightarrow{a} \mu) \in P \\ \qquad\qquad\qquad\qquad\quad\; \exists \nu \in \mathrm{Distr}(S) \,.\, t \xrightarrow{a} \nu \;\; \wedge \mu \equiv_R \nu & \text{,if } (s \xrightarrow{a} \mu) \notin P \end{cases}$$

   with R the smallest equivalence relation such that

$$R \supseteq \{(s,t) \in \mathit{supp}(\mu) \times \mathit{supp}(\nu) \mid (s \xrightarrow{\tau} t) \in \mathcal{T}\}$$

4. The transition is stuttering: $L(s) = L(t)$.

Since we want to reduce *prior to* the generation of the full state space, we cannot first construct the set C_i and check if it satisfies all these conditions. Rather, we overapproximate these requirements by checking symbolically if summand i is such that all transitions it may ever generate satisfy the conditions. In this section, we characterise confluent summands, by explaining how these four conditions can be derived from their specification. Examples will be provided in the next section, when we discuss heuristics for checking these characterisations practically.

1. *Action-invisible summands.* We can easily check if all transitions that can be generated by a summand have an invisible action: the summand should just have τ as its action.

Definition 6.19 (Action-invisible summands). *A summand i is* action-invisible *if $a_i(\boldsymbol{b_i}) = \tau$.*

It is immediately clear from the operational semantics of the MLPE that action-invisible summands fulfill the first requirement of confluence.

2. *Non-probabilistic summands.* To check if a summand can only yield Dirac distributions, there should be only one possible next state for each valuation of the global and local parameters that enables its condition.

Definition 6.20 (Non-probabilistic summands). *A summand i is* non-probabilistic *if for every $\boldsymbol{v} \in \boldsymbol{G}$ and $\boldsymbol{d'_i} \in \boldsymbol{D_i}$ such that $c_i(\boldsymbol{v}, \boldsymbol{d'_i})$, there exists a unique state $\boldsymbol{v'} \in \boldsymbol{G}$ such that $\boldsymbol{n_i}(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e'_i}) = \boldsymbol{v'}$ for every $\boldsymbol{e'_i} \in \boldsymbol{E_i}$*[4]*.*

It should also be immediately clear from the operational semantics of the MLPE that non-probabilistic summands fulfill the second requirement of confluence.

3. *Commuting summands.* For a summand i to satisfy the third condition, we need to check if every transition $s \xrightarrow{a} \mu$ enabled together with a transition $s \xrightarrow{\tau} \mathbb{1}_t$ generated by i can be mimicked from state t. Additionally, the mimicking transition $t \xrightarrow{a} \nu$ should be from the same group as $s \xrightarrow{a} \mu$. Since each group precisely consists of all transitions generated by one summand, this boils down to the requirement that if $s \xrightarrow{a} \mu$ is generated by summand j, then so should $t \xrightarrow{a} \nu$ be.

To check if all transitions generated by a summand i commute in this way with all other transitions, we check for each summand j if *all* transitions that it may generate commute with *all* transitions that summand i may generate. In that case, we say that the two summands *commute.* Again, we look at the specification of the summands and not at the actual transitions they generate.

Clearly, two summands i, j commute if they cannot disable each other and do not influence each other's action parameters, probabilities and next states. After all, in that case any transition $s \xrightarrow{a} \mu$ enabled from state s by summand j

[4]We could also weaken this condition slightly to

$$\forall \boldsymbol{e'_i} \in \boldsymbol{E_i} \cdot f_i(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e'_i}) > 0 \implies \boldsymbol{n_i}(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e'_i}) = \boldsymbol{v'}$$

However, in that case we would need to be more strict later on, requiring the probabilities of a confluent summand to remain invariant. For the current formulation, the probability expression can be changed without influencing the next state.

must still be enabled from state t as a transition $t \xrightarrow{a} \nu$, the τ-transition from summand i is still enabled in all states in the support of μ and the order of the execution of the two transitions does not influence the observable behaviour or the final state. Hence, $\mu \equiv_R \nu$. Since Definition 6.7 does not require transitions to commute with themselves, a summand can also commute with itself by generating only one transition per state.

To formally define commuting summands, we already assume that one of them is action-invisible and non-probabilistic. Hence, we can write $\boldsymbol{n_i}(\boldsymbol{v}, \boldsymbol{d'_i})$ for its unique next state given $\boldsymbol{v}$ and $\boldsymbol{d'_i}$. For an action-invisible non-probabilistic summand i to commute with a summand j, we have to investigate their behaviour for all state vectors $\boldsymbol{v} \in \boldsymbol{G}$ and local variable vectors $\boldsymbol{d'_i} \in \boldsymbol{D_i}, \boldsymbol{d'_j} \in \boldsymbol{D_j}$ such that both summands are enabled: $c_i(\boldsymbol{v}, \boldsymbol{d'_i}) \wedge c_j(\boldsymbol{v}, \boldsymbol{d'_j})$. The maximal progress assumption dictates that interactive summands and Markovian summands can never be enabled at the same time, so we only need to check commutativity among the interactive summands.

Definition 6.21 (Commuting summands). *An action-invisible non-probabilistic summand i* commutes *with a summand j (possibly $i = j$) if for all $\boldsymbol{v} \in \boldsymbol{G}$, $\boldsymbol{d'_i} \in \boldsymbol{D_i}, \boldsymbol{d'_j} \in \boldsymbol{D_j}$ such that $c_i(\boldsymbol{v}, \boldsymbol{d'_i}) \wedge c_j(\boldsymbol{v}, \boldsymbol{d'_j})$:*

- *Summand i cannot not disable summand j, and vice versa:*

$$c_j(\boldsymbol{n_i}(\boldsymbol{v}, \boldsymbol{d'_i}), \boldsymbol{d'_j}) \wedge c_i(\boldsymbol{n_j}(\boldsymbol{v}, \boldsymbol{d'_j}, \boldsymbol{e'_j}), \boldsymbol{d'_i})$$

  for every $\boldsymbol{e'_j} \in \boldsymbol{E_j}$ such that $f_j(\boldsymbol{v}, \boldsymbol{d'_j}, \boldsymbol{e'_j}) > 0$.
- *Summand i cannot influence the action parameters of summand j:*

$$b_j(\boldsymbol{v}, \boldsymbol{d'_j}) = b_j(\boldsymbol{n_i}(\boldsymbol{v}, \boldsymbol{d'_i}), \boldsymbol{d'_j})$$

  Note that summand i was already assumed to have no action parameters (as its action is always τ), so these cannot be influenced by j anyway—hence, no converse equality has to be checked.
- *Summand i cannot influence the probability expression of summand j:*

$$f_j(\boldsymbol{v}, \boldsymbol{d'_j}, \boldsymbol{e'_j}) = f_j(\boldsymbol{n_i}(\boldsymbol{v}, \boldsymbol{d'_i}), \boldsymbol{d'_j}, \boldsymbol{e'_j})$$

  for every $\boldsymbol{e'_j} \in \boldsymbol{E_j}$.

  Note that summand i was already assumed to have a unique next state $\boldsymbol{v'} \in \boldsymbol{G}$ for any state vector $\boldsymbol{v} \in \boldsymbol{G}$ and local variable vector $\boldsymbol{d'_i} \in \boldsymbol{D_i}$, so no converse equality has to be checked. (Under the adapted condition for potentially confluent summands discussed in Footnote 4 on page 147, this would not be the case anymore.)
- *Execution of summands i and j in either order yield the same next state:*

$$\boldsymbol{n_j}(\boldsymbol{n_i}(\boldsymbol{v}, \boldsymbol{d'_i}), \boldsymbol{d'_j}, \boldsymbol{e'_j}) = \boldsymbol{n_i}(\boldsymbol{n_j}(\boldsymbol{v}, \boldsymbol{d'_j}, \boldsymbol{e'_j}), \boldsymbol{d'_i})$$

  for every $\boldsymbol{e'_j} \in \boldsymbol{E_j}$ such that $f_j(\boldsymbol{v}, \boldsymbol{d'_j}, \boldsymbol{e'_j}) > 0$.

or, alternatively, if $i = j \wedge \boldsymbol{n_i}(\boldsymbol{v}, \boldsymbol{d'_i}) = \boldsymbol{n_j}(\boldsymbol{v}, \boldsymbol{d'_j})$.

From the discussion above, it follows that if i commutes with all summands including itself, it satisfies the third requirement of confluence.

Remark 6.22. Definition 6.21 can be phrased as one formula, characterising the commutativity of two summands. An action-invisible and non-probabilistic summand i commutes with a summand j (possibly $i = j$) if:

$$c_i(\boldsymbol{v}, \boldsymbol{d_i'}) \wedge c_j(\boldsymbol{v}, \boldsymbol{d_j'})$$
$$\Longrightarrow$$
$$\left(\begin{array}{ll} & f_j(\boldsymbol{v}, \boldsymbol{d_j'}, \boldsymbol{e_j'}) > 0 \implies c_j(\boldsymbol{n_i}(\boldsymbol{v}, \boldsymbol{d_i'}), \boldsymbol{d_j'}) \wedge c_i(\boldsymbol{n_j}(\boldsymbol{v}, \boldsymbol{d_j'}, \boldsymbol{e_j'}), \boldsymbol{d_i'}) \\ \wedge & b_j(\boldsymbol{v}, \boldsymbol{d_j'}) = b_j(\boldsymbol{n_i}(\boldsymbol{v}, \boldsymbol{d_i'}), \boldsymbol{d_j'}) \\ \wedge & f_j(\boldsymbol{v}, \boldsymbol{d_j'}, \boldsymbol{e_j'}) = f_j(\boldsymbol{n_i}(\boldsymbol{v}, \boldsymbol{d_i'}), \boldsymbol{d_j'}, \boldsymbol{e_j'}) \\ \wedge & f_j(\boldsymbol{v}, \boldsymbol{d_j'}, \boldsymbol{e_j'}) > 0 \implies \boldsymbol{n_j}(\boldsymbol{n_i}(\boldsymbol{v}, \boldsymbol{d_i'}), \boldsymbol{d_j'}, \boldsymbol{e_j'}) = \boldsymbol{n_i}(\boldsymbol{n_j}(\boldsymbol{v}, \boldsymbol{d_j'}, \boldsymbol{e_j'}), \boldsymbol{d_i'}) \end{array}\right)$$
$$\vee \big(i = j \wedge \boldsymbol{n_i}(\boldsymbol{v}, \boldsymbol{d_i'}) = \boldsymbol{n_j}(\boldsymbol{v}, \boldsymbol{d_j'})\big)$$

where $\boldsymbol{v}, \boldsymbol{d_i'}, \boldsymbol{d_j'}$ and $\boldsymbol{e_j'}$ universally quantify over $\boldsymbol{G}$, $\boldsymbol{D_i}$, $\boldsymbol{D_j}$ and $\boldsymbol{E_j}$, respectively. □

As these formulas are quantifier-free and in practice often either trivially false or true, they can generally be solved using an SMT solver for the data types involved. For n summands, n^2 formulas need to be solved; the complexity of this procedure depends on the data types.

4. *Stuttering summands.* Finally, a confluence summand should only generate transitions that do not change change the state labelling. Hence, for all $\boldsymbol{v} \in \boldsymbol{G}$ and $\boldsymbol{d_i'} \in \boldsymbol{D_i}$:

$$c_i(\boldsymbol{v}, \boldsymbol{d_i'}) \implies L(\boldsymbol{v}) = L(\boldsymbol{n_i}(\boldsymbol{v}, \boldsymbol{d_i'}))$$

Recall that we defined the state labelling of the MA corresponding to a MAPA specification such that each state is labelled by the set of visible actions it immediately enables (Definition 4.22). Hence, for a summand to be invisible with respect to the state labelling it should leave invariant the set of enabled visible actions. This requirement can be alleviated by hiding all actions that are not used in the properties of interest.

If a summand i is action-invisible and commutes with all summands, we already know that it can never disable another summand (if it disables itself that is fine, since it cannot produce any visible actions). Hence, we only still need to verify whether it can never *enable* a summand having a visible action.

Definition 6.23 (Stuttering summands). *An action-invisible non-probabilistic summand i that commutes with all summands is* stuttering *if, for each summand j,*

- *The condition of j can never be enabled by i if j has a visible action, i.e.,*

$$c_i(\boldsymbol{v}, \boldsymbol{d_i'}) \wedge a_j \neq \tau \wedge \neg c_j(\boldsymbol{v}, \boldsymbol{d_j'}) \implies \neg c_j(\boldsymbol{n_i}(\boldsymbol{v}, \boldsymbol{d_i'}), \boldsymbol{d_j'})$$

for all $\boldsymbol{v} \in \boldsymbol{G}$, $\boldsymbol{d_i'} \in \boldsymbol{D_i}$ and $\boldsymbol{d_j'} \in \boldsymbol{D_j}$.

Clearly, stuttering summands satisfy the fourth requirement of confluence.

Concluding, a summand that is action-invisible, non-probabilistic and stuttering, and that also commutes with all summands including itself, generates only confluent transitions.

6.4.2 Heuristics for confluent summands

Although the symbolic characterisation introduced in the previous section is sound, it may not always be feasible or efficient to apply. Only action invisibility can easily be checked syntactically, so we introduce heuristics for checking if a summand is non-probabilistic, commuting and stuttering. Although our heuristics are rather simple and intuitive, they are widely applicable (as demonstrated by our case studies in Chapter 9).

2. *Non-probabilistic summands.* In Definition 6.20 we discussed a minimal condition for a summand i to be non-probabilistic. We check this heuristically in the following way.

P. Instead of checking for all possible valuations of the global and local variables whether indeed $\boldsymbol{n_i}(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e'_i}) = \boldsymbol{v}'$ for every $\boldsymbol{e'_i} \in \boldsymbol{E_i}$, we just check if $|\boldsymbol{E_i}| = 1$. After all, summands that do have a probabilistic choice are very likely to indeed have different next states—otherwise, this choice could easily be eliminated. On the other hand, if there is no probabilistic choice, indeed there is a unique next state for every $\boldsymbol{v} \in \boldsymbol{G}$ and $\boldsymbol{d'_i} \in \boldsymbol{D_i}$.

3. *Commuting summands.* To check for commutativity according to Definition 6.21, we define several heuristics. As mentioned before, these are only needed for the interactive summands; all Markovian summands are trivially disabled due to the maximal progress assumption. Each heuristic individually is a sufficient condition for the requirements of Definition 6.21. Hence, only one of them has to hold. They are based on (1) two summands never being enabled at the same time, (2) two summands coinciding and not having any local nondeterministic choice, and (3) two summands never influencing each other's conditions, action parameters, probabilities and next states due to disjoint variable use.

C_1. *Never enabled at the same time.* If two summands are never enabled at the same time, i.e., $c_i(\boldsymbol{v}, \boldsymbol{d'_i}) \wedge c_j(\boldsymbol{v}, \boldsymbol{d'_j})$ does not hold for any possible combination of valuations $\boldsymbol{v} \in \boldsymbol{G}$, $\boldsymbol{d'_i} \in \boldsymbol{D_i}$ and $\boldsymbol{d'_j} \in \boldsymbol{D_j}$, then the commutativity requirements hold vacuously. Since the variables $\boldsymbol{d'_i}$ and $\boldsymbol{d'_j}$ are only used locally per summand (and hence cannot disable another summand), we focus on the global variables.

Our heuristics range over the global variables $\boldsymbol{g}$ that have a finite domain, and check if there is at least one g_k that disables either i or j for each of its valuations. That is, we check if

$$\forall v_k \in G_k \;.\; \neg c_i[g_k := v_k] \vee \neg c_j[g_k := v_k]$$

applying the expression simplification heuristics described in Section 4.5 to try to rewrite the conditions to `false`. If this indeed succeeds for all

$v_k \in G_k$ for some global variable g_k, then apparently $c_i(\boldsymbol{v}, \boldsymbol{d'_i}) \wedge c_j(\boldsymbol{v}, \boldsymbol{d'_j})$ can never occur. Hence, the commutativity requirements hold due to the fact that their premise never holds.

Example 6.24. Consider the following MLPE:

$$\begin{aligned} X(pc : \{1, \ldots, 10\}, x : \{1, 2, 3, 4\}) = \\ pc = 3 \wedge x < 3 \Rightarrow \tau \cdot X(4, 2) && (1) \\ + \ pc = 3 \wedge x \geq 3 \Rightarrow \tau \cdot X(5, x) && (2) \\ + \ \ldots \end{aligned}$$

Clearly, these two summands commute. To see why, note that for every value of x either c_i or c_j can be rewritten to `false`. Hence, since each state vector $\boldsymbol{v} \in \boldsymbol{G}$ must contain one of these values, the two summands are never enabled at the same time and thus $c_i(\boldsymbol{v}, \boldsymbol{d'_i}) \wedge c_j(\boldsymbol{v}, \boldsymbol{d'_j})$ can never hold. □

C_2. *Coinciding summands without local choice.* If the two summands for which commutativity is analysed coincide (i.e., $i = j$), we can satisfy the requirements by showing that $\boldsymbol{n_i}(\boldsymbol{v}, \boldsymbol{d'_i}) = \boldsymbol{n_j}(\boldsymbol{v}, \boldsymbol{d'_j})$ for every $\boldsymbol{v} \in \boldsymbol{G}$ and all $\boldsymbol{d'_i}, \boldsymbol{d'_j} \in \boldsymbol{D_i}$. We do so by checking whether $|\boldsymbol{D_i}| = 1$. Since this implies that $\boldsymbol{d'_i} = \boldsymbol{d'_j}$, the condition reduces to $\boldsymbol{n_i}(\boldsymbol{v}, \boldsymbol{d'_i}) = \boldsymbol{n_i}(\boldsymbol{v}, \boldsymbol{d'_i})$, with $\boldsymbol{d'_i}$ the single element of $\boldsymbol{D_i}$. Obviously, this holds for all $\boldsymbol{v} \in \boldsymbol{G}$.

Example 6.25. Consider the following MLPE:

$$\begin{aligned} X(x : \mathbb{N}, y : \mathbb{N}, z : \mathbb{N}) = \\ \textstyle\sum_{i:\{1,2\}} 5 < x < 10 \Rightarrow \tau \cdot X(x + i, y + 1) && (1) \\ + \ y > 2 \Rightarrow \tau \cdot X(x, y - 1) && (2) \end{aligned}$$

The second summand clearly commutes with itself, since it does not contain a nondeterministic choice and hence only produces at most one transition in each state. The first summand, though, may produce two transitions. Hence, it is not immediately clear anymore if these commute. □

C_3. *Disjoint variables.* If the global variables that are used by summand i (according to Definition 5.12) are disjoint from the global variables that are changed by summand j (according to Definition 5.1), and the global variables that are used by summand j are disjoint from the global variables that are changed by summand i, then clearly the order of their execution does not matter. For instance, $b_j(\boldsymbol{v}, \boldsymbol{d'_j}) = b_j(\boldsymbol{n_i}(\boldsymbol{v}, \boldsymbol{d'_i}), \boldsymbol{d'_j})$ trivially holds since all variables needed to decide $\boldsymbol{b_j}$ are used in j and hence unchanged by i; i.e., they do not differ between $\boldsymbol{v}$ and $\boldsymbol{n_i}(\boldsymbol{v}, \boldsymbol{d'_i})$. The other requirements hold by a similar argument.

We implemented some additional heuristics that for instance take into account that an update $x := x + 1$ cannot disable a condition $x > 5$, and that updates like $x := x + 2$, $x := x + 5$ and $x := x - 3$ commute due to their linearity. More precisely, our heuristics that check for disjoint variables verify whether for every variable g_k that is changed in i, the following four conditions hold:

(a) g_k does not occur in $\boldsymbol{b_j}$;
(b) g_k does not occur in f_j;
(c) g_k does not occur in c_j, or g_k is increased by i and only used in c_j in an expression of the form $g_k \geq e$ or $g_k > e$ such that g_k does not occur in e (possibly occurring within a conjunction or disjunction, but not in any other function);
(d) g_k is not used in $\boldsymbol{n_j}$, or g_k is only used in $\boldsymbol{n_j}$ to update itself and both $\boldsymbol{n_i}$ and $\boldsymbol{n_j}$ change g_k by adding or subtracting a concrete value.

For all variables changed in j, the symmetric conditions are checked. Note that for condition (a) this holds by definition, since $\boldsymbol{b_i}$ was already assumed to be empty due to the summand being action-invisible. Also, the symmetric counterpart of condition (b) can be omitted, since we also already assumed that $|\boldsymbol{E_i}| = 1$ by heuristic P. Therefore, f_i must be 1 for all possible valuations of its free variables and hence these valuations are irrelevant.

Example 6.26. Consider the following MLPE:

$$\begin{aligned} & X(x : \mathbb{N}, y : \mathbb{N}, z : \mathbb{N}) = \\ & \qquad 5 < x < 10 \Rightarrow \tau \cdot X(x+1, y+1, z) && (1) \\ & \quad + \; y > 2 \Rightarrow a \textstyle\sum_{i:\{1,2\}} \frac{1}{2} : X(x, y-1, i) && (2) \end{aligned}$$

Our heuristics are able to detect that the two summands of this MLPE commute. To see why, note that summand 1 changes variables x and y, while summand 2 changes variables y and z.

None of the changed variables occur as action parameters in the other summand, so condition (a) is satisfied. Also, the probability $\frac{1}{2}$ of the second summand does not rely on any of the variables changed in the first summand, so also condition (b) is satisfied easily.

For condition (c), observe that y and z (the variables changed in summand 2) are not used in the condition of summand 1. Also, x is not used in the condition of summand 2, so it being changed in summand 1 is fine. Finally, variable y is changed in the first summand and used in the second. This is allowed, though, since the change is an increase that can never disable the condition[5].

For condition (d), first note that x is not used in $\boldsymbol{n_2}$ and z is not used in $\boldsymbol{n_1}$. The variable y is changed in both and used in both, but only to update itself in a linear way. Hence, the condition is satisfied. Indeed, the order of execution of the summands does not influence the final value of y, since $(y-1)+1 = (y+1)-1 = y$. □

[5]It may enable it, though; hence, unless the action a is hidden, the requirement discussed next is actually not satisfied.

4. *Stuttering summands.* Definition 6.23 presented a sufficient condition for the transitions of a summand i to never change the state labelling. We check it heuristically in the following manner, for each summand j:

S. If $a_j = \tau$, the condition already holds vacuously[6]. Otherwise, we check whether summand i can never enable summand j. This is guaranteed if for every variable g_k that is changed in i, either

(a) g_k does not occur in c_j, or
(b) $n_{i,k}$ is a constant value and $c_j[g_k := n_{i,k}]$ can be evaluated to `false`.

If a summand satisfies all heuristics, then by the reasoning above and the characterisation from the previous section, all its transitions can safely be considered confluent.

We conclude this section with an example illustrating our heuristic detection of confluence.

Example 6.27. Consider the following parallel MAPA specification:

$$X = \tau \cdot b \cdot X \qquad\qquad Y = a \cdot c \cdot Y$$
$$\mathit{System} = \partial_{\{b,c\}}(X \,||\, Y)$$

Assuming $\gamma(b,c) = d$, linearisation yields

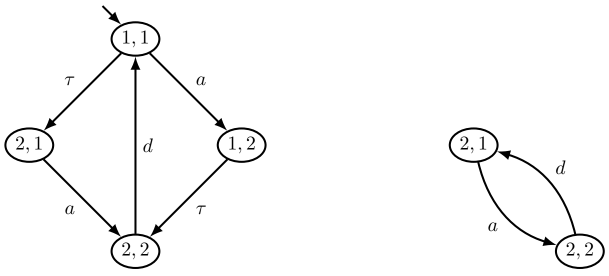
$$\begin{aligned} &Z(pc_1 : \{1,2\}, pc_2 : \{1,2\}) = \\ &\qquad pc_1 = 1 \Rightarrow \tau \cdot Z(2, pc_2) && (1)\\ &\quad + pc_2 = 1 \Rightarrow a \cdot Z(pc_1, 2) && (2)\\ &\quad + pc_1 = 2 \wedge pc_2 = 2 \Rightarrow d \cdot Z(1,1) && (3) \end{aligned}$$

with initial state $Z(1,1)$. Assume that we are only interested in observing the a-action; i.e., the state labelling is either $\varnothing$ (for every state that does not enable a) or $\{a\}$ (for every state that does enable a).

Due to the visible actions, only the first summand may be confluent. Heuristic P is satisfied by the first summand, since it indeed does not have a probabilistic choice.

The first summand commutes with itself, due to the absence of a nondeterministic local choice (I_2). It also commutes with the second summand, since there is no overlap between the variables changed by summand 1 (pc_1) and the variables used by summand 2 (pc_2) and vice versa (I_3). Finally, the first summand commutes with the third summand, as they can never be enabled at the same time (I_1). This is immediate from the fact that each possible value of pc_1 either disables the first summand or the third.

[6]If it is already known during state space generation which actions A' will be used for model checking, we can already consider the state labelling invariant if the enabledness of these actions is invariant. Hence, we can already consider the condition satisfied if $a_j \notin A'$. This is indeed what happens in our implementation. Even better, irrelevant actions can be hidden; this additionally makes them candidates to be confluent.



(a) The unreduced state space. (b) The reduced state space.

Figure 6.10: State space generation using confluence.

Finally, for heuristic S we still need to check whether the first summand cannot enable another summand having a visible action. Since only a is of interest, we just need to consider the second summand. Indeed, the first summand does not change any variables that are used in the condition of the second summand, so also S is satisfied.

Now, this knowledge can be used for reduction during state space generation. Whenever the confluent summand is enabled in some state it is taken immediately, and the intermediate state is not stored. Figure 6.10 demonstrates the state spaces obtained without and with application of the confluence information. □

Remark 6.28. To detect confluence, we range over all summands. For each summand, we check if it has label τ and no real probabilistic choice. Additionally, we check if it commutes with all other summands. In total, we have to check for each *pair* of summands if their conditions are mutually exclusive, if they coincide or if their variables are disjoint. Hence, in principle the complexity of confluence checking is in $O(n^2)$, with n the size of the MLPE.

Actually, our implementation tries to detect mutual exclusion of conditions by ranging over all possible values of the parameters with an enumerated type (such as $\{1, \ldots, 10\}$) and seeing if indeed for none of these values both conditions are enabled. Hence, this brings the worst-case complexity to $O(n^2 + k \cdot |I|^2)$, with k the total number of enumerated data values of all MLPE parameter together, and $|I|$ the number of summands. □

6.5 Failing alternatives

In the previous sections we introduced a notion of confluence for MAs, showing that confluent transitions connect divergence-sensitive branching bisimilar states. While developing the theory, we investigated the viability of several variations on the definitions. In this section we discuss some of these variations, explaining why they do not serve our purpose.

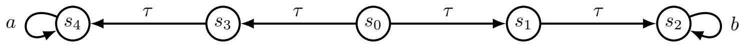


Figure 6.11: A counterexample for convertible confluent connectivity.

6.5.1 Convertible confluent connectivity

Given a confluent transition $s \xrightarrow{\tau}_{\mathcal{T}} t$, we require the distributions μ and ν of a transition $s \xrightarrow{a} \mu$ and its mimicking transition $t \xrightarrow{a} \nu$ to assign equal probabilities to each equivalence class of the relation

$$R \supseteq \{(s,t) \in supp(\mu) \times supp(\nu) \mid (s \xrightarrow{\tau} t) \in \mathcal{T}\}$$

Based on the intuition that confluent transitions connect bisimilar states, it may seem possible to relax this definition by using the larger relation

$$R = \{(s,t) \in supp(\mu) \times supp(\nu) \mid s \twoheadleftarrow\!\!\twoheadrightarrow_{\mathcal{T}} t\}$$

To see why this would be incorrect, consider the system depicted in Figure 6.11. Consider the confluence classification $\mathcal{P} = \{C\}$ with C the set of all τ-transitions in this model, and let $\mathcal{T} = \{C\}$. According to the relaxed definition of confluence, this set $\mathcal{T}$ would indeed be confluent. Since only s_0 has more than one outgoing transition, we only have to check whether they commute. Hence, we need to show for $s_0 \xrightarrow{\tau}_{\mathcal{T}} s_1$ that $s_0 \xrightarrow{\tau} s_3$ can be mimicked from s_1, and the other way around. Indeed, $s_1 \xrightarrow{\tau} s_2$ and $s_3 \twoheadleftarrow\!\!\twoheadrightarrow_{\mathcal{T}} s_2$, and $s_3 \xrightarrow{\tau} s_4$ and $s_1 \twoheadleftarrow\!\!\twoheadrightarrow_{\mathcal{T}} s_4$.

Still, it would clearly be incorrect to give one of these confluent transitions priority over the other—this would disable one of the visible transitions labelled a and b. The reason for this is that the relaxed definition allows the transitions to be mimicked to be used for the convertibility path. However, as in the end it is abstracted away, this path does not exist anymore in the reduced system.

6.5.2 Joinable confluent connectivity

To solve the problem discussed in the previous section, we may be tempted to use joinability instead of convertibility, i.e., consider

$$R = \{(s,t) \in supp(\mu) \times supp(\nu) \mid s \twoheadrightarrow\twoheadleftarrow_{\mathcal{T}} t\}$$

Indeed, this strengthens the definition and would not consider the transitions in Figure 6.11 emanating from s_0 to be confluent anymore. However, consider

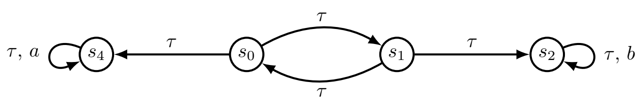


Figure 6.12: A counterexample for joinable confluent connectivity.

the system in Figure 6.12. Again, consider the confluence classification $\mathcal{P} = \{C\}$ with C the set of all τ-transitions in this model, and let $\mathcal{T} = \{C\}$. It is not hard to verify that all conditions for confluence are satisfied. For instance, for $s_0 \xrightarrow{\tau}_{\mathcal{T}} s_1$ we have to show that $s_0 \xrightarrow{\tau} s_4$ can be mimicked from s_1. Indeed, $s_1 \xrightarrow{\tau} s_0$ and $s_0 \twoheadrightarrow\twoheadleftarrow_{\mathcal{T}} s_4$.

Based on $\mathcal{T}$, we could for instance omit the transition $s_0 \xrightarrow{\tau} s_4$. However, this is clearly incorrect, as is prevents the observable a-transition from occurring on any path emanating from s_0. The problem is in the fact that while $s \xrightarrow{\tau}_{\mathcal{T}} t_1$ and $s \xrightarrow{\tau}_{\mathcal{T}} t_2$ imply $t_1 \twoheadrightarrow\twoheadleftarrow_{\mathcal{T}} t_2$ for all pairs of transitions in the system, not necessarily $s \twoheadrightarrow_{\mathcal{T}} t_1$ and $s \twoheadrightarrow_{\mathcal{T}} t_2$ imply $t_1 \twoheadrightarrow\twoheadleftarrow_{\mathcal{T}} t_2$. This problem is well-known in the context of term rewriting: the weak Church-Rosser property (local joinability, also called local confluence in term rewriting terminology) does not imply the Church-Rosser property (global joinability, also called confluence in term rewriting terminology) [BN98].

This problem can be solved in multiple ways. In [TSvdP11], when defining weak probabilistic confluence, we explicitly required $\twoheadrightarrow\twoheadleftarrow$ to be an equivalence relation. While this correctly ensures that no behaviour is lost, it is hard to verify in practice. Therefore, in this chapter we decided to solve the problem by requiring confluent connectivity between μ and ν by having direct confluent transitions from the support of μ to the support of ν.

6.5.3 Diamond-shaped mimicking

As discussed in Remark 6.8, the confluence classification and the corresponding requirement that transitions are mimicked by transitions from the same class together ensure that confluent transitions are mimicked by confluent transitions. This was needed for representation maps to exist. Without the confluence classification this would not be the case anymore, as can be seen from the system in Figure 6.13(a).

Without using a confluence classification, we could consider the two outgoing transitions from s_0 to be confluent. Indeed, the conditions for confluence are satisfied. However, confluent transitions are mimicked by non-confluent transitions, and hence no representation map can exist. After all, the confluent transitions from s_0 to s_1 and s_2 imply that $\varphi(s_1) = \varphi(s_2)$. However, we also require each state to be able to reach its representative while only traversing confluent transitions. These two requirements now contradict each other.

It may seem that confluent mimicking can also be ensured by requiring

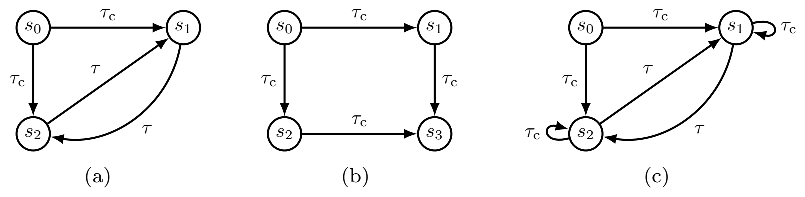


Figure 6.13: Counterexamples for the existence of a representation map.

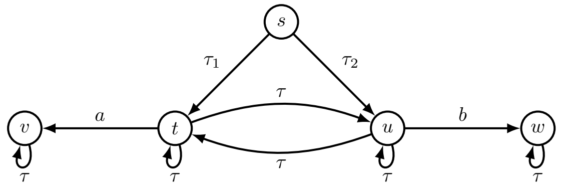
commutation to always happen in the shape of a diamond. This would disallow the confluent set taken before for Figure 6.13(a). Also, for models like the one depicted in Figure 6.13(b) indeed confluent mimicking holds; for $s_0 \xrightarrow{\mathcal{T}} s_1$ to be confluent $s_2 \xrightarrow{\mathcal{T}} s_3$ has to be, and for $s_0 \xrightarrow{\mathcal{T}} s_2$ to be confluent $s_1 \xrightarrow{\mathcal{T}} s_3$ has to be. However, this would still not entirely solve the problem. Consider for instance the system in Figure 6.13(c). Now, we can show that the two transitions from s_0 and the two self-loops together form a confluent set, that does satisfy the property of only commuting in diamonds. As before, also for this confluent set no valid representation map can be found.

To solve this problem, we need to explicitly require confluent transitions to be mimicked by confluent transitions. We chose to do so by grouping transitions by means of a confluence classification, and requiring transitions to be mimicking within their own group.

6.5.4 Explicit confluent mimicking

As discussed before, the confluence classification is used to ensure that confluent transitions are mimicked by confluent transitions to make sure representation maps exist. It may seem viable to just explicitly require confluent transitions to be mimicked by confluent transitions; actually, that was how previous work dealt with this problem [BvdP02].

To see what goes wrong, consider the following system:



When requiring confluent transitions to be mimicked by confluent transitions (or equivalently explicitly requiring $\twoheadrightarrow\twoheadleftarrow_{\mathcal{T}}$ to be transitive) instead of using a confluent classification, the sets

$$\mathcal{T}_1 = \{(s,\tau_1,t),(t,\tau,t),(u,\tau,u),(v,\tau,v),(w,\tau,w)\}$$
$$\mathcal{T}_2 = \{(s,\tau_2,u),(t,\tau,t),(u,\tau,u),(v,\tau,v),(w,\tau,w)\}$$

would both be perfectly valid confluent sets. Still, $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$ does not satisfy these requirements. After all, whereas it designates $s \xrightarrow{\tau_1} t$ to be confluent, its mimicking transition $u \xrightarrow{\tau} t$ from u (needed since $s \xrightarrow{\tau_2} u$ is in $\mathcal{T}$) is not confluent. Hence, the old notions were not closed under union[7].

[7] Since [BvdP02] constructed a confluent set per summand and then took the union of these sets, the fact that confluent sets are not closed under unions may have broken their approach. However, as mentioned before, no problems arise in their implementation due to the fact that they also only check for mimicking transitions by the same summand. Hence, although their claim that confluent sets can just be combined to get larger confluent sets was incorrect in general, their application of this claim did not produce any problems due to the more restricted context in which it was applied.

By using a confluence classification and requiring transitions to be mimicked by the same group, we ascertain that this kind of bad compositionality behaviour does not occur. After all, for $\mathcal{T}_1$ to be a valid confluent set, the confluence classification should be such that $s \xrightarrow{\tau_2} u$ and its mimicking transition $t \xrightarrow{\tau} u$ are in the same group. So, for $s \xrightarrow{\tau_2} u$ to be confluent (as prescribed by $\mathcal{T}_2$), also $t \xrightarrow{\tau} u$ would need to be confluent. The latter is impossible, since the b-transition from u cannot be mimicked from t, and hence $\mathcal{T}_2$ is disallowed.

6.6 Contributions

We introduced confluence reduction for MAs: the first practical reduction technique for this model that abstracts from invisible transitions. We showed that it preserves divergence-sensitive branching bisimulation, and hence yields quantitatively behavioural equivalent models. In addition to working on MAs, our novel notion of confluence reduction has two additional advantages over previous notions. First, it preserves divergences, and hence does not alter minimal reachability probabilities. Second, it is closed under unions, enabling us to separately detect confluence of different sets of transitions and combine the results. We also showed that a representation map approach can be used safely to reduce systems on-the-fly, and discussed how to detect confluence syntactically on the process-algebraic language MAPA—both using a symbolic (logical) characterisation and a set of heuristics.

Our case studies in Chapter 9 will demonstrate that significant reductions can be obtained, reducing state spaces sometimes by an order of magnitude. A decrease in the degree of nondeterminism, as is to be expected from confluence reduction, often yields even better reductions in analysis time than in state space size.

We conjecture that our technique can also be applied in different settings, for instance in the PRISM model checker [KNP11]. Due to the fact that this tool employs MTBDDs to do its analysis, an application of confluence during model checking may be difficult. However, we could detect confluence on its modelling language in quite the same way as described in this chapter, and use this information to syntactically optimise specifications. For instance, we may add guards to non-confluent commands to disable them in case at least one confluent command is enabled. Since the representation map approach cannot be applied in this context, it would require acyclicity of the confluent commands to avoiding the ignoring problem.

CHAPTER 7

Confluence Reduction versus Partial Order Reduction

A Theoretical Comparison

"In theory, theory and practice are the same.
In practice, they are not."

Albert Einstein

The previous chapter introduced confluence reduction for MAs. As mentioned before, this technique is most closely related to the concept of *partial order reduction* (POR). Both use a notion of independence between transitions of a system, either explicitly or implicitly, and try to reduce the state space by eliminating redundant paths in the system.

In the non-probabilistic setting, partial order reduction techniques have been defined for a range of property classes, most notably $\text{LTL}_{\setminus X}$ and $\text{CTL}^*_{\setminus X}$ [GKPP95, GKPP99, WW96, Val96, Pel98]. Most work on confluence reduction has been designed to guarantee that the reduced system is branching bisimilar to the original system; thus, these techniques preserve virtually all branching properties (in particular, $\text{CTL}^*_{\setminus X}$). There is not so much work on weaker variants of confluence, though [LM09] explores a variant that makes no distinction between visible and invisible actions and does not require acyclicity—hence, it preserves only deadlocks, similar to weaker versions of ample and stubborn sets [Val96].

While POR has not yet been defined for MAs, it was already generalised to the probabilistic setting. In [BGC04] and [DN04], the *ample sets* approach was lifted from labelled transition systems to Markov decision processes (MDPs), providing reductions that preserve probabilistic $\text{LTL}_{\setminus X}$. These techniques were refined in [BDG06] to also preserve $\text{PCTL}^*_{\setminus X}$, a branching logic. Later, a revision of partial order reduction for distributed schedulers was introduced and implemented in PRISM [GDF09]. In [BGC09], the use of fairness constraints in combination with ample sets for the quantitative analysis of MDPs was first introduced. Later, the so-called weak stubborn set method was defined for a class of safety properties of MDPs under fairness constraints [HKQ11].

Ample sets and confluent transitions are defined and detected quite differently: ample sets are defined by first giving an independence relation for the action labels, whereas confluence is a property of a set of (invisible) transitions in the final state space. Even so, the underlying ideas are similar on the intuitive level.

Indeed, [LM09] even describes them as 'two lines of work' in the area of partial order reduction. Since both techniques are in general not able to achieve optimal reductions as compared to the bisimulation-minimal quotient, we are interested to see if there are scenarios that can be handled by one technique but not by the other, or whether their reduction capabilities are equally powerful. Therefore, an obvious question is: to what extent do ample sets and confluent transitions coincide? This chapter addresses that question by comparing the notion of probabilistic ample sets from [BDG06] to a slightly reformulated variant on the notion of confluence from Chapter 6. We restrict to ample sets, because they are currently the most well-established notion of partial order reduction for MDPs.

Our approach. We introduce MDPs as a restriction of MAs, and recall some of their basic terminology. Then, we present the theory on ample set reduction from [BDG06] by means of the concept of a *reduction function*. Such functions specify for each state which actions (and hence which transitions) are enabled. We reformulate confluence reduction in the same way, taking into account that MDPs are insensitive to action visibility—i.e., there is no notion of τ-actions; only the state labelling is assumed to be observable.

We compare the two notions and show that, when preserving branching time behaviour, confluence reduction is strictly more powerful than ample set reduction. For this purpose, we prove that every nontrivial ample set can be mimicked by a confluent set, while also providing examples where confluent transitions do not qualify as ample sets. In such cases, confluence reduction is able to reduce more than ample set reduction. Mainly, confluence imposes fewer restrictions on the independence of actions. To continue, we pinpoint precisely in what way confluence is more general than ample sets, and show how the definitions of both can be adjusted to make them coincide.

While revealing exactly where the extra reduction potential with confluence comes from, the results we present support the idea that confluence reduction is a well-suited alternative to the thus far more often used partial order reduction methods. In particular, this is a major consideration in contexts where (1) detection of confluence using heuristics that make use of these more relaxed conditions is possible, or where (2) the conditions of confluence are just easier to check than their partial order reduction counterparts.

1. The first situation seems to occur in the context of statistical model checking and simulation. In this context, [BFHH11] used partial order reduction to remove spurious nondeterminism from models to allow them to be analysed statistically. As the reduction is applied directly to explicit models rather than high-level specifications, the more relaxed confluence conditions may come in handy. Indeed, the next chapter shows that confluence reduction *is* able to remove nondeterminism that partial order reduction could not, thereby allowing more models to be analysed using statistical model checking techniques. Our results in this chapter provide theoretical support for this intuition.
2. The second situation seems to arise when working with process-algebraic modelling languages. As demonstrated in [Blo01, BvdP02] for the non-

probabilistic setting and in the previous chapter for the setting of Markov automata, it is quite natural to detect confluence in such a context. Although it is also possible to work with POR in this setting (see for instance [FTW10]), the way confluence is phrased seems to be more closely related to the models involved, making its detection more straightforward.

Alternatively, our results (in particular Theorem 7.31) allow for the use of more relaxed definitions—incorporating a notion of *local* independence—if partial order reduction is used. In addition to providing these practical opportunities, our precise comparison of confluence and partial order reduction fills a significant gap in the theoretical understanding of the two notions.

The theory is presented in such a way that the results hold for non-probabilistic automata as well, as they form a special case of the theory where all probability distributions are deterministic. Hence, as a side effect we also answer the question of how the non-probabilistic variants of ample set reduction and confluence reduction relate.

Our findings imply that results and techniques applicable to confluence can be used in conjunction with ample sets—for instance, the state space generation technique based on *representative states*, already known in the context of confluence reduction as shown in [BvdP02] and in the previous chapter. Applying this technique to POR replaces explicit checking of the cycle condition, in addition to further reducing the number of states and transitions. The latter is important, especially if the MDP is to be subjected to further analysis.

Organisation of the chapter. After recalling some basic preliminaries in Section 7.1, we present the notions of ample set reduction and confluence reduction in Section 7.2. Then, in Section 7.3 we discuss how ample set reduction can be thought of as a special case of confluence reduction. We show what kind of restrictions and relaxations are needed to make them coincide, thereby pinpointing the exact differences of the methods. Finally, Section 7.4 concludes by summarising the contributions of this chapter.

Origins of the chapter. This chapter is based on a journal paper that will soon be published in *Theoretical Computer Science* [HT13a]. That paper was written in collaboration with Henri Hansen from the Institute of Mathematics at Tampere University of Technology, Tampere, Finland (working at the Temasek Laboratories of the National University of Singapore, Singapore during some of the work). The author updated the definitions slightly, replacing the more complicated notion of equivalence of probability distributions used in [HT13a] by the more simple way of dealing with this as presented in the previous chapter (and first published in [HT13b]).

7.1 Preliminaries

Probabilistic ample set reduction has been defined on the Markov decision process. This model is a subclass of the MA, having no Markovian rates and finite sets of states and actions, and allowing each action to occur only once per state. Because of the latter restriction, the transition relation is often formalised slightly different from what we have done in the previous chapters. In this chapter, we conform to MDP terminology [Put05] and formalise the transition relation as a function assigning to each pair of a state and action the probability distribution for the next state. To allow actions to be disabled, we now use the set $\mathrm{Distr}_\perp(S) = \mathrm{Distr}(S) \cup \{\perp\}$ that not only contains all probability distributions over the set of states S, but also the subdistribution $\perp$ that assigns probability 0 to every state $s \in S$.

Hence, an MDP consists of states that are labelled by atomic propositions, an initial state, and a probabilistic action-labelled transition function. From each state s, a subset of the actions is enabled; for every enabled action a, a probability distribution $P(s,a)$ specifies for every other state s' the probability $P(s,a)(s')$ of ending up in s' after taking a from s.

Recall from Section 3.2 that we assume a countable universe of actions *Act*.

Definition 7.1 (MDPs). *A* Markov decision process (MDP) *is tuple* $M = (S, s^0, A, P, \mathrm{AP}, L)$*, where*

- S *is a finite set of* states*;*
- $s^0 \in S$ *is the* initial state*;*
- $A \subseteq Act$ *is a finite set of* actions*;*
- $P\colon (S \times A) \to \mathrm{Distr}_\perp(S)$ *is the* probabilistic transition function*;*
- AP *is a finite set of* state labels*;*
- $L\colon S \to \mathscr{P}(\mathrm{AP})$ *is the* state-labelling function.

If $P(s,a) = \perp$, the action a is not enabled from s. Otherwise, $P(s,a)(s')$ is the probability of going to s' when executing a from s.

We still like to be able to speak about transitions of an MDP, and hence introduce the notation (s,a,μ) for a transition from s, taking an action a and having a next-state distribution μ. Note that such a transition is present if $P(s,a) = \mu$. As before, we often write $s \xrightarrow{a} \mu$ instead of (s,a,μ).

Definition 7.2 (Supporting notations for MDPs). *Given an MDP $M = (S, s^0, A, P, \mathrm{AP}, L)$, we denote the set of all possible transitions of M by*

$$\Delta_M = \{(s,a,\mu) \in S \times A \times \mathrm{Distr}(S) \mid P(s,a) = \mu\}$$

Additionally,

- *We write $s \xrightarrow{a} \mu$ if $(s,a,\mu) \in \Delta_M$, and $s \xrightarrow{a}$ if $s \xrightarrow{a} \mu$ for some $\mu \in \mathrm{Distr}(S)$. Also, we define $en(s) = \{a \in A \mid s \xrightarrow{a}\}$ to be the set of actions enabled from state s.*
- *We reuse all notations defined in Chapter 3 for extended transitions also for transitions in an MDP—including the notations for paths, reachability, joinability and convertibility—and subscript them with a set of transitions $\mathcal{T} \subseteq \Delta_M$ to restrict to using only transitions from this set.*
- *We write $s \xrightarrow{a_1 a_2 \dots a_n} s'$ if there exists a path*

$$s_0 \xrightarrow{a_1, \mu_1} s_1 \xrightarrow{a_2, \mu_2} \dots \xrightarrow{a_n, \mu_n} s_n$$

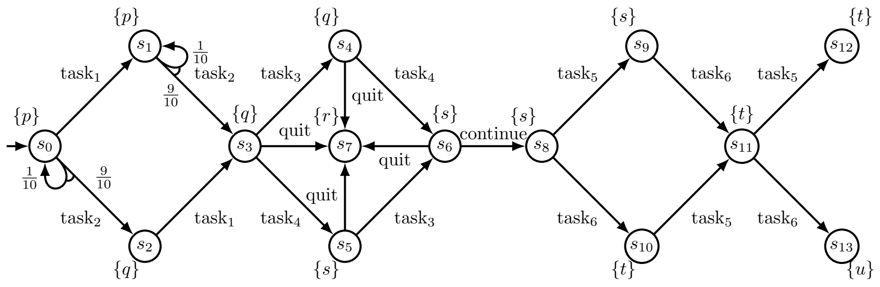


Figure 7.1: An MDP M representing a flow chart.

for some distributions $\mu_i \in \mathrm{Distr}(S)$*, with* $s_0 = s$ *and* $s_n = s'$*. We write* $s \xrightarrow{a_1 a_2 \ldots a_n}$ *if* $s \xrightarrow{a_1 a_2 \ldots a_n} s'$ *for some* $s' \in S$*.*

A subset $\Delta' \subseteq \Delta_M$ *of transitions of an MDP is* acyclic *if there does not exist a cycle in the subgraph of the MDP when only considering the transitions of* Δ'*.*

Note that we used $\mathrm{Distr}(S)$ instead of $\mathrm{Distr}_\perp(S)$ in Definition 7.2. Hence, $\mu \neq \perp$.

Example 7.3. Figure 7.1 depicts an MDP M consisting of 14 states, that will serve as a running example throughout this chapter. It represents a flow chart, specifying the ways in which six tasks (that all have to be executed) can be performed. The tasks occur in pairs: first $task_1$ and $task_2$ need to be executed, then $task_3$ and $task_4$, and finally we need to do $task_5$ and $task_6$. Each pair of tasks can be executed in either order. Furthermore, the execution of $task_2$ fails with probability $\frac{1}{10}$, in which case it can be attempted again. Moreover, after finishing the first two tasks and before starting the last two, we can quit or choose to continue. Finally, after all tasks have been completed, it is allowed to repeat either $task_5$ or $task_6$. We assume that the effect of the even-numbered tasks is visible to the environment (indicated by a change of atomic proposition due to such a transition), while the odd-numbered tasks are invisible.

Note that for this MDP we have

$$S = \{s_i \mid 0 \leq i \leq 13\} \qquad \text{and} \qquad A = \{task_i \mid 1 \leq i \leq 6\} \cup \{quit, continue\}$$

The probabilistic transition function is again visualised by (the absence of) arrows. For instance, $P(s_0, task_2) = \mu$ such that $\mu(s_0) = \frac{1}{10}$ and $\mu(s_2) = \frac{9}{10}$, and $P(s_0, quit) = \perp$. Furthermore, we have $s^0 = s_0$ and $\mathrm{AP} = \{p, q, r, s, t, u\}$. The labelling is indicated for each state, e.g., $L(s_2) = \{q\}$. We find that $en(s_3) = \{quit, task_3, task_4\}$, as well as $s_5 \xrightarrow{task_3\ continue\ task_5\ task_6}$. □

The following definition introduces three important concepts for transitions and actions: determinism, stuttering and reducibility.

Definition 7.4 (Determinism, Stuttering and Reducibility). *Given an MDP* $M = (S, s^0, A, P, \mathrm{AP}, L)$*,*

- *A transition $s \xrightarrow{a} \mu$ is* deterministic *if μ is deterministic (i.e., assigns probability* 1 *to a single state), and an action $a \in A$ is a* deterministic action *in M if all a-labelled transitions in Δ_M are deterministic. Assuming an implicit MDP, we denote the set of all deterministic actions by $A_{\text{det}} \subseteq A$. Given a deterministic transition $s \xrightarrow{a} \mathbb{1}_t$, we write $target(s, a) = t$;*
- *A transition $s \xrightarrow{a} \mu$ is* stuttering *if $L(s') = L(s)$ for each $s' \in supp(\mu)$, and an action $a \in A$ is a* stuttering action *in M if all a-labelled transitions in Δ_M are stuttering. We denote the set of all stuttering actions by $A_{\text{st}} \subseteq A$;*
- *A transition $s \xrightarrow{a} \mu$ is* reducible *if it is both deterministic and stuttering, and an action $a \in A$ is a* reducible action *if all a-labelled transitions in Δ_M are reducible. We denote the set of all reducible actions by $A_{\text{red}} = A_{\text{st}} \cap A_{\text{det}}$;*
- *A finite path $s \xrightarrow{a_1 a_2 \dots a_n} s'$ or infinite path $s \xrightarrow{a_1 a_2 \dots}$ is* reducible *if every action a_i on it is reducible.*

As before, we sometimes abuse notation a little by writing $s \xrightarrow{a} s'$ instead of $s \xrightarrow{a} \mathbb{1}_{s'}$ for deterministic transitions.

Note that $s \xrightarrow{a} \mu$ may be a reducible transition even if a is not a reducible action, but not vice versa. Finally, given a sequence of reducible (and thus deterministic) actions $a_1 a_2 \dots a_n$, talking about "the" path of this sequence from some state s makes sense, because the states that are visited are unique. We do so for the rest of this chapter.

Example 7.5. In the MDP M in Figure 7.1, all transitions except for the ones labelled by $task_2$ are deterministic. Hence, all actions except for $task_2$ are deterministic. All transitions labelled by odd tasks are stuttering, as well as the *continue* transition, since the atomic propositions in their source and target states all correspond. Hence, all odd-labelled *task* actions and the *continue* action are stuttering. Combining this, we obtain

$$A_{\text{red}} = \{task_i \mid i \in \{1, 3, 5\}\} \cup \{continue\} \qquad \square$$

A wide class of reductions for an MDP can be defined using the construct called a *reduction function.* Such a function determines a sub-MDP by deciding for each state which outgoing actions are enabled in the reduced MDP. The transition function of the reduced MDP then consists of all transitions that are still enabled after the reduction function is applied, and the set of states consists of all states that are still reachable using the reduced transition function.

Definition 7.6 (Reduction functions). *Given an MDP $M = (S_M, s^0_M, A, P_M, \text{AP}, L_M)$, a* reduction function *for M is any function $R \colon S_M \to \mathscr{P}(A)$ with $R(s) \subseteq en(s)$ for every $s \in S_M$. Given a reduction function R, the* reduced MDP for M with respect to R *is the minimal MDP $M_R = (S_R, s^0_R, A, P_R, \text{AP}, L_R)$ such that $s^0_R = s^0_M$ and*

- *If $s \in S_R$ and $a \in R(s)$, then $P_R(s, a) = P_M(s, a)$ (and hence by definition of P_R also $supp(P_M(s, a)) \subseteq S_R$);*

- *If $s \in S_R$ and $a \notin R(s)$, then $P_R(s,a) = \bot$;*
- *$L_R(s) = L_M(s)$ for every $s \in S_R$,*

where minimal should be interpreted as having the smallest set of states.

Given a reduction function $R\colon S \to \mathscr{P}(A)$, we define $\overline{R}\colon S \to \mathscr{P}(A)$ by

$$\overline{R}(s) = \begin{cases} \varnothing & \textit{if } R(s) = en(s) \\ R(s) & \textit{otherwise} \end{cases}$$

The transitions designated by $\overline{R}$ are called the nontrivial transitions *of the reduction. We say that a reduction function R is* acyclic *if the original MDP restricted to the transitions designated by $\overline{R}$ is acyclic.*

In other words, $\overline{R}$ assigns to each state s the subset of actions that are enabled by R in case a real reduction is made for s. Otherwise, it assigns no actions to s.

Example 7.7. A reduction function R for the MDP in Figure 7.1 is given by $R(s_0) = \{task_1\}$, $R(s_1) = \{task_2\}$, $R(s_3) = \varnothing$ and $R(s_i) = en(s_i)$ for every other state s_i. The reduced MDP with respect to R consists of solely the states s_0, s_1 and s_3, and the two transitions connecting them. We have $\overline{R}(s_0) = \{task_1\}$ and $\overline{R}(s_1) = \varnothing$, and find that R is acyclic (which is immediate, as the only nontrivial transition is no self-loop). □

When reducing MDPs, we clearly want to retain some behaviour to still be able to verify certain properties. The reductions we deal with preserve PCTL$^*_{\setminus X}$ (a probabilistic variant of CTL$^*_{\setminus X}$; see for instance [BK08]).

7.2 Ample sets and confluence for MDPs

This section presents the theory of the ample set reduction technique. Its correctness is based on the concepts of weight functions and probabilistic visible (bi)simulation [Grö08]. We briefly introduce these concepts, without going into many details. We also illustrate how the concepts developed for confluence reduction in Chapter 6 can be applied to MDPs using reduction functions.

Definition 7.8 (Weight functions). *Let $R \subseteq S_1 \times S_2$ be a binary relation and let $\mu \in \mathrm{Distr}_\bot(S_1)$ and $\nu \in \mathrm{Distr}_\bot(S_2)$ be probability distributions. We write $\mu \sqsubseteq_R \nu$ if $\mu, \nu \neq \bot$ and there exists a* weight function $w\colon S_1 \times S_2 \to [0,1]$ *such that for all $s_1 \in S_1$ and $s_2 \in S_2$,*

- *$w(s_1, s_2) > 0$ implies $(s_1, s_2) \in R$;*
- *$\displaystyle\sum_{s \in S_2} w(s_1, s) = \mu(s_1)$ and $\displaystyle\sum_{s \in S_1} w(s, s_2) = \nu(s_2)$.*

Example 7.9. Consider the two systems in Figure 7.2. Clearly s_0 and t_0 are not equivalent, as the observable behaviour after the a-transitions differs. However, all behaviour of s_0 can be mimicked by t_0; something that can be shown using weight functions. We let R be a relation to indicate the simulation. That is, we

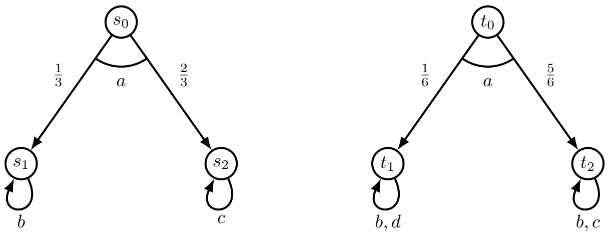


Figure 7.2: Two MDPs to illustrate weight functions.

let $(s, t) \in R$ denote that t can mimic all behaviour from s—below, we make this more precise.

We want to show that $R = \{(s_0, t_0), (s_1, t_1), (s_1, t_2), (s_2, t_2)\}$ is a valid simulation relation. Except for the first element this is clear, since all behaviour of s_1 can indeed be mimicked by t_1 and t_2, and all behaviour of s_2 can be mimicked by t_2. Now, we want to show that also $(s_0, t_0) \in R$ is valid. Can the a-transition from s_0 be mimicked by t_0? There is an a-transition, but the probabilities differ. We now define a weight function, as follows:

$$w(s_1, t_1) = \tfrac{1}{6} \qquad w(s_1, t_2) = \tfrac{1}{6} \qquad w(s_2, t_2) = \tfrac{2}{3}$$

Basically, this weight function shows how the probability mass can 'flow' from the left distribution μ to the right distribution ν. All requirements are satisfied, and hence $\mu \sqsubseteq_R \nu$. Indeed, s_0 can go with probability $\frac{1}{3}$ to a state that can do a b-transition, and for t_0 this probability is at least as high. The c-transition can be executed from s_0 with probability $\frac{2}{3}$, and t_0 can also mimic this with a probability that is at least as high. □

Next, we recall the notion of probabilistic visible bisimulation [Grö08]. It is based on probabilistic visible simulation, which basically formalises our observation of mimicking behaviour in the example above. For two states (s, s') to be probabilistically visible similar, (1) they need to have the same state labelling, (2) every transition $s \xrightarrow{a} \mu$ needs to either (a) have a reducible (deterministic and stuttering) action and go to a state t that is also similar to s' or (b) be mimicked from s' after a (possibly empty) path of reducible transitions that only visits states that simulate s, and (3) every diverging path from s needs to be mimicked by a diverging path from s'.

Definition 7.10 (Probabilistic visible bisimulation). *Let $M_1 = (S_1, s_1^0, A, P_1, \mathrm{AP}, L_1)$ and $M_2 = (S_2, s_2^0, A, P_2, \mathrm{AP}, L_2)$ be MDPs, and let $R \subseteq S_1 \times S_2$ be a binary relation. Then, R is a* probabilistic visible simulation *for (M_1, M_2) if $(s_1^0, s_2^0) \in R$ and, for every $(s, s') \in R$, the following three conditions hold:*

1. *$L_1(s) = L_2(s')$;*
2. *If $a \in en(s)$, then either*

   (a) $a \in A_{\mathrm{red}}$ and $(target(s, a), s') \in R$, or

(b) there is a reducible path $s' \xrightarrow{b_1 \dots b_n} s''$ in M_2 such that $(s, s'_i) \in R$ for every state s'_i on this path, $a \in en(s'')$ and $P_1(s,a) \sqsubseteq_R P_2(s'', a)$;

3. If there is an infinite reducible path $s \xrightarrow{b_1 b_2 \dots}$ in M_1 such that $(s_i, s') \in R$ for every s_i on this path, then there is a finite reducible path $s' \xrightarrow{a_1 \dots a_n} s'_n$ in M_2, $n \geq 1$, such that $(s, s'_i) \in R$ for every s'_i on this path (possibly excluding s'_n), and $(s_k, s'_n) \in R$ for at least one s_k (with $k > 0$) on the path $s \xrightarrow{b_1 b_2 \dots}$.

A binary relation R is a probabilistic visible bisimulation *for (M_1, M_2) if it is a probabilistic visible simulation for (M_1, M_2) and R^{-1} is a probabilistic visible simulation for (M_2, M_1).*

We say that two MDPs M_1, M_2 are probabilistically visibly bisimilar, denoted by $M_1 \approx_{\text{pvb}} M_2$, if there is a probabilistic visible bisimulation that relates them.

We note that probabilistic visible bisimulation is very much related to our notion of divergence-sensitive branching bisimulation. The only real difference is that probabilistic visible bisimulation requires transitions to be labelled by actions that are deterministic and stuttering globally in conditions 2a and 2b, while the branching step only requires the transitions to be this themselves.

7.2.1 Ample sets

Although there are many techniques that are called “partial order reduction”, we focus on the *ample set method* as presented in [BDG06]. This is the only partial order reduction technique we are aware of that preserves all probabilistic branching time properties. To present the definition, we first need to introduce the notion of independence. Intuitively, two actions a, b are independent if they don’t disable each other, and if the probability of ending up at any state by first taking a and then taking b is the same as when the actions are taken the other way around.

Definition 7.11 (Independence). *Given an MDP $M = (S, s^0, A, P, \text{AP}, L)$, two actions $a, b \in A$ are* independent *if $a \neq b$ and for every state $s \in S$ with $\{a, b\} \subseteq en(s)$ the following conditions hold:*

- *If $s' \in supp(P(s,a))$, then $b \in en(s')$ (and symmetrically);*
- *We have*

$$\sum_{s' \in S} P(s,a)(s') \cdot P(s',b)(t) = \sum_{s' \in S} P(s,b)(s') \cdot P(s',a)(t)$$

for every $t \in S$.

If a and b are not independent, we say that they are dependent*. An action a is dependent on a set B if there exists at least one $b \in B$ on which a depends.*

Example 7.12. In the MDP M given in Figure 7.1, the actions $task_1$ and $task_2$ are independent. After all, there is only one state in which both are enabled: s_0. From there, indeed, these two actions do not disable each other. Moreover, when first executing $task_1$ and then executing $task_2$, the probability of ending up in s_1 is $\frac{1}{10}$ and the probability of ending up in s_3 is $\frac{9}{10}$. When executing the tasks the other way around, we obtain the same probabilities.

Similarly, it can be shown that $task_3$ and $task_4$ are independent. Note that $task_5$ and $task_6$ are not independent, as they are both enabled in s_{11} and from there can disable each other. □

Based on this notion of dependence, the ample set constraints can be defined.

Definition 7.13 (Ample set reduction). *Let $M = (S, s^0, A, P, \mathrm{AP}, L)$ be an MDP without terminal states. Then, a reduction function $R \colon S \to \mathscr{P}(A)$ for M is an* ample set reduction function *if it satisfies the following conditions in every state $s \in S$:*

A0 $\varnothing \neq R(s) \subseteq en(s)$*;*

A1 *If $R(s) \neq en(s)$, then $R(s) \subseteq A_{\mathrm{st}}$;*

A2 *For every path $s \xrightarrow{a_1 a_2 \dots a_n b} t$ in M such that $b \notin R(s)$ and b depends on $R(s)$, there exists an $1 \leq i \leq n$ such that $a_i \in R(s)$;*

A3 *For every cycle $s \xrightarrow{a_1 a_2 \dots a_n} s$ in M_R, $R(s_i) = en(s_i)$ for at least one state s_i on this cycle;*

A4 *If $R(s) \neq en(s)$, then $|R(s)| = 1$ and $R(s) \subseteq A_{\mathrm{det}}$.*

The sets $R(s)$ are called ample sets.

Except for A4, the ample set provisos are standard in traditional model checking. They make sure that no deadlocks are introduced (A0, A2) and all visible behaviour is preserved (A1, A3). Proviso A3 is known as the *cycle* condition, and prevents visible behaviour from being postponed indefinitely. The last proviso is needed in a probabilistic branching-time setting. We refer to [Grö08, BK08] for an extended explanation of these provisos.

Note that we could also extend these conditions to allow MDPs *with* terminal states. In that case A0 should be changed to allow $R(s) = \varnothing$ if $en(s) = \varnothing$. Note also that conditions A1 and A4 can be combined by saying that either $R(s) = en(s)$ or $R(s)$ contains exactly one reducible action.

Example 7.14. A valid ample set reduction function R for M is given by $R(s_0) = \{task_1\}$ and $R(s_i) = en(s_i)$ for all other states. Note that all ample set conditions vacuously hold for all fully-expanded states, so we only need to investigate s_0. The conditions A0, A1 and A4 are trivial to verify. Also A3 is easy, since the only possible cycle in M_R is an infinite loop through s_1 (although this has probability 0): indeed $R(s_1) = en(s_1)$. Finally, to see why A2 holds, note that every path from s_0 either immediately traverses $task_1$ (which is indeed in $R(s_0)$) or starts with a number of times $task_2$ and then $task_1$; for all traces of the second kind, $task_2$ is independent of $R(s_0)$ and $task_1$ is in $R(s_0)$, satisfying the condition.

This reduction function only gets rid of state s_2. Note that no additional reduction is possible. In s_3, s_4, s_5 and s_6, no subset of the enabled actions can be chosen as an ample set, since none of the actions is independent of the *quit* action (as *quit* disables all other actions). Also, in s_8 no reduction is possible, since $task_5$ and $task_6$ are not independent (after all, in state s_{11} they can disable each other). □

The following result states that ample sets are sound for MDP reduction.

Theorem 7.15 ([BDG06]). *If R is an ample set reduction function for M, then $M \approx_{\text{pvb}} M_R$, and consequently M and M_R satisfy the same PCTL$^*_{\backslash X}$-formulae.*

7.2.2 Confluence

Since MDPs are a subclass of MAs, in principle all theory developed in Chapter 6 on confluence for MAs is still valid for MDPs. However, there are some small technical differences between the existing notions of ample set reduction and the notion of confluence reduction defined earlier in this thesis:

- The ample set reduction technique was defined in a state-based setting, where transitions do not have to be labelled by τ to be invisible; they just have to connect equally-labelled states. Hence, a model does not need to have any τ-actions for an ample set reduction to be possible, while this is the case for the notion of confluence defined in Chapter 6. For a fair comparison, we therefore update the definition of confluence to also be so liberal to allow any action label to be considered invisible.
- The ample set reduction technique was defined in terms of reduction functions, whereas we defined confluence reduction using a representation map approach. For a fair comparison, we therefore now redefine confluence reduction using the concept of reduction functions as well.
- Ample set reduction was shown to be correct by proving that it preserves probabilistic visible bisimulation. This implies that PCTL$^*_{\backslash X}$ is preserved. To apply the same notion of bisimulation to confluence reduction, we have the strengthen the concept slightly: instead of requiring confluent *transitions* to be stuttering and deterministic, we have to require their *actions* to be so. As mentioned below Definition 7.4, this does make the concept more demanding; however, it does enable a fairer comparison.

Since we are not concerned with taking unions of confluent sets, we can simplify their definition slightly by omitting the confluence classification. We just assume a set $\mathcal{T}$ of transitions, and do still require confluent transitions to be mimicked by confluent transitions as before.

Definition 7.16 (Probabilistic confluence). *Let $M = (S, s^0, A, P, \text{AP}, L)$ be an MDP. Then, a set $\mathcal{T} \subseteq \Delta_M$ is* probabilistically confluent *if all its transitions have a reducible action, and for all $s \xrightarrow{a}_{\mathcal{T}} t$ and all transitions $s \xrightarrow{b} \mu$, either*

- $\exists \nu \in \text{Distr}(S) \, . \, t \xrightarrow{b} \nu \wedge \mu \equiv_{R^{\mathcal{T}}_{\mu,\nu}} \nu \wedge \big((s \xrightarrow{b} \mu) \in \mathcal{T} \implies (t \xrightarrow{b} \nu) \in \mathcal{T}\big)$, *or*
- $b \in A_{\text{red}}$ *and* $\mu = \mathbb{1}_t$*,*

with $R^{\mathcal{T}}_{\mu,\nu}$ the smallest equivalence relation such that

$$R^{\mathcal{T}}_{\mu,\nu} \supseteq \{(s,t) \in supp(\mu) \times supp(\nu) \mid \exists a \,.\, (s \xrightarrow{a} t) \in \mathcal{T}\}$$

A transition $s \xrightarrow{a} t$ is confluent *if there exists a Markovian confluent set $\mathcal{T}$ such that $s \xrightarrow{a}_{\mathcal{T}} t$.*

Note that, compared to Definition 6.7 and as discussed above, Definition 7.16 requires confluent transitions to have a reducible *action* and assumes a confluence classification with just one class. Also, we omitted the assumption $(s \xrightarrow{b} \mu) \neq (s \xrightarrow{a} t)$ and added instead the clause $b \in A_{\text{red}}$ and $\mu = \mathbb{1}_t$. The reason for this is that while before there could only be one reducible transition from s to t (since it was required to have action label τ), now there can be multiple with different action labels. This notion of confluence is incomparable to the one in Chapter 6: the current variant is more liberal in the sense that actions do not have to be labelled by τ to be confluent, and more strict in the sense that *actions* instead of *transitions* have to be reducible.

The proofs from Chapter 6 can easily be adapted to show that confluent joinability still coincides with confluent convertibility and that confluent transitions connect states that are equivalent according to probabilistic visible bisimulation (as we did in [HT13a] for a slightly stronger version).

Based on this notion of probabilistic confluence, we can easily define a confluence reduction function. Instead of using representatives, it gives priority to confluent transitions and ignores all their neighbouring transitions.

Definition 7.17 (Probabilistic confluence reduction). *For an MDP $M = (S, s^0, A, P, \text{AP}, L)$, a reduction function $T\colon S \to \mathscr{P}(A)$ is a* confluence reduction function for M *if there exists a probabistically confluent set $\mathcal{T} \subseteq \Delta_M$ such that, for every $s \in S$ either*

- *$T(s) = en(s)$, or*
- *$T(s) = \{a\}$ for some $a \in A_{\text{red}}$ such that $(s, a, \mathbb{1}_t) \in \mathcal{T}$ for some $t \in S$.*

Then, we also say that T is a confluence reduction function under $\mathcal{T}$.

Note that, in every state, a confluence reduction function either fully explores all outgoing transitions (which are then not required to be confluent) or explores only one of them (which is then required to be confluent). This way, it may happen that confluent transitions are taken indefinitely, ignoring the presence of other actions. This problem is well known in the theory of partial order reduction as the *ignoring problem* [Val90, EP10], and is dealt with by the cycle condition A3 of the ample set method. For confluence we dealt with this problem before by using the representation map approach, choosing one representative in each BSCC. When using reduction functions, it can be dealt with by requiring acyclicity of the reduction function.

Example 7.18. Consider again the MDP M given in Figure 7.1. We define

$$\begin{aligned}\mathcal{T} = \{&(s_0, task_1, \mathbb{1}_{s_1}), (s_2, task_1, \mathbb{1}_{s_3}), (s_3, task_3, \mathbb{1}_{s_4}),\\ &(s_5, task_3, \mathbb{1}_{s_6}), (s_8, task_5, \mathbb{1}_{s_9}), (s_{10}, task_5, \mathbb{1}_{s_{11}})\}\end{aligned}$$

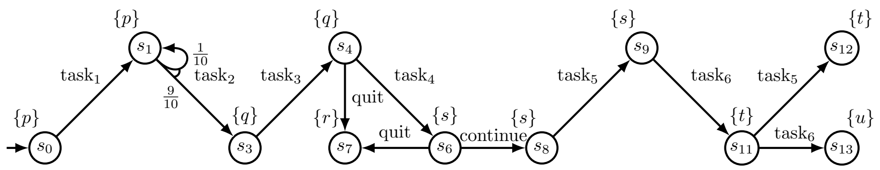


Figure 7.3: An MDP reduced based on confluence.

Indeed, all of these transitions have reducible actions. Moreover, the confluence requirements are easy to check in the same way as we illustrated in Example 6.9.

Based on $\mathcal{T}$, we can define the reduction function T given by $T(s_0) = \mathit{task}_1$, $T(s_3) = \mathit{task}_3$, $T(s_8) = \mathit{task}_5$ and $T(s) = \mathit{en}(s)$ for all other states s. The reduced MDP obtained in this way is shown in Figure 7.3. Note that, compared to the maximal ample set reduction that could be obtained for this MDP, we reduced on two more occasions in the MDP. □

Our main result of this section is Theorem 7.19, which establishes the correctness of acyclic confluence reduction functions with respect to probabilistic visible bisimulation (note that acyclicity was introduced in Definition 7.6). It was proven by the author and Henri Hansen in [HT13a] for a slightly more restrictive version of confluence, and also holds for the variant just introduced. Since the proof is very similar to the proof of Theorem 6.18, we omit it.

Theorem 7.19. *Let M be an MDP, $\mathcal{T}$ a probabilistically confluent set of transitions from M and T an acyclic confluence reduction function under $\mathcal{T}$. Let M_T be the reduced MDP. Then,*

$$M \approx_{\text{pvb}} M_T$$

Proposition 3.4.10 from [Grö08] gives the following corollary.

Corollary 7.20. *If T is an acyclic confluence reduction function for M, then M and M_T satisfy the same PCTL$^*_{\setminus X}$-formulae.*

7.3 Comparing ample sets and confluence

The relation between ample sets and confluence is not straightforward. In this section, we will first see that confluence is strictly more general, by proving that every ample set reduction also is a confluence reduction. In addition, we discuss the aspects that differentiate ample sets from confluence. To show that these are the only differences, we provide variations to the concepts that make them coincide. The choice of which concept is varied in each situation, is to some extent arbitrary. Restricting confluence or relaxing ample sets is not the issue here, the objective is to prove that we have identified the essential differences. However, the variations are made in such a way that the resulting notions are useful in practice. Restrictions of confluence rule out features that are plausibly hard to implement in practice, and relaxed features of ample sets are such that they have been used in practice.

7.3.1 Why confluence is strictly more powerful

The starting point of our investigation is given by Theorem 7.21 below. It shows that, if the ample set method allows a state to explore only one of its outgoing transitions, the confluence method also allows this. Therefore, any reduction that can be achieved by the use of ample sets can also be achieved using confluence (from Definition 7.16).

Theorem 7.21. *Let A be an ample set reduction function for an MDP $M = (S, s^0, A, P, \mathrm{AP}, L)$. Then, the set $\mathcal{T}_A = \{(s, a, \mu) \in \Delta_M \mid a \in \overline{A}(s)\}$ is acyclic, and consists of probabilistically confluent transitions.*

This result obviously also holds for weaker notions of confluence (as for instance presented in [TSvdP11]), which are even more powerful.

In the proof of Theorem 7.21, we construct a confluent set containing all transitions from $\mathcal{T}_A$. We can use this set to define a confluence reduction function, for every state having a transition in $\mathcal{T}_A$ choosing that transition and for every state for which this is not the case fully exploring all transitions. Since $\mathcal{T}_A$ is acyclic, this reduction function is acyclic as well.

On the other hand, it is not the case that every confluent transition can be chosen to be in a nontrivial ample set. Confluence reduction turns out to be more liberal on several aspects: it may reduce multiple transitions between the same two states, triangles and diamonds in which one transition is mimicked by a differently labelled transition, and also in the presence of actions that are only locally independent. All four cases are illustrated by the following examples.

Example 7.22. Consider the MDPs in Figure 7.4 (with the atomic propositions per state indicated in curly brackets). For these MDPs, all transitions are deterministic. Note also that all a-transitions are stuttering and therefore reducible. Even more, they are constructed in such a way that the outgoing a-transitions from every state s_1 are confluent, i.e., $\{s_1 \xrightarrow{a} s_2\}$ is a confluent set for all subfigures. Hence, confluence reduction allows us to omit the b-transition from each state s_1, thus removing six transitions and two states.

In Figure 7.4(a), the b-action is reducible too. Due to the second item of Definition 7.16, this transition does not prohibit the a-transition from being confluent. After all, this part of the definition basically allows confluent transitions to disable other reducible transitions having the same source and target state as the confluent transition, as illustrated here. Therefore, confluence reduction

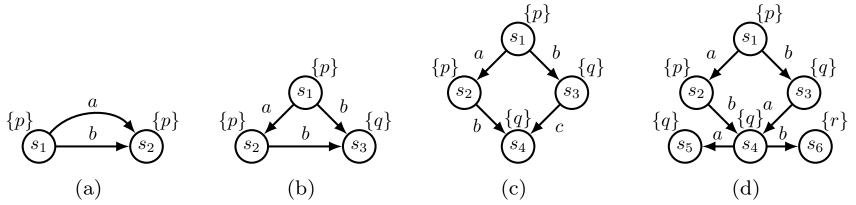


Figure 7.4: Confluence triumphs over ample sets.

may choose either one of these two transitions and could for instance reduce based on $\mathcal{T} = \{(s_1, a, \mathbb{1}_{s_2})\}$. The ample set conditions do not allow this; they require complete independence between a and b for $\{a\}$ to be a valid ample set for s_1. Hence, the only valid ample set for s_1 is $\{a, b\}$.

In Figure 7.4(b), the b-transition is not reducible. Furthermore, a and b are dependent, since b disables a. However, the a-transition from s_1 can still be considered confluent, taking $\mathcal{T} = \{(s_1, a, \mathbb{1}_{s_2})\}$ as the underlying confluent set for confluence reduction. This is due to the fact that although visible actions must still be enabled after a confluent transition, the confluent action does not need to be enabled after the visible action. Again, ample set reduction cannot reduce since a and b are not independent.

Although it may seem that reduction in case of triangle constructions such as Figure 7.4(b) only removes some transitions, it can in theory make a significant difference in the number of states. Imagine for instance a system in which every state has a transition *quit* to a single deadlock state (as is partially the case in Figure 7.1). Then, not one action is independent of *quit*, and ample set reduction would not be able to provide any reduction. However, such transitions would not interfere with confluence. Every confluence reduction that would be possible without the *quit* transitions is still possible with the *quit* transitions.

In Figure 7.4(c), the a-transition can be considered confluent since the diamond shape is closed perfectly (taking $\mathcal{T} = \{(s_1, a, \mathbb{1}_{s_2}), (s_3, c, \mathbb{1}_{s_4})\}$). Even though b disables a, there is a transition from s_3 to s_4 that can easily be shown confluent. The ample set conditions strictly require reducible transitions to be mimicked by equally-named transitions, disallowing any reduction for this model.

In Figure 7.4(d), the outgoing a-transition from s_1 is confluent since the diamond shape of independence is present (taking $\mathcal{T} = \{(s_1, a, \mathbb{1}_{s_2}), (s_3, a, \mathbb{1}_{s_4})\}$). The fact that a can disable b later on in the system does not matter for confluence. The ample set conditions, however, do require a and b to be globally independent for $\{a\}$ to be a valid ample set for s_1. As this is not the case, no reductions can be achieved with ample set reduction. □

Confluence mainly provides more reduction since it is defined based on the actual low-level transitions at a given state of the model, whereas the independence notion of ample set reduction works on higher-level actions and is considered to be global. That is, the dependency relation is assumed to be the same for every state. In practice, however, heuristics for detecting confluent transitions symbolically (as in [BvdP02] and in the previous chapter) often also take this more global point of view, which diminishes the difference.

Differences and heuristics. Our implementation in SCOOP (see Chapter 9) only considers transitions to be potentially confluent if they are labelled by τ. Hence, the differences between confluence and POR depicted in Figures 7.4(a) and 7.4(c) do not occur. If differently labelled transitions were possibly considered invisible as well, situation 7.4(a) could easily be detected efficiently during state space generation.

The heuristics implemented to decide on behaviour mimicking (basically checking if summands do not influence each other) do not allow any reduction in the scenarios depicted in Figures 7.4(b) and 7.4(c). However, situations as shown in Figure 7.4(d) can occur if $a = \tau$ and the action label b is used in multiple summands (as demonstrated below), and are indeed reduced using our heuristics.

Example 7.23. Consider the following MAPA specification:

$$\begin{aligned} & X(pc_1 : \{1,3\}, pc_2 : \{1,3\}) = \\ & \quad pc_1 = 1 \Rightarrow \tau \cdot X(2, pc_2) && (1) \\ & \quad + \ pc_2 = 1 \Rightarrow b \cdot X(pc_1, 2) && (2) \\ & \quad + \ pc_2 = 2 \wedge pc_2 = 2 \Rightarrow (\tau \cdot X(3,2) + b \cdot X(2,3)) && (3) \end{aligned}$$

This specification yields precisely the MA of Figure 7.4(d), with $a = \tau$. Indeed, our confluence implementation is able to reduce this system from six to four states. □

7.3.2 Closing the gap between confluence and ample sets

To show that the differences discussed above are indeed the only differences between confluence and ample sets, we remove them and show that the resulting notions indeed coincide.

As a first step, we precisely prohibit all the liberal aspects of confluence that make the reductions in Figure 7.4(a), 7.4(b) and 7.4(c) work—hence, disallowing 'shortcuts' and confluent transitions relying on differently labelled confluent transitions for the diamond shape to close.

As a second step, we loosen the independence concept of ample sets so that it corresponds better to the more local approach of confluence, allowing ample sets to reduce Figure 7.4(d). Note that we do this safely, i.e., Theorem 7.19 is never compromised during the process, as all these notions will still be confluent in the sense used in that theorem.

Restricted confluence. First of all, we restrict the notion of confluence by strengthening the requirements for two distributions to be equivalent. For this purpose, we introduce a notion called *equivalence up to $\mathcal{T}$-steps* to force commutation of transitions to occur in the diamond structure of independence. This results in a notion of confluence that can no longer reduce Figure 7.4(b).

Definition 7.24 (Equivalence up to $\mathcal{T}$-steps). *Let $M = (S, s^0, A, P, \mathrm{AP}, L)$ be an MDP, $\mathcal{T} \subseteq \Delta_M$ a set of deterministic transitions of M, and let $\mu, \nu \in \mathrm{Distr}_\perp(S)$ be two probability distributions. Then, we say that μ is* equivalent up to $\mathcal{T}$-steps *to ν, denoted by $\mu \rightsquigarrow_\mathcal{T} \nu$, if $\mu, \nu \neq \perp$ and there exists a partitioning $supp(\mu) = \biguplus_{i=1}^n S_i$ of the support of μ and an ordering $supp(\nu) = \langle s_1, \dots, s_n \rangle$ of the support of ν, such that*

$$\forall 1 \leq i \leq n \ . \ \mu(S_i) = \nu(s_i) \wedge \forall s \in S_i \ . \ \exists a \in A \ . \ (s, a, \mathbb{1}_{s_i}) \in \mathcal{T}$$

Example 7.25. Consider the MDP in Figure 7.5, and let

$$\mathcal{T} = \{(s_0, a, \mathbb{1}_{s_1}), (s_2, a, \mathbb{1}_{s_6}), (s_3, a, \mathbb{1}_{s_5}), (s_4, a, \mathbb{1}_{s_5})\}$$

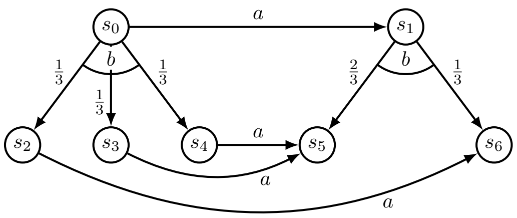


Figure 7.5: An MDP to demonstrate $\rightsquigarrow_{\mathcal{T}}$.

Moreover, let $\mu = P(s_0, b)$ and $\nu = P(s_1, b)$. It now follows that $\mu \rightsquigarrow_{\mathcal{T}} \nu$, by taking the partitioning $\mathit{supp}(\mu) = S_1 \cup S_2$ with $S_1 = \{s_2\}$ and $S_2 = \{s_3, s_4\}$, and the ordering $\mathit{supp}(\nu) = \langle s_6, s_5 \rangle$. Now, indeed $\mu(S_1) = \frac{1}{3} = \nu(s_6)$ and $\mu(S_2) = \frac{2}{3} = \nu(s_5)$. Also, there is a transition in $\mathcal{T}$ connecting s_2 to s_6, and there are transitions in $\mathcal{T}$ connecting s_3 and s_4 to s_5. □

It is easy to see that $\mu \rightsquigarrow_{\mathcal{T}} \nu$ implies that $\mu \equiv_{R^{\mathcal{T}}_{\mu,\nu}} \nu$, with $R^{\mathcal{T}}_{\mu,\nu}$ the smallest equivalence relation such that

$$R^{\mathcal{T}}_{\mu,\nu} \supseteq \{(s,t) \in \mathit{supp}(\mu) \times \mathit{supp}(\nu) \mid \exists a \, . \, (s \xrightarrow{a} t) \in \mathcal{T}\}$$

Hence, we can use equivalence up to $\mathcal{T}$-steps in the definition of confluence without losing correctness.

When symbolic analysis is carried out for ample sets and similar methods, the relations that are extracted are usually assumed symmetric: if a and b are independent, then they do not disable each other. This is much due to the way algorithms for generating them often work (though not always, see for instance [HKQ11]). The above stronger version of up-to-equivalence features this same symmetry.

In addition to strengthening equivalence of distributions, we also restrict confluence by requiring reducible actions to be mimicked as well. Hence, we omit the second item of Definition 7.16; the practical interpretation is similar to the one mentioned above. After this change, no reduction is possible anymore in the model of Figure 7.4(a).

Definition 7.26 (Restricted probabilistic confluence). *Let $M = (S, s^0, A, P, \mathrm{AP}, L)$ be an MDP. Then, a set $\mathcal{T} \subseteq \Delta_M$ is* restrictedly probabilistically confluent *if all its transitions have a reducible action, and for all $s \xrightarrow{a}_{\mathcal{T}} t$ and all transitions $s \xrightarrow{b} \mu$ ($b \neq a$), it holds that*

- $\exists \nu \in \mathrm{Distr}(S) \, . \, t \xrightarrow{b} \nu \wedge \mu \rightsquigarrow_{\mathcal{T}} \nu \wedge \big((s \xrightarrow{b} \mu) \in \mathcal{T} \implies (t \xrightarrow{b} \nu) \in \mathcal{T}\big)$.

We call a reduction function with an underlying restricted confluent set a restricted confluence reduction function.

We add the restriction $b \neq a$, to ensure that confluent transitions still commute with themselves. Since in the original definition every confluent transition also already commuted with itself, this does not weaken the concept. Hence, Definition 7.26 is a true restriction of Definition 7.16.

Finally, we saw in Figure 7.4(c) that confluence allows reducible transitions to be mimicked by actions with different names. If we want confluence reduction and ample sets to coincide, we need to make sure that actions are not allowed to rely on other actions to 'close their diamonds'. From the point of view of symbolic analysis, this restriction matches the practical heuristics used for ample set reduction: only pairwise analysis of actions is required, and the algorithms for generating ample sets or similar notions mostly rely on these sort of binary relations. For this purpose we introduce the concept of *action-separability*, requiring that each subset of $\mathcal{T}$ that can be obtained by only keeping one specific action, is confluent. That way, confluence reduction functions such as the one in Figure 7.4(c) are not allowed anymore.

Definition 7.27 (Action-separable confluence). *Let $M = (S, s^0, A, P, \mathrm{AP}, L)$ be an MDP, then a confluent set $\mathcal{T} \subseteq \Delta_M$ of transitions of M is* action-separable *if for every action $a \in A$ the subset*

$$\mathcal{T}_a = \{(s, a, \mu) \in S \times \{a\} \times \mathrm{Distr}(S) \mid (s, a, \mu) \in \mathcal{T}\}$$

of a-labelled confluent transitions is confluent (so possibly empty).

A confluence reduction function $T\colon S \to \mathscr{P}(A)$ is action-separable *if its underlying confluent set $\mathcal{T}$ is.*

Relaxing ample sets. Independence is judged by the ample set constraints in a global manner, whereas confluence deals with the notion of equivalent distributions, which is much more local.

To make confluence and ample sets coincide, independence should also be determined locally, i.e., given a state, dependency of a and b makes a difference only in parts of the MDP that can be reached without executing the ample action first. This corresponds to the fact that confluence only puts restrictions on commutation of actions before a confluent transition.

The practical side of this change lies in dynamic analysis. We can, for instance, initially consider that a and b are dependent due to symbolic analysis. However, after finishing exploring some part of the possible states following a state s, we may come to the conclusion that the dependency never manifests anywhere where a has not been executed yet, and thus declare a independent of b locally in s. This idea originates from [KP92] and [GP93], and also corresponds exactly to the way the stubborn set definitions (see, e.g., [Val96]) deal with dependency in the non-probabilistic case: only executions starting from the current state that do not include any stubborn actions, are relevant from the point of view of commutativity.

To define local independence, let $R_a(s) \subseteq S$ be the set of states s' such that $s \xrightarrow{c_1 \cdots c_n} s'$ for some path where there is no i such that $c_i = a$.

Definition 7.28 (Local independence). *Given an MDP $M = (S, s^0, A, P, \mathrm{AP}, L)$, a state $s \in S$, and two actions $a, b \in A$, we say that a is* independent of b at s *if* $a \neq b$ *and for every state* $s' \in R_a(s)$ *such that* $\{a, b\} \subseteq en(s')$ *the following conditions hold:*

- *If* $s^* \in supp(P(s', a))$, *then* $b \in en(s^*)$ *(and symmetrically);*
- *We have*

$$\sum_{s^* \in S} P(s', a)(s^*) \cdot P(s^*, b)(t) = \sum_{s^* \in S} P(s', b)(s^*) \cdot P(s^*, a)(t)$$

for every $t \in S$.

If a *is not independent of* b *at* s*, we say that it is* dependent of b at s.

Note that this definition coincides with the original definition of independence, except that the conditions only have to hold for all states in $R_a(s)$ instead of all states in S.

Also note that local (in)dependence is not a symmetric relation. For a to be independent of b at s we only consider the states in $R_a(s)$; this is in general a set different from $R_b(s)$.

Example 7.29. In Example 7.12 we noticed that the actions $task_5$ and $task_6$ in Figure 7.1 were not independent, since there is a state (namely s_{11}) in which they can disable each other. However, taking local independence, we see that $R_{\text{task}_5}(s_8) = \{s_8, s_{10}\}$ and $R_{\text{task}_6}(s_8) = \{s_8, s_9\}$, and we can verify that the independence conditions are satisfied by all of these states. Hence, $task_5$ is independent of $task_6$ at s_8 and also $task_6$ is independent of $task_5$ at s_8. Therefore, if the ample set conditions used local independence instead of global independence, it would be allowed to take either $task_5$ or $task_6$ as an ample set for s_8. □

Under the local dependency condition, we can now relax the ample set conditions slightly.

Definition 7.30 (Relaxed ample sets). *A set* $A(s)$ *is a* relaxed ample set *if it meets the criteria of Definition 7.13, except that A2 is replaced by the following condition:*

*A2** *For every path* $s \xrightarrow{a_1 a_2 \ldots a_n b} t$ *in* M *such that* $b \notin A(s)$ *and some* $a \in A(s)$ *is dependent on* b *at* s*, there exists an* $1 \leq i \leq n$ *such that* $a_i \in A(s)$.

Comparison. Our most important theorem of this chapter is now ready to be proven. It says that our restriction of confluence reduction and relaxation of ample set reduction indeed coincide. Hence, the differences we identified earlier indeed precisely characterise the gap between the two notions.

Theorem 7.31. *Let* $M = (S, s^0, A, P, \text{AP}, L)$ *be an MDP. Then,* $T \colon S \to \mathscr{P}(A)$ *is an acyclic action-separable restricted confluence reduction function if and only if* T *is a relaxed ample set reduction function.*

Note that an acyclic action-separable restricted confluence reduction function is just a special case of an acyclic confluence reduction function, as used in Theorem 7.19, so it too preserves probabilistic visible bisimulation. Since relaxed ample set reduction functions coincide with confluence now, we immediately have the result that they too still preserve probabilistic visible bisimulation.

As all of our propositions and theorems hold just as well in case there are no probabilistic transitions, and the probabilistic notions of ample set reduction and confluence reduction in that case reduce to their non-probabilistic variants (except that we preserve divergences), the following corollary is also immediate.

Corollary 7.32. *In the non-probabilistic setting, confluence reduction is able to reduce more than ample set reduction. With some adjustments (as in Definitions 7.24, 7.26, 7.27, 7.28 and 7.30), the two notions coincide.*

7.3.3 Practical implications

To reduce the number of states of an MDP even more than when applying a reduction function, we can again apply the representation map approach as presented in Chapter 6. We demonstrate this procedure once more on the running example of this chapter.

Example 7.33. Consider again the MDP in Figure 7.1 and the probabilistically confluent set provided in Example 7.18. As stated there,

$$\begin{aligned}\mathcal{T} = \{&(s_0, task_1, \mathbb{1}_{s_1}), (s_2, task_1, \mathbb{1}_{s_3}), (s_3, task_3, \mathbb{1}_{s_4}),\\ &(s_5, task_3, \mathbb{1}_{s_6}), (s_8, task_5, \mathbb{1}_{s_9}), (s_{10}, task_5, \mathbb{1}_{s_{11}})\}\end{aligned}$$

In the absence of cycles in $\mathcal{T}$, there is only one possible representation map under $\mathcal{T}$:

$$\begin{array}{lll}\varphi_{\mathcal{T}}(s_0) = s_1 & \varphi_{\mathcal{T}}(s_2) = s_4 & \varphi_{\mathcal{T}}(s_3) = s_4\\ \varphi_{\mathcal{T}}(s_5) = s_6 & \varphi_{\mathcal{T}}(s_8) = s_9 & \varphi_{\mathcal{T}}(s_{10}) = s_{11}\end{array}$$

and $\varphi_{\mathcal{T}}(s) = s$ for all other states s. The quotient MDP under this representation map is shown in Figure 7.6. □

The representation map approach is not only useful for confluence reduction, but also for ample set reduction. After all, from Theorem 7.21 we know that every ample set reduction is a confluence reduction. The representation map

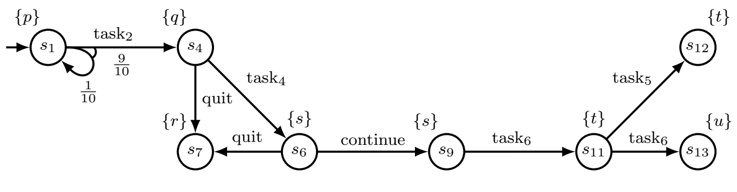


Figure 7.6: A quotient MDP under a representation map.

approach serves as an alternative implementation of the cycle condition of ample sets or the acyclicity requirement used earlier in this chapter. The cycle condition is satisfied in the sense that the quotient MDP never indefinitely ignores any behaviour of the original MDP.

7.4 Contributions

We redefined confluence reduction in an MDP-based setting, enabling a comparison to probabilistic partial order reduction based on ample sets in branching time. We proved that every nontrivial ample set can be mimicked by a confluent set, and that in some cases reductions are possible using confluence but not using ample sets. Therefore, at least in theory confluence reduction is able to reduce more than the ample set method. We also showed the exact way in which confluence and ample sets have to be modified for the two notions to coincide. This way, the gap between their expressivity is identified precisely. These results hold for the non-probabilistic variants of the two reduction techniques as well.

Our observation that probabilistic ample set reduction can be mimicked by probabilistic confluence reduction has additional implications, some of which are highly practical. One such implication is that the use of a representation map for reduced state space generation, already applied earlier in combination with confluence reduction, can also be applied for partial order reduction as a substitute for the cycle condition.

As both ample sets and confluence are detected symbolically on the language level, the quality of the heuristics applied there will decide which notion works best in practice. The results in this chapter already strengthen our theoretical understanding of the two methods, and this is independent of the heuristics that are applied. Obviously, no matter how such heuristics are improved, the results in this chapter will remain valid. We showed that at least one of the advantages of confluence over POR can indeed be exploited by our heuristics.

To complement the theoretical results presented in this chapter, the next chapter will provide a comparison of confluence reduction and partial order reduction from a practical point of view. Although the practical difference may be limited with regard to the number of additional reductions, we will show that the way confluence deals with independence allows more systems to be subjected to statistical model checking: a significant improvement.

We conjecture that the ideas presented in this chapter can easily be used to define a notion of branching time partial order reduction for MAs, based on our notion of confluence for MAs presented in Chapter 6. Then, a result similar to Theorem 7.21 might be applied to prove its correctness.

CHAPTER 8

Confluence Reduction in Statistical Model Checking

A Practical Comparison to Partial Order Reduction

"*There's no sense in being precise when you don't even know what you're talking about.*"

John von Neumann

As mentioned before, model checking is subject to the state space explosion problem, with probabilistic model checking being particularly affected due to its additional numerical complexity. Several techniques have been introduced to stretch the limits of model checking, while preserving its basic nature of performing state space exploration to obtain results that unconditionally hold for the entire state space. Among these techniques are the basic reduction techniques we introduced in Section 4.5 and the dead variable reduction technique from Chapter 5, as well as confluence reduction and partial order reduction (POR) as discussed in Chapters 6 and 7. As discussed previously, partial order reduction and confluence reduction both work by selecting a subset of the transitions of a model—and thus a subset of the reachable states—in a way that ensures that the reduced system is equivalent to the complete system.

A much different approach for probabilistic models is statistical model checking (SMC) [HLMP04, LDB10, YS02]: instead of exploring—and storing in memory—the entire state space, or even a reduced version of it, discrete-event simulation is used to generate traces through the state space. This comes at constant memory usage and thus circumvents the state space explosion entirely, but can only approximate probabilities of interest. Statistical methods such as sequential hypothesis testing are then used to make sure that the *probability* of returning the wrong result is below a certain threshold. As a simulation-based approach, however, SMC is in principle limited to fully stochastic models such as Markov chains [Har10].

Previously, an approach based on POR was presented [BFHH11] to extend SMC and simulation to the nondeterministic model of Markov decision processes (MDPs). In that approach, simulation proceeds as usual until a nondeterministic choice is encountered; then, an on-the-fly check is performed to find a singleton subset of the available transitions that satisfies the *ample set* conditions of linear-time probabilistic POR [BGC04, DN04]. If such an ample set is found, simulation can continue according to it, with the guarantee that ignoring the other transitions does not affect the verification results—that is, the nondeterminism was *spurious*. Yet, the ample set conditions are based on the notion of *independence* of actions, which can in practice only feasibly be checked on a symbolic/syntactic level (using conditions such as J1 and J2 in [BFHH11]). This limits the approach to resolving spurious nondeterminism only when it results from the *interleaving* of behaviours of concurrently executing deterministic components.

It is absolutely vital for the search for a valid singleton subset to succeed in the approach discussed above: one choice that cannot be resolved means that the entire analysis fails and SMC cannot safely be applied to the given model at all. Hence, any additional reduction power is highly welcome. In this chapter, we therefore introduce an alternative approach that uses confluence reduction instead of POR. We demonstrate that the findings of Chapter 7—which showed that confluence is in theory more powerful than branching time POR—indeed enable reductions that POR did not allow. That way, some models can now be analysed using SMC whereas this was not possible before.

We note that a confluence-based approach is not better in all cases, though, since the approach in [BFHH11] in based on *linear time* POR with an added requirement of having singleton ample sets. Compared to branching time POR, this notion is more liberal in the sense that it allows ample actions to be probabilistic. As confluence requires deterministic transitions, this also gives the POR-based approach an advantage—at the cost of preserving less properties, though $\text{LTL}_{\setminus X}$ is still more than enough for SMC.

Except for potentially reducing more in some cases, an additional advantage of confluence reduction is that it is more easily implemented on the level of the concrete state space alone—without any need to go back to the symbolic/syntactic level for an independence check (as was done for POR). As opposed to the approach in [BFHH11], it thus allows even spurious nondeterminism that is internal to components to be ignored during simulation. Of course, models containing non-spurious nondeterminism can still not be dealt with.

Our approach. While the notion of confluence from Chapter 7 is based on MDPs too, it is not yet suitable for use during SMC. After all, it required *actions* instead of *transitions* to be stuttering and deterministic. In SMC we cannot check this explicitly, as the complete state space is not available. Moreover, we especially do not want to apply syntactic heuristics, to make the technique as powerful as possible. Hence, we change the requirement of stuttering and deterministic actions back to stuttering and deterministic transitions—basically again taking the same approach as in Chapter 6. In addition to this alteration, we also relax the definition of confluence by allowing visible transitions to be mimicked by differently-labelled transitions. After all, simulation works with a fully composed, closed system, where action labels have become irrelevant. We thus achieve more reduction/detection power at no computational cost; yet, this adapted notion of confluence still preserves PCTL^* formulae [BK08] without the *next* operator.

We introduce an algorithm for detecting our new notion of probabilistic

approach	nondeterminism	probabilities	memory	error bounds
POR-based [BFHH11]	spurious interleavings	max = min	$\underline{s \ll n}$	$\underline{\text{unchanged}}$
confluence-based	spurious	max = min	$\underline{s \ll n}$	$\underline{\text{unchanged}}$
learning [HMZ+12]	$\underline{\text{any}}$	max only	$s \to n$	convergence

Table 8.1: SMC approaches for nondeterministic models (with n states).

confluence on a concrete state space. The algorithm is inspired by, but different from, the one given in [GvdP00]; in particular, it does not require initial knowledge of the entire state space and can therefore be used on-the-fly during simulation.

Finally, we evaluate the new confluence-based approach to SMC on a set of three representative examples using an implementation within the modes statistical model checker [BHH12] for the MODEST modelling language [BDHK06]. We clearly identify its strengths and limitations. Since the previous POR-based approach has also been implemented in modes, we compare the two in terms of reduction power and, on the one case that can actually be handled by the POR-based implementation as well, performance. In this case, confluence reduction turns out to be somewhat faster (15%–40%), but that may well be due to a suboptimal implementation of the POR check.

Related work. Aside from [BFHH11] and an approach that focuses on planning problems and infinite-state models [LP12], the only other solution to the problem of nondeterminism in SMC that we are aware of is recent work by Henriques et al. [HMZ+12]. They use reinforcement learning, a technique from artificial intelligence, to actually learn the resolutions of nondeterminism (by memoryless schedulers) that *maximise* probabilities for a given bounded LTL property. While this allows SMC for models with arbitrary nondeterministic choices (not only spurious ones), scheduling decisions need to be stored for every *explored* state. Memory usage can thus be as in traditional model checking, but is highly dependent on the structure of the model and the learning process. As the number of runs of the algorithm increases, the answer it returns will converge to the actual result, but definite error probabilities are not given. The approaches based on confluence and POR do not introduce any additional overapproximation and thus have no influence on the usual error bounds of SMC. Table 8.1 gives a condensed overview of the three approaches (where we measure memory usage in terms of the maximal number of states s stored at any time; see Section 8.5 for concrete values).

Organisation of the chapter. Section 8.1 first provides an informal introduction to the concept of statistical model checking. Then, we present the preliminaries for this chapter in Section 8.2. Section 8.3 provides a slightly altered definition of confluence, specifically suitable for use during SMC. Section 8.4 introduces our algorithm for on-the-fly detection of this new notion of confluence, followed by an evaluation by means of three case studies in Section 8.5. Finally, Section 8.6 concludes by summarising the contributions of this chapter.

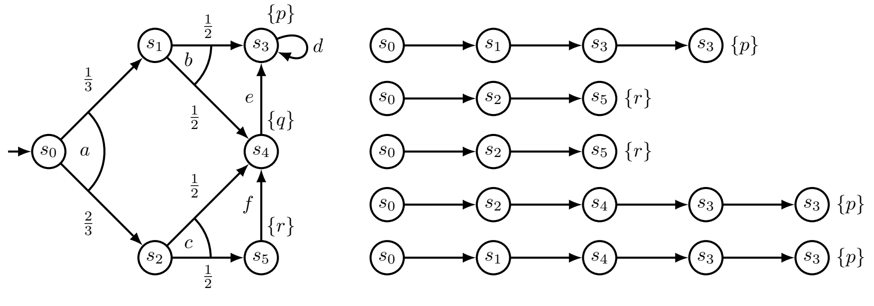
Origins of the chapter. This chapter was written in collaboration with Arnd Hartmanns from the Dependable Systems and Software group at Saarland University, Saarbrücken, Germany. The author contributed most of the theoretical part and the technical details of the algorithm, while Arnd Hartmanns made the implementation in the Modest Toolset and performed all the case studies along with their analysis. This work was published in the proceedings of the *5th NASA Formal Methods Symposium* (NFM) [HT13b] and a corresponding technical report [HT13c].

8.1 Statistical model checking in a nutshell

Quantitative model checking is normally concerned with the generation of a complete state space. All states are stored, and upon the arrival at a new state it is compared to all previously visited states to see if it is already present in the state space. After generating the complete model, numerical algorithms are employed to compute for instance the probability of eventually reaching a state satisfying a certain property. These techniques aim at computing the actual probability of such properties, at the cost of high memory usage.

8.1.1 Basics of statistical model checking

Statistical model checking does not generate entire state spaces. On the contrary, in principle it only stores a single state in memory. Each step of the procedure is concerned with obtaining a next state, often by means of a probabilistic choice. This way, a single *trace* through the system is obtained. The procedure requires only constant memory, since nothing is stored except for the state visited last. While generating traces through a model in this way, a statistical model checker basically just counts how many of them satisfy the property under consideration—this technique is often called Monte Carlo sampling. Upon completing a large number of traces, it can then simply report on the probability of this property by dividing the number of traces satisfying it by the total number of traces that were executed.



(a) A DTMC.

(b) Five traces through a DTMC.

Figure 8.2: Statistical model checking of a simple DTMC.

Example 8.1. Consider the MDP in Figure 8.2(a). Since it does not contain any nondeterminism, it is actually a DTMC. Suppose now that we want to investigate the probability of eventually reaching a state that satisfies the atomic proposition r. Analytically, this is easily seen to be $\frac{2}{3} \cdot \frac{1}{3} = \frac{1}{3}$. For larger systems, though, this may not be so easy.

When performing statistical model checking, we start in s_0 and throw a three-sided die to decide how to move on. Sometimes we continue from s_1, sometimes we continue from s_2. From the second state we visit, again the probabilistic choice is resolved by means of a probabilistic experiment, and so on. Upon reaching s_5 we can stop, as a state satisfying r is visited. Upon reaching s_3 we can also terminate, after noticing that from there it is only still possible to continuously loop through a state not satisfying r.

Executing this procedure five times, we may end up with the traces depicted in Figure 8.2(b). Based on this information, we can conclude that the probability of eventually reaching a state satisfying atomic proposition r is about $\frac{2}{5}$. Clearly, for a larger number of traces this probability will converge to the actual probability of $\frac{1}{3}$. □

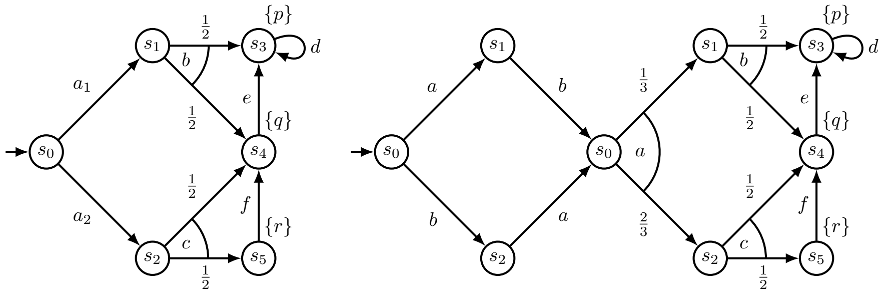
In addition to providing probabilities, statistical model checking can also provide *confidence intervals*—after a large number of runs a statistical model checker may for instance conclude that the MDP in Figure 8.2(a) has a probability of reaching a state satisfying r within the interval $[0.32, 0.34]$ with 95% confidence.

Note that for more complicated systems, trace length is an important concern. For bounded properties (i.e., "a packet should be received before a given timeout"), runs should be at least as long as needed to decide the property under consideration. For unbounded properties (i.e., "a packet should be received eventually"), we can only decide with certainty if the property holds upon either (1) reaching a state that satisfies it, (2) reaching a deadlock state or (3) reaching a cycle of states that cannot be escaped. A statistical model checker may be configured to abort if neither of these conditions is satisfied after a predefined maximum number of steps. Alternatively, we could of course still compute one-sided confidence limits to conclude for instance that the probability of eventually receiving a packet is larger than 0.8 with at least 95% confidence.

8.1.2 Dealing with nondeterminism

Statistical model checking is based on the idea that every trace through a model is associated with a probability. In the presence of nondeterminism, though, this assumption is invalidated: dice cannot decide how to continue from a state that has several successors without an accompanying probability distribution.

Example 8.2. Consider the MDP depicted in Figure 8.3(a), and assume that we again intend to compute the probability of eventually satisfying atomic proposition r. The system starts with a nondeterministic choice, that actually influences the property under consideration. After all, moving up via the a_1-transition makes it impossible to ever reach state s_5. When taking the a_2-transition, on the other hand, the probability of reaching s_5 is $\frac{1}{2}$. Hence, the actual probability of ever reaching a state satisfying r is within the interval $[0, \frac{1}{2}]$.



(a) An MDP. (b) An MDP suitable for statistical model checking.

Figure 8.3: Statistical model checking of MDPs.

Statistical model checking would not be able to come up with this interval. At best, it could resolve the nondeterministic choice by means of a uniform probabilistic choice. This would result in a probability around $\frac{1}{4}$. However, if in practice some nondeterministic choices are *not* resolved in a uniform manner, this result is not trustworthy anymore. $\square$

Although nondeterministic choices *may* induce untrustworthy results, this is not necessarily the case. It could be that the way a nondeterministic choice is resolved does not influence the property under consideration: it is *spurious*. If all nondeterministic choices in an MDP are in fact spurious, the interval that is obtained from traditional quantitative model checking is a single point. Hence, statistical model checking would also provide a valid result—it can just resolve each nondeterministic choice in an arbitrary way.

Example 8.3. Consider the MDP in Figure 8.3(b). Clearly, the nondeterministic choice in the beginning does not influence the probability of reaching the state s_5. Hence, we could just instruct a statistical model checking to always take the a-transition to s_1, and then continue as usual. $\square$

The question remains how to know upfront whether a nondeterministic choice does not influence the property under consideration. A safe underapproximation is to check whether it does not influence *any* property we could imagine. Actually, this is precisely what confluence reduction is all about: it checks if certain transitions could be given priority without altering the system's behaviour[1]. Hence, if one of the transitions of a nondeterministic choice can be shown to be confluent, it can be given priority without influencing the property under consideration.

This chapter introduces more precisely how to apply confluence reduction in the context of statistical model checking, enabling the memory-efficient analysis

[1]For confluence to be checked, part of the state space has to be explored. After all, we need to verify whether all neighbouring transitions are mimicked, as discussed in Chapter 6. Hence, the addition of confluence reduction to statistical model checking introduces the need to store more than one state in memory. The case studies in Section 8.5 will show that this number depends on the model structure, but in general is rather small.

of the subset of MDPs that contain only confluent nondeterministic choices.[2]

8.2 Preliminaries

Our investigation of confluence reduction for statistical model checking takes place in the context of the Modest Toolset. This framework is based on Markov decision processes (MDPs), almost identical to the variant defined in the previous chapter (Definition 7.1), but slightly different in the sense that a state is now allowed to have multiple outgoing transitions with the same label.

For convenience of the reader—though at the cost of some repetition—we present the altered definition again instead of only explaining the (minor) differences.

Definition 8.4 (MDPs). *A* Markov decision process (MDP) *is a tuple* $M = (S, s^0, A, P, \mathrm{AP}, L)$*, where*

- S *is a countable set of* states*;*
- $s^0 \in S$ *is the* initial state*;*
- $A \subseteq Act$ *is a finite set of* actions*;*
- $P \subseteq S \times A \times \mathrm{Distr}(S)$ *is the* probabilistic transition relation*;*
- AP *is a finite set of* state labels*;*
- $L \colon S \to \mathscr{P}(\mathrm{AP})$ *is the* state-labelling function.

Given a state $s \in S$*, we define its set of* enabled transitions

$$en(s) = \{(s, a, \mu) \in \{s\} \times A \times \mathrm{Distr}(S) \mid s \xrightarrow{a} \mu\}$$

We will use S_M*,* A_M*, …, to refer to the components of an MDP* M*. If the MDP is clear from the context, these subscripts are omitted.*

Note that the current definition of MDPs precisely coincides with the definition of MAs (Definition 3.5) when restricting MAs to finite sets of actions and state labels and an empty set of Markovian transitions. Hence, all notations introduced for MAs can directly be applied to these MDPs. Also, we reuse the concepts presented in Definition 7.4 on determinism and stuttering.

With Modest we work in a state-based verification setting where properties only refer to the atomic propositions of states. The action labels are solely meant for synchronisation during parallel composition. Since SMC is applied to the result of parallel composition we consider closed systems only, and therefore we can ignore the action labels. We do care about whether or not transitions change the observable behaviour of the system, i.e., the atomic propositions. As before, transitions not changing this behaviour are called *stuttering*. In this chapter, we write $s \xrightarrow{\tau} \mu$ to indicate that a transition is stuttering. Transitions labelled by a letter different from τ can be either stuttering or not.

[2]In hindsight, if all nondeterministic choices turn out to be spurious, we could just as well have taken the old approach of just resolving nondeterministic choices in a uniform probabilistic manner. This would indeed have provided the same result. However, the old approach would not have shown us that this result indeed is a valid result under *all* possible schedulers. Our approach based on confluence does provide us with this information.

As in the previous chapter, we again apply *reduction functions* to indicate which transitions to keep and which to omit. Whereas in Definition 7.6 we could specify the outgoing transitions of a state by listing their actions, this is now not possible anymore due to the fact that there may be multiple transitions with the same label. Hence, we alter the concept of a reduction function slightly, making it select from the outgoing *transitions* of a state instead of its actions.

Definition 8.5 (Reduction functions). *Given an MDP* $M = (S_M, s^0_M, A, P_M, \mathrm{AP}, L_M)$*, a* reduction function *is any function* $R \colon S_M \to \mathscr{P}(P_M)$ *such that* $R(s) \subseteq en(s)$ *for every* $s \in S_M$*. Given a reduction function* R*, the* reduced MDP for M with respect to R *is the minimal MDP* $M_R = (S_R, s^0_R, A, P_R, \mathrm{AP}, L_R)$ *such that* $s^0_R = s^0_M$ *and*

- *if* $s \in S_R$ *and* $(s, a, \mu) \in R(s)$*, then* $(s, a, \mu) \in P_R$ *and* $supp(\mu) \subseteq S_R$*;*
- $L_R(s) = L_M(s)$ *for every* $s \in S_R$*,*

where minimal should be interpreted as having the smallest set of states and the smallest set of transitions.

Given a reduction function R *and a state* $s \in S_R$*, we say that* s *is a* reduced state *if* $R(s) \neq en(s)$*. All outgoing transitions of a reduced state are called* nontrivial transitions*. We say that a reduction function is* acyclic *if there are no cyclic paths when only nontrivial transitions are considered.*

8.3 Confluence for statistical model checking

In this chapter we are dealing with a state-based context; only the atomic propositions that are assigned to each state are of interest. Therefore, we base our definition of confluence on the one introduced in Chapter 7. We adjust it slightly to adapt to the setting of SMC.

8.3.1 Confluence sets for statistical model checking

In this chapter, we apply the following definition of confluence.

Definition 8.6 (Probabilistic confluence). *Let* $M = (S, s^0, A, P, \mathrm{AP}, L)$ *be an MDP, then a subset* $\mathcal{T} \subseteq P$ *of transitions from* M *is* probabilistically confluent *if it only contains deterministic stuttering transitions, and for all* $s \xrightarrow{a}_{\mathcal{T}} t$ *and all transitions* $s \xrightarrow{b} \mu$*, either*

- $\exists c \in A, \nu \in \mathrm{Distr}(S) \,.\, t \xrightarrow{c} \nu \wedge \mu \equiv_{R^{\mathcal{T}}_{\mu,\nu}} \nu \wedge \big((s \xrightarrow{b} \mu) \in \mathcal{T} \implies (t \xrightarrow{c} \nu) \in \mathcal{T}\big)$*, or*
- $\mu = \mathbb{1}_t$*,*

with $R^{\mathcal{T}}_{\mu,\nu}$ *the smallest equivalence relation such that*

$$R^{\mathcal{T}}_{\mu,\nu} \supseteq \{(s,t) \in supp(\mu) \times supp(\nu) \mid \exists a \,.\, (s \xrightarrow{a} t) \in \mathcal{T}\}$$

A transition is probabilistically confluent *if there exists a probabilistically confluent set that contains it.*

Compared to Definition 7.16, this definition is more liberal in two aspects. We discuss them here, and explain why the correctness arguments are not influenced by these changes.

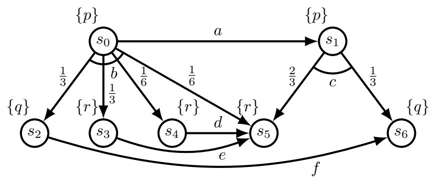
First, not necessarily $b = c$ anymore—previously this was needed to preserve the rather general notion of probabilistic visible bisimulation. Equivalent systems according to this notion preserve state-based as well as action-based properties. However, in our current setting the actions are only for synchronisation of parallel components, and have no purpose anymore in the final model. Therefore, we can just as well rename them all to a single action. Then, if a transition is mimicked, the action will be the same by construction and our definition coincides with the old one—so, probabilistic visible bisimulation is still preserved. Even easier, we chose to omit the required accordance of action names altogether. Although this implies that our notion of confluence does not preserve probabilistic visible bisimulation anymore, this does not influence the state-based properties that may be analysed.

Second, whereas Definition 7.16 required the *actions* of confluent transition to be stuttering and deterministic, we now only require the confluent transitions to be invisible and deterministic *themselves*. Similarly, if $\mu = \mathbb{1}_t$ we do not care anymore whether $b \in A_{\text{red}}$, since we already do know that the transition $s \xrightarrow{b} \mu$ is stuttering and deterministic in this case (since $s \xrightarrow{a} t$ is). The reason for this change is that during simulation we only know part of the state space. However, it is also not needed for correctness, as a local argument about mimicking behaviour until some joining point can clearly never be broken by transitions after this point. Indeed, Definition 6.7 also only required confluent transitions to be invisible and deterministic themselves.

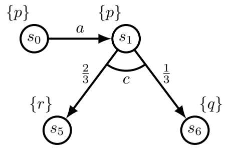
Remark 8.7. Note that, just like in Definition 7.16, we do not consider a confluence classification anymore. Again, this can be seen as a special case of the old definition, having a confluence classification with one group that precisely contains all transitions that we want to denote confluent. The removal of an explicit confluence classification does make it necessary to be cautious when taking the union of two confluent sets; this is not necessarily again a confluent set (as discussed in Section 6.5.4). □

In contrast to probabilistic POR—as investigated in detail in the previous chapter—confluence also allows mimicking by differently-labelled transitions, commutativity in triangles instead of diamonds, and local instead of global independence. Additionally, its coinductive definition is well-suited for on-the-fly detection, as we show in this chapter. However, as confluence preserves branching time properties, it cannot reduce probabilistic interleavings, a scenario that can be handled by the linear time notion of POR used in [BFHH11].

Example 8.8. To illustrate Definition 8.6, consider Figure 8.4(a). Let $\mathcal{T}$ be the set consisting of the transitions labelled by a, d, e and f. Note that these transitions indeed are all deterministic and stuttering. We denote by μ the probability distribution associated with the b-transition from s_0, and by ν the one associated with the c-transition from s_1.



(a) Original system.



(b) Reduced system.

Figure 8.4: An MDP to demonstrate confluence reduction.

We find $R^{\mathcal{T}}_{\mu,\nu} \supseteq \{(s_2, s_6), (s_3, s_5), (s_4, s_5)\}$, and therefore

$$R^{\mathcal{T}}_{\mu,\nu} = Id \cup \{(s_2, s_6), (s_6, s_2), (s_3, s_4), (s_4, s_3), (s_3, s_5), (s_5, s_3), (s_4, s_5), (s_5, s_4)\}$$

with Id the identity relation. Hence, $R^{\mathcal{T}}_{\mu,\nu}$ partitions the state space into the four equivalence classes $\{s_0\}, \{s_1\}, \{s_2, s_6\}$ and $\{s_3, s_4, s_5\}$. We find that $\mu \equiv_{R^{\mathcal{T}}_{\mu,\nu}} \nu$, since

$$\begin{aligned} \mu(\{s_0\}) &= \nu(\{s_0\}) = 0 \\ \mu(\{s_1\}) &= \nu(\{s_1\}) = 0 \\ \mu(\{s_2, s_6\}) &= \nu(\{s_2, s_6\}) = \tfrac{1}{3} \\ \mu(\{s_3, s_4, s_5\}) &= \nu(\{s_3, s_4, s_5\}) = \tfrac{2}{3} \end{aligned}$$

Based on the above, we can show that $\mathcal{T}$ is a valid confluent set according to Definition 8.6. First, all its transitions are indeed stuttering and deterministic. Second, for the transitions from s_2, s_3 and s_4, nothing interesting has to be checked. After all, from their source states there are no other outgoing transitions, and every transition satisfies the condition $\mu = \mathbb{1}_t$ for itself. For $s_0 \xrightarrow{a} \mathbb{1}_{s_1}$, we do need to check if $s_0 \xrightarrow{b} \mu$ can be mimicked. Indeed, there is a transition $s_1 \xrightarrow{c} \nu$, and as we saw above $\mu \equiv_{R^{\mathcal{T}}_{\mu,\nu}} \nu$, as required. □

8.3.2 Confluence reduction

We now define confluence reduction functions. As before, such a function always chooses to either fully explore a state, or to only explore one of its outgoing confluent transitions.

Definition 8.9 (Probabilistic confluence reduction). *For an MDP $M = (S, s^0, A, P, \mathrm{AP}, L)$, a reduction function $T \colon S \to \mathscr{P}(P)$ is a* confluence reduction function *for M if there exists some probabilistically confluent set $\mathcal{T} \subseteq P$ such that, for every $s \in S$ either*

- *$T(s) = en(s)$, or*
- *$T(s) = \{(s, a, \mathbb{1}_t)\}$ for some $a \in A$ and $t \in S$ such that $(s, a, \mathbb{1}_t) \in \mathcal{T}$.*

In such a case, we also say that T is a confluence reduction function *under $\mathcal{T}$.*

Note the resemblance with Definition 7.17—the only difference is that we now select between the transitions instead of the actions of each state.

As discussed before in earlier chapters, confluent transitions may be taken indefinitely, ignoring the presence of other actions—the *ignoring problem* [EP10]. It was dealt with by the cycle condition of the ample set method of POR, and by the requirement of having an acyclic reduction function in the confluence reduction technique introduced in the previous chapter. In the current context, we again deal with it by requiring the reduction function to be acyclic. Acyclicity can be checked during simulation and statistical model checking in the same way as was done for POR in [BFHH11]: for some predefined constant l, always check whether in the last l steps at least one state was fully explored (i.e., the state already contained only one outgoing transition)[3].

Example 8.10. For the system of Figure 8.4(a), we already found a valid confluent set. Based on this set, we can define the reduction function T given by $T(s_0) = \{(s_0, a, \mathbb{1}_{s_1})\}$ and $T(s) = en(s)$ for every other state s. That way, the reduced system is given by Figure 8.4(b).

Note that the two models indeed share the same properties, such as that the (minimum and maximum) probability of eventually observing r is $\frac{2}{3}$. □

Confluence reduction preserves $\text{PCTL}^*_{\setminus X}$, and hence basically all interesting quantitative properties (including $\text{LTL}_{\setminus X}$, as was preserved in [BFHH11]).

Theorem 8.11. *Let M be an MDP, $\mathcal{T}$ a confluent set of its transitions and T an acyclic confluence reduction function under $\mathcal{T}$. Let M_T be the reduced MDP. Then, M and M_T satisfy the same $PCTL^*_{\setminus X}$ formulae.*

8.4 On-the-fly detection of probabilistic confluence

Non-probabilistic confluence was first detected directly on concrete state spaces to reduce them modulo branching bisimulation [GvdP00]. Although the complexity was linear in the size of the state space, the method was not very useful: it required the complete unreduced state space to be available, which could already be too large to generate. Therefore, two directions of improvements were pursued.

The first idea was to detect confluence on higher-level process-algebraic system descriptions, as first done in [Blo01, BvdP02] for labelled transition systems and generalised to the probabilistic setting of MAs (and hence MDPs) in this thesis. The other direction was to use the ideas from [GvdP00] to on-the-fly detect non-probabilistic weak or strong confluence [MW12, PLM03] during state space generation. These techniques are based on boolean equation systems and have not yet been generalised to the probabilistic setting.

We present a novel on-the-fly algorithm that works on concrete probabilistic state spaces and does not require the unreduced state space, making it perfectly applicable during simulation for statistical model checking of MDPs. Given

[3]As mentioned in [BFHH11], the value of l does not impact the performance at all; it just decides how long to continue before giving up on satisfying the cycle condition and terminating. By default, we therefore use a high value for l, e.g., $l = 1000$.

a specific transition in the model it underapproximates whether or not that transition is confluent. To do so, it explores a (generally small) part of the model. For each state having more than one transition, we apply this algorithm to see if at least one of these transitions is confluent. If so, we continue along this transition; otherwise, SMC can only abort.

8.4.1 Detailed description of the algorithm

Our algorithm is presented on page 193. It assumes an implicit underlying state space, so that it at least can request the state labelling $L(s)$ of a given state s and is able to iterate over all neighbours $s \xrightarrow{b} \mu$ of a transition $s \xrightarrow{a} \mathbb{1}_t$.

Given a deterministic transition $s \xrightarrow{a} \mathbb{1}_t$, the result of the function call *checkConfluence*($s \xrightarrow{a} \mathbb{1}_t$) tells us whether or not this transition is confluent. We first discuss this function *checkConfluence*, and then the function *checkEquivalence* on which it relies (which determines whether or not two distributions are equivalent up-to confluent steps).

These functions do not yet fully take into account the fact that confluent transitions have to be mimicked by confluent transitions. Therefore, we have an additional function *checkConfluentMimicking* that is called after termination of *checkConfluence* to see if indeed no violations of this condition occur.

The function *checkConfluence* first checks if a transition is invisible and was not already detected to be confluent before. Then, it is added to the global set of confluent transitions $\mathcal{T}$. To check whether this is valid, a loop checks if indeed all outgoing transitions from s commute with $s \xrightarrow{a} \mathbb{1}_t$. If so, we return *true* and keep the transition in $\mathcal{T}$. Otherwise, all transitions that were added to $\mathcal{T}$ during these checks are removed again and we return *false*. Note that it would not be sufficient to only remove $s \xrightarrow{a} \mathbb{1}_t$ from $\mathcal{T}$, since during the loop some transitions may have been detected to be confluent (and hence added to $\mathcal{T}$) based on the fact that $s \xrightarrow{a} \mathbb{1}_t$ was in $\mathcal{T}$. As $s \xrightarrow{a} \mathbb{1}_t$ turned out not to be confluent, we can also not be sure anymore if these other transitions are indeed actually confluent.

The loop to check whether all outgoing transitions commute with s follows directly from the definition of confluent sets, which requires for every $s \xrightarrow{b} \mu$ that either $\mu = \mathbb{1}_t$, or that there exists a transition $t \xrightarrow{c} \nu$ such that $\mu \equiv_R \nu$, where $t \xrightarrow{c} \nu$ has to be in $\mathcal{T}$ if $s \xrightarrow{b} \mu$ is. Indeed, if $\mu = \mathbb{1}_t$ we immediately continue to the next transition (this includes the case that $s \xrightarrow{b} \mu = s \xrightarrow{a} \mathbb{1}_t$). Otherwise, we range over all transitions $t \xrightarrow{c} \nu$ to see if there is one such that $\mu \equiv_R \nu$. For this, we use the function *checkEquivalence*(μ, ν), described below. Also, if $s \xrightarrow{b} \mu \in \mathcal{T}$, we have to check if also $t \xrightarrow{c} \nu \in \mathcal{T}$. We do this by checking it for confluence, which immediately returns if it is already in $\mathcal{T}$, and otherwise tries to add it.

If indeed we find a mimicking transition, we continue. If $s \xrightarrow{b} \mu$ cannot be mimicked, confluence of $s \xrightarrow{a} \mathbb{1}_t$ cannot be established. Hence, we reset $\mathcal{T}$ as discussed above, and return *false*. If this did not happen for any of the outgoing transitions of s, then $s \xrightarrow{a} \mathbb{1}_t$ is indeed confluent and we return *true*.

The function *checkEquivalence* checks whether $\mu \equiv_R \nu$. Since $\mathcal{T}$ is constructed on-the-fly, during this check some of the transitions from the support of μ might have not been detected to be confluent yet, even though they are. Therefore,

Algorithm 6: Detecting confluence on a concrete state space.

```
global $\mathit{Set}\langle \mathit{Transition}\rangle\ \mathcal{T} := \varnothing$
global $\mathit{Set}\langle \mathit{Transition}, \mathit{Transition}\rangle\ C := \varnothing$

bool $\mathit{checkConfluence}(s \xrightarrow{a} \mathbb{1}_t)$ {
  if $L(s) \neq L(t)$ then
      return $\mathit{false}$
  else if $s \xrightarrow{a} \mathbb{1}_t \in \mathcal{T}$ then
      return $\mathit{true}$

  $\mathit{Set}\langle \mathit{Transition}\rangle\ \mathcal{T}_{\text{old}} := \mathcal{T}$
  $\mathit{Set}\langle \mathit{Transition}, \mathit{Transition}\rangle\ C_{\text{old}} := C$
  $\mathcal{T} := \mathcal{T} \cup \{s \xrightarrow{a} \mathbb{1}_t\}$
  foreach $s \xrightarrow{b} \mu$ do
      if $\mu = \mathbb{1}_t$ then continue
      foreach $t \xrightarrow{c} \nu$ do
          if $\mathit{checkEquivalence}(\mu, \nu)$ and
          $(s \xrightarrow{b} \mu \notin \mathcal{T}$ or $(\exists u \,.\, \nu = \mathbb{1}_u$ and $\mathit{checkConfluence}(t \xrightarrow{c} \mathbb{1}_u)))$ then
              $C := C \cup \{(s \xrightarrow{b} \mu, t \xrightarrow{c} \nu)\}$
              continue outermost loop
          end
      $\mathcal{T} := \mathcal{T}_{\text{old}}$
      $C := C_{\text{old}}$
      return $\mathit{false}$
  return $\mathit{true}$
}

bool $\mathit{checkEquivalence}(\mu, \nu)$ {
  $Q := \{\{p\} \mid p \in \mathit{supp}(\mu) \cup \mathit{supp}(\nu)\}$
  foreach $u \xrightarrow{d} \mathbb{1}_v$ such that $u \in \mathit{supp}(\mu),\ v \in \mathit{supp}(\nu)$ do
      if $\mathit{checkConfluence}(u \xrightarrow{d} \mathbb{1}_v)$ then
          $Q := \{q \in Q \mid u \notin q \wedge v \notin q\} \cup \{ \bigcup_{\substack{q \in Q \\ u \in q \vee v \in q}} q\}$
  if $\mu(q) = \nu(q)$ for every $q \in Q$ then
      return $\mathit{true}$
  else
      return $\mathit{false}$
  end
}

bool $\mathit{checkConfluentMimicking}$ {
  foreach $(s \xrightarrow{b} \mu, t \xrightarrow{c} \nu) \in C$ do
      if $s \xrightarrow{b} \mu \in \mathcal{T}$ and $t \xrightarrow{c} \nu \notin \mathcal{T}$ then
          if $\mathit{checkConfluence}(t \xrightarrow{c} \nu)$ then
              return $\mathit{checkConfluentMimicking}$
          else
              return $\mathit{false}$
          end
  return $\mathit{true}$
```

instead of checking for connecting transitions that are already in $\mathcal{T}$, we try to add possible connecting transitions to $\mathcal{T}$ using a recursive call.

In accordance to Definition 8.6, we first determine the smallest equivalence relation that relates states from the support of μ to states from the support of ν in case there is a confluent transition connecting them. We do so by constructing a set of equivalence classes Q, i.e., a partitioning of the state space according to this equivalence relation. We start with the smallest possible equivalence relation, in which each equivalence class is a singleton. Then, for each confluent transition $u \xrightarrow{d} \mathbb{1}_v$, with $u \in \mathit{supp}(\mu)$ and $v \in \mathit{supp}(\nu)$, we merge the equivalence classes containing u and v. Finally, we can easily compute the probability of reaching each equivalence class of Q by either μ or ν. If all of these probabilities coincide, indeed $\mu \equiv_R \nu$ and we return *true*; otherwise, we return *false*.

The function *checkConfluentMimicking* is called after *checkConfluence* designated a transition to be confluent, to verify if $\mathcal{T}$ satisfies the requirement that confluent transitions are mimicked by confluent transitions. After all, when a mimicking transition for some transition $s \xrightarrow{b} \mu$ was found, it may have been the case that $s \xrightarrow{b} \mu$ was not yet in $\mathcal{T}$ while in the end it is. Hence, *checkConfluence* keeps track of the mimicking transitions in a global set C. If a transition $s \xrightarrow{a} \mathbb{1}_t$ is shown to be confluent, all pairs $(s \xrightarrow{b} \mu, t \xrightarrow{c} \nu)$ of other outgoing transitions from s and the transitions that were found to mimic them from t are added to C. If $s \xrightarrow{a} \mathbb{1}_t$ turns out not to be confluent after all, the mimicking transitions that were found in the process are removed again.

Based on C, *checkConfluentMimicking* ranges over all pairs $(s \xrightarrow{b} \mu, t \xrightarrow{c} \nu)$, checking if one violates the requirement. If no such pair is found, we return *true*. Otherwise, the current set $\mathcal{T}$ is not valid yet. However, it could be the case that $t \xrightarrow{c} \nu$ is not in $\mathcal{T}$, while it is confluent (but since $s \xrightarrow{b} \mu$ was not in $\mathcal{T}$ at the moment the pair was added to C, this was not checked earlier). Therefore, we still try to denote $t \xrightarrow{c} \nu$ as confluent. If we fail, we return *false*. Otherwise, we check again for confluent mimicking using the new set $\mathcal{T}$.

8.4.2 Correctness

The following theorem states that the algorithm is sound. We assume that C and $\mathcal{T}$ are not reset to their initial value $\varnothing$ after termination of *checkConfluence*.

Theorem 8.12. *Given a transition $p \xrightarrow{l} \mathbb{1}_q$, checkConfluence($p \xrightarrow{l} \mathbb{1}_q$) and checkConfluentMimicking together imply that $p \xrightarrow{l} \mathbb{1}_q$ is confluent.*

Note that the converse of this theorem does not always hold. To see why, consider the situation that *checkConfluentMimicking* fails because a transition $s \xrightarrow{b} \mu$ was mimicked by a transition $t \xrightarrow{c} \nu$ that is not confluent, and $s \xrightarrow{b} \mu$ was added to $\mathcal{T}$ later on. Although we then abort, there might have been another transition $t \xrightarrow{d} \rho$ that could also have been used to mimic $s \xrightarrow{b} \mu$ and that *is* confluent. We chose not to consider this due to the additional overhead of the implementation. Additionally, in none of our case studies this situation occurred.

Applying the algorithm more than once. The confluence detection algorithm often needs to be applied multiple times during SMC: every time that a nondeterministic state is visited, the algorithm is used to detect whether it is safe to restrict to only traversing one of the outgoing transitions. What we then basically do, is combine several confluent sets into one larger confluent set. As we saw in Chapter 6, taking unions of confluent sets can be problematic. Therefore, our definition there was based on a confluence classification. However, as mentioned in Remark 8.7, we do not take into account such a classification in our definitions in this chapter—to make the notion easier to apply in an SMC setting.

Hence, we need to argue why multiple executions of the algorithm indeed can be combined to reduce the same model multiple times. We do so by noting that confluence reduction yields an equivalent model, as can be observed by combining Theorem 8.11 and Theorem 8.12. By transitivity of $\text{PCTL}^*_{\setminus X}$ equivalence, it immediately follows that it is perfectly fine to reduce a model once, and then to apply the same reduction technique again on the reduced model while completely forgetting about the first reduction.

The only issue here is that this is not entirely what happens. After all, if a nondeterministic state s is reached and confluence reduction allows us to remove all but one of its outgoing transitions, this is not remembered due to the nature of SMC. Hence, if the computation at some point returns to s, it again has all its outgoing transitions. This implies that we are not continuing with the reduced model, and hence that the above reasoning does not immediately apply. This can be solved, though, in a rather intuitive manner—making sure that we indeed *are* continuing working with the actual reduced model.

Let's say we're trying to resolve the nondeterminism in state s. It has a transition $s \xrightarrow{a} t$, that we have shown to be confluent by a confluent set $\mathcal{T}$ obtained from our algorithm. Now:

1. Make a copy s' of s. Let s' have all transitions of s, and let all transitions of the original system that went to s now go to s'. Clearly, the new system is $\text{PCTL}^*_{\setminus X}$ equivalent to the original system—for instance, by the bisimulation relation $\mathit{Id} \cup \{(s, s'), (s', s)\}$. Basically, due to the SMC algorithm not remembering the reductions we will apply to s, this copying process models precisely what actually happens in practice.
2. Note that $s \xrightarrow{a} t$ is also confluent in the altered system, by observing that if a transition $s \xrightarrow{b} \mu$ was confluent originally, then so is the corresponding transition from s' in the new model. All other confluent transitions also remain confluent, as they behave identically.
3. Reduce, by omitting all transitions from s except for $s \xrightarrow{a} t$. We know now that the reduced system has the same properties as the original system. So, we can forget about the confluence information and continue with this model. Indeed, all behaviour from s is still present (but now from s'), reflecting the fact that if simulation returns to s, the omitted transitions are back again.

Remark 8.13. Note that if s is reachable from t, this process only increases the size of the state space. Although in the setting of SMC this is fine (as the size of the state space is irrelevant anyway), it would be very undesirable in the context of normal model checking. Hence, this is why this technique is only valid as a way of arguing the correctness of our SMC method, but cannot be used to more easily enable us to take unions of confluent sets in a setting such as the one in the previous two chapters. □

8.5 Evaluation

The modes tool[4] provides SMC for models specified in the Modest language [BHH12]. It allowed SMC for MDPs using the POR-based approach of [BFHH11]. We have now implemented the confluence-based approach presented in this chapter in modes as well. In this section, we apply it to three examples to evaluate its applicability and performance impact. They were selected so as to allow us to clearly identify its strengths and limitations. For each, we (1) give an overview of the model, (2) discuss, if POR fails, why it does and which, if any, modifications were needed to apply the confluence-based approach, and (3) evaluate memory use and runtime.

The performance results are summarised in Table 8.5. For the runtime assessment, we compare to simulation with uniformly-distributed probabilistic resolution of nondeterminism. Although such a hidden assumption cannot lead to trustworthy results in general (but is implemented in many tools), it is a good *baseline* to judge the *overhead* of confluence checking. We generated 10 000 runs per model instance to compute probabilities p_{smc} for case-specific properties. Using reasoning based on the Chernoff-Hoeffding bound [PRI13], this guarantees the following probabilistic error bound: $\text{Prob}(|p - p_{\text{smc}}| > 0.01) < 0.017$, where p is the actual probability of the property under consideration.

We measure memory usage in terms of the maximum number of extra states kept in memory at any time during confluence (or POR) checking, denoted by s. We also report the maximum number of "lookahead" steps (i.e., the maximal number of *checkConfluence* calls on the stack) necessary in the confluence/POR checks as k, as well as the average length t of a simulation trace and the average number c of nontrivial confluence checks, i.e., of nondeterministic choices encountered, per trace.

To get a sense for the size of the models considered, we also attempted model checking (using mcpta [HH09], which relies on PRISM [KNP11]). Note that we do not intend to perform a rigorous comparison of SMC and traditional model checking in this section and instead refer the interested reader to dedicated comparison studies such as [YKNP06]. Model checking for the BEB example was performed on a machine with 120 GB of RAM [BFHH11]; all other measurements used a dual-core Intel Core i5 M450 system with 4 GB of RAM running 64-bit Windows 7.

8.5.1 Dining cryptographers

First, we consider the classical dining cryptographers problem [Cha88]: N cryptographers use a protocol that has them toss coins and communicate the

[4]modes is part of the Modest Toolset, available at `www.modestchecker.net`.

		uniform:	partial order:			**confluence**:					model checking:	
model	params	time	time	k	s	time	k	s	c	t	states	time
dining crypto-graphers (N)	(3)	1 s	–	–	–	3 s	4	9	4.0	8.0	609	1 s
	(4)	1 s	–	–	–	11 s	6	25	6.0	10.0	3 841	2 s
	(5)	1 s	–	–	–	44 s	8	67	8.0	12.0	23 809	7 s
	(6)	1 s	–	–	–	229 s	10	177	10.0	14.0	144 705	26 s
	(7)	1 s	–	–	–	– timeout –					864 257	80 s
CSMA/CD (RF, BC_{max})	$(2,1)$	2 s	–	–	–	4 s	3	46	5.4	16.4	15 283	11 s
	$(1,1)$	2 s	–	–	–	4 s	3	46	5.4	16.4	30 256	49 s
	$(2,2)$	2 s	–	–	–	10 s	3	150	5.1	16.0	98 533	52 s
	$(1,2)$	2 s	–	–	–	10 s	3	150	5.1	16.0	194 818	208 s
BEB (K, N, H)	$(4,3,3)$	1 s	3 s	3	4	1 s	3	7	3.3	11.6	$>10^3$	>0 s
	$(8,7,4)$	2 s	7 s	4	8	4 s	4	15	5.6	16.7	$>10^7$	>7 s
	(16,15,5)	3 s	18 s	5	16	11 s	5	31	8.3	21.5	– memout –	
	(16,15,6)	3 s	40 s	6	32	34 s	6	63	11.2	26.2	– memout –	

Table 8.5: Confluence simulation runtime overhead and comparison.

outcome with some of their neighbours at a restaurant table in order to find out whether their master or one of them just paid the bill, without revealing the payer's identity in the latter case. We model this problem as the parallel composition of N instances of a `Cryptographer` process that communicate via synchronisation on shared actions, and consider as properties the probabilities of (a) protocol termination and (b) correctness of the result.

The model is a nondeterministic MDP. In particular, the order of the synchronisations between the cryptographer processes is not specified, and could conceivably be relevant. It turns out that all nondeterminism can be discarded as spurious by the confluence-based approach, though, allowing the application of SMC to this model. The computed probability p_{smc} is 1.0 for both properties, which coincides with the actual probabilities.

The POR-based approach does not work: although the nondeterministic ordering of synchronisations between non-neighbouring cryptographers is due to interleaving, the choice of which neighbour to communicate with first for a given cryptographer process is a nondeterministic choice *within* that process—such nondeterminism cannot be resolved with the POR implementation.

Concerning performance, we see that runtime increases drastically with the number of cryptographers, N. An increase is expected, since the number of steps until independent paths from nondeterministic choices join again (k) depends directly on N. It is so drastic due to the sheer amount of branching that is present in this model. At the same time, the model is extremely symmetric and can thus be handled easily with a symbolic model checker like PRISM.

8.5.2 IEEE 802.3 CSMA/CD

As a second example, we take the Modest model of the Ethernet (IEEE 802.3) CSMA/CD approach that was introduced in [HH09]. It consists of two identical stations attempting to send data at the same time, with collision detection and a randomised backoff procedure that tries to avoid collisions for subsequent retransmissions. We consider the probability that both stations eventually manage to send their data without collision. The model is a probabilistic timed automaton (PTA), but delays are fixed and deterministic, making it equivalent to an MDP (with real variables for clocks, updated on transitions that explicitly represent the delays; modes does this transformation automatically and on-the-fly). The model has two parameters: a time reduction factor RF (i.e., delays of t time units with $RF = 1$ correspond to delays of $\frac{t}{2}$ time units with $RF = 2$), and the maximum value used in the exponential backoff part of the protocol, BC_{max}.

Unfortunately, modes immediately reports nondeterminism that cannot be discarded as spurious. Inspection of the reported lines in the model quickly shows a nondeterministic choice between two probabilistic transitions—which confluence cannot handle. Fortunately, this problem can easily be eliminated through an additional synchronisation, leading to $p_{\text{smc}} = 1.0$ (which is the correct result). POR still fails, for reasons similar to the previous example: initially, both stations send at the same time, the order being determined nondeterministically. In the process representing the shared medium, this must be an *internal* nondeterministic choice. In contrast to the problem for confluence, this cannot be fixed easily.

In terms of runtime, the confluence checks incur a moderate overhead for this example. Compared to the dining cryptographers, the slowdown is much less even where more states need to be explored in each check (s); performance appears to more directly depend on k, which stays low in this case.

8.5.3 Binary exponential backoff

The previous two examples clearly indicate that the added power of confluence reduction pays off, allowing SMC for models where it is not possible with POR. Still, we also need a comparison of the two approaches. For this purpose, we revisit the MDP model of the binary exponential backoff (BEB) procedure that was used to evaluate the POR-based approach in [BFHH11]. The probability we compute is that of some host eventually getting access to the shared medium, for different values of the model parameters K (maximum backoff counter value), N (number of tries per station before giving up) and H (number of stations/hosts involved).

Again, for the confluence check to succeed, we first need to minimally modify the model by making a probabilistic transition synchronise. This appears to be a recurring issue, yet the relevant model code could quite clearly be identified as a modelling artifact without semantic impact in both examples where it appears. We then obtain $p_{\text{smc}} = 0.91$ for model instance $(4, 3, 3)$, otherwise $p_{\text{smc}} = 1.0$.

The runtime overhead necessary to get trustworthy results by enabling either confluence or POR is again moderate. This is despite longer paths being explored in the confluence checks compared to the CSMA/CD example (k). The confluence-based approach is somewhat faster than POR in this implementation. As noted in [BFHH11], large instances of this model cannot be solved with classical model checking due to the state space explosion problem.

8.6 Contributions

We defined a more slightly more liberal variant of probabilistic confluence than the one defined in Chapter 7, tailored for the core simulation step of statistical model checking. It has more reduction potential at no extra computational cost, but still preserves $\text{PCTL}^*_{\setminus X}$. We provided an algorithm for on-the-fly detection of confluence during simulation and implemented this algorithm in the modes SMC tool[5].

Compared to a previous approach based on partial order reduction [BFHH11], the use of confluence allows new kinds of nondeterministic choices to be handled, in particular lifting the limitation to spurious interleavings—now, also spurious nondeterminism within a process may be resolved. Heuristics for partial order reduction generally consider all actions within the same process to be dependent [BK08, HP95, BFHH11].

In fact, for two of the three examples we presented, SMC is only possible using the new confluence-based technique, showing the additional power to be relevant. In terms of performance, it is somewhat faster than the POR-based approach, but the impact relative to (unsound) simulation using an arbitrary scheduler largely depends on the amount of lookahead that needs to be performed, for both approaches. Again, on two of our examples, the impact was moderate and should in general be acceptable to obtain trustworthy results. Most importantly, the memory overhead is negligible, and one of the central advantages of SMC over traditional model checking is thus retained.

As confluence preserves branching time properties, it cannot handle the interleaving of probabilistic choices. Although—as we showed—these can often be avoided, for some models POR might work while confluence does not. Hence, neither of the techniques subsumes the other, and it is best to combine them: if one cannot be used to resolve a nondeterministic choice, the SMC algorithm can still try to apply the other. Implementing this combination is trivial and yields a technique that handles the union of what confluence and POR can deal with.

[5]Note that, as also mentioned at the beginning of this chapter, the implementation and case studies were not done by the author, but by Arnd Hartmanns.

Part IV

Practical Validation

CHAPTER 9

Implementation and Case Studies

"It doesn't matter how beautiful your theory is,
it doesn't matter how smart you are.
If it doesn't agree with experiment, it's wrong."

Richard P. Feynman

The previous chapters described the process-algebraic language MAPA for modelling MAs, as well as several reduction techniques for optimising MAPA specifications and generating their corresponding MAs.

To validate the applicability of MAPA and the significance of its reduction techniques, we developed a prototype tool called SCOOP. It takes as input either a MAPA specification or a generalised stochastic Petri net (GSPN), linearises to an MLPE and applies all our reduction techniques while generating an MA. These MAs can be subjected to analysis by a tool called IMCA, developed by Dennis Guck at RWTH Aachen University and Hassan Hatefi at Saarland University [GHKN12, HH12, GHH$^+$13a]. We briefly present the analysis techniques that are available for MAs, as implemented in IMCA. These techniques have not been developed by the author, but provide purpose to the MAPA language and the MAs that can be generated. We do not go into any technical details, but only discuss IMCA's features from a user perspective.

Together with IMCA and the tool GEMMA for parsing GSPN specifications, SCOOP is part of the MaMa (Modelling and Analysis of Markov Automata) tool chain. We provide four case studies that illustrate the capabilities of SCOOP as part of this tool chain: an ingenious construction of a handshake register, a well-known leader election protocol, a polling system and a processor grid architecture. We illustrate how to model such systems in MAPA and what kind of properties can be computed. Our results confirm that the basic reduction techniques (Section 4.5), dead variable reduction (Chapter 5) and confluence reduction (Chapter 6) all are valuable tools when generating and analysing MAPA models.

Organisation of the chapter. In Section 9.1 we introduce the tool SCOOP, describing its input and output formats and discussing the MaMa tool chain in which it is embedded. Section 9.2 presents the analysis techniques implemented by IMCA. Then, Section 9.3 presents our case studies and discusses their individual results. Finally, Section 9.4 concludes by summarising the contributions of this chapter and providing an overarching discussion on the merits of our reduction techniques as illustrated by the case studies.

Origins of the chapter. An earlier version of SCOOP was introduced by a tool paper in the proceedings of the *8th International Conference on Quantitative Evaluation of Systems* (QEST) [Tim11], whereas the last version was described by a regular paper in the proceedings of the *10th International Conference on Quantitative Evaluation of Systems* (QEST) [GHH+13a] and a corresponding technical report [GHH+13b]). Additionally, part of the case studies already appeared throughout the papers that introduced the techniques described in the previous chapters.

9.1 Implementation

We developed the prototype tool SCOOP[1] in Haskell, based on a simple data language that allows the modelling of several kinds of protocols and systems. Excluding the generated parser, it has just over 3,100 lines of code.

A web-based interface (developed by Axel Belinfante using his Puptol tool[2]) makes the tool convenient to use, but it can also be used stand-alone. The tool is capable of linearising MAPA specifications, as well as applying parallel composition, hiding, encapsulation and renaming (as explained in Chapter 4). Since Haskell is a functional language, the algorithms in our implementation are almost identical to their mathematical representations presented in this thesis. The tool also implements all reduction techniques introduced in Section 4.5, Chapter 5 and Chapter 6. Where applicable, it encodes MLPEs as LPPEs as explained in Section 4.2.5.

9.1.1 Input

SCOOP takes as input any specification in the MAPA language, and allows the use of a couple of built-in data types and several built-in functions for these data types. By default it supports booleans, integers, rational numbers, lists and stacks. Additionally, users can define enumerative types for modelling convenience.

For booleans, we support the basic operations: negation, conjunction and disjunction. Also, we include conditional expressions. For integers and rationals, we support the standard numerical operations: addition, subtraction, multiplication, division and exponentiation. Additionally, we implemented functions for obtaining the minimum or maximum of two numbers and for modulo computation. Finally, integers and rationals can be compared using the standard mathematical operators $<$, $\leq$, $=$, $\geq$ and $>$. For lists and stacks we allow elements to be appended, read, updated, deleted, pushed or popped, and we can obtain their size.

The syntax expected by SCOOP largely corresponds to the notations we introduced in Chapter 4. The most important concepts are presented in Table 9.1, linking the MAPA notations introduced in Chapter 4 to the corresponding textual

[1]The implementation, including a web-based interface, can be found at `http://fmt.cs.utwente.nl/~timmer/scoop`. A list of all features and the way to use them is shown at `http://fmt.cs.utwente.nl/~timmer/scoop/manual.html`.

[2]See `http://fmt.ewi.utwente.nl/puptol/`.

Concept	MAPA	SCOOP
Nondeterministic sum	$\sum_{d:D} p$	`sum(d:D, p)`
Probabilistic sum	$a(m)\sum_{d:D} f : p$	`a(m) . psum(d:D, f : p)`
Markovian rate	$(5) \cdot p$	`<5> .  p`
Process instantiation	$X(\dots)$	`X[...]`
Hiding	$\tau_{\{a,b\}}(q)$	`hide(a,b : q)`
Encapsulation	$\partial_{\{a,b\}}(q)$	`encap(a,b : q)`
Renaming	$\rho_{(a \mapsto b)}(q)$	`rename((a,b) : q)`
Communication	$\gamma(a, b) = c$	`comm (a,b,c)`
Initial process	$X(\dots) \,\|\|\, Y(\dots)$	`init X[...] || Y[...]`

Table 9.1: Syntax of SCOOP.

representations in SCOOP[3]. The specifications presented in this chapter should be understandable using this table.

To complement SCOOP, Bamberg developed the GEMMA tool [Bam12] to transform GSPN specifications to the MAPA language. We apply this tool as a preprocessor for SCOOP. We will not discuss the syntax and semantics of GSPNs; rather, we refer the interested reader to [Bam12] for a detailed introduction.

9.1.2 Output

After generating an MLPE—applying any subset of the reduction techniques—SCOOP can also generate its state space (an MA) and display it in several ways. Most importantly, SCOOP can export an MA to the input language for the IMCA model checker [GHH+13a] for further analysis. Also, it can provide a human-readable enumeration of all states and transitions, just count how many there are, or show the deadlock states.

For the subclass of PAs, it can also export to the AUT format for analysis with the CADP toolset [GLMS13], or to a transition matrix for analysis using PRISM [KNP11]. For PA specifications using booleans and integers only, the tool can also translate a (possibly reduced) LPPE directly to a PRISM specification, enabling the user to use this probabilistic model checker to symbolically compute quantitative properties of the model. In case a PA is actually a DTMC, the resulting AUT file can easily be translated to a pair of TRA and LAB files for analysis with MRMC [KZH+11][4]. For CTMCs we did not yet implement a translation for analysis with MRMC, but this would be an easy extension.

9.1.3 The MaMa tool chain

We connected the three tools GEMMA, SCOOP and IMCA into a single tool chain, by having GEMMA export MLPEs and having SCOOP export the

[3]We refer to `http://fmt.cs.utwente.nl/~timmer/scoop/syntax.html` for a detailed overview of the syntax of SCOOP's input language.

[4]See `http://fmt.cs.utwente.nl/~timmer/scoop/casestudies/roulette` for an application of this trajectory.

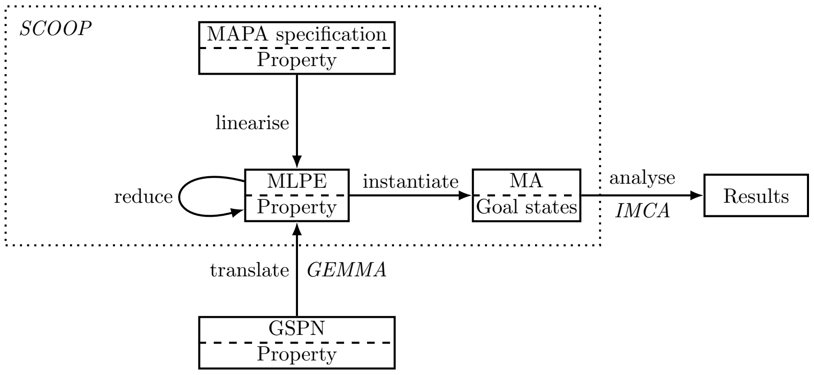


Figure 9.2: Analysing Markov Automata using MaMa.

(reduced) state space of an MLPE in the IMCA input language. Together, these tools are called the MaMa tool chain, visualised in Figure 9.2.

For analysis purposes, SCOOP is able to translate properties, based on the actions and parameters of a MAPA specification, to a set of goal states in the underlying MA. That way, in one easy process systems and their properties can be modelled in MAPA (or as a GSPN and then translated to MAPA by GEMMA), translated to an optimised MLPE by SCOOP, and exported to the IMCA tool for analysis.

9.1.4 Coupling SCOOP with LTSmin

At the time of writing, Stefan Blom is working on a coupling between SCOOP and the LTSmin tool [BvdPW10]. This tool has extensive functionality for very efficient state space generation, optimising memory usage, applying caching and employing multi-core and distributed model checking techniques. Preliminary results indicate that this indeed significantly speeds up the generation of MAs based on MLPEs. According to Blom, this is mostly due to the caching functionality that takes into account which summands update which parameters. This functionality was already present in LTSmin. All MLPE-optimising reduction techniques implemented by SCOOP, except for confluence reduction, can directly be used, as the coupling is implemented in such a way that LTSmin builds the state space efficiently while relying on SCOOP to deliver the successors of a given state. Confluence reduction, on the other hand, is also active during state space generation due to the search for representatives. Therefore, this is not immediately applicable. It is currently in the process of being implemented in LTSmin's state space generation procedure.

Basically, LTSmin just takes over SCOOP's state space generation routine, and hence can hopefully soon be used without even being visible to the user. Together, SCOOP and LTSmin behave precisely as SCOOP did so far, taking in MAPA specifications and producing MAs for analysis in IMCA—only significantly faster. We envision significant performance gains from the coupling of SCOOP and LTSmin, combining SCOOP's syntactic reductions with LTSmin's meticulously optimised state space generation.

9.1.5 Confluence implementation

When applying confluence reduction, the target states of each transition are substituted by their representatives. These representatives are found by following confluent transitions until arriving in a BSCC, as discussed earlier in Chapter 6. During this search, several states are visited but not stored in the reduced state space. The actual number of states that are visited during state space generation is therefore larger than the number displayed in our tables—the number presented there is the actual number of states in the reduced state space.

To keep memory usage in the order of the reduced state space, the representation map is deliberately not stored by default. Therefore, if non-representative states are visited for the second time they are not recognised and their representative is recomputed. In principle, the number of states visited during state space generation may thus even be larger than the original number of states.

For models with a lot of cyclicity, not remembering anything may not be the most efficient approach. Therefore, SCOOP also allows to store part of the representation map, trading speed for memory usage. Whenever a state s is reached and its representative t is computed, the pair (s, t) is then stored in a hash table. That way, whenever s is visited again, its representative t does not need to be recomputed. For one of our case studies (the leader election protocol) this variant of confluence reduction is applied. This is also indicated in the table presenting its results. We investigated the possibility of additionally storing pairs (s', t) for all states s' visited on the path from s to t while looking for a representative for s. However, that did not turn out to be beneficial.

Depending on the model structure and the specific confluence implementation that is used, confluence reduction sometimes only marginally improves or even slows down state space generation time. Nonetheless, the reduction in state space size is still useful if the model is subject to further analysis, as it often is.

9.1.6 Compositional analysis

An interesting approach is to generate state spaces by building them compositionally [BCS10, LM09]. That is, first, two components are generated and composed and the result is minimised. Then, the minimised composition is again composed with a third component and the result is minimised. This continues, until the complete system has been generated in a compositional manner.

Although in principle this approach seems to be applicable to MAPA specifications as well, we did not implement it yet. One important reason for not choosing such an approach is that some reduction techniques benefit from the fact that an MLPE contains information about the full system. For instance, dead variable reduction may reset variables based on a global specification; those variables may not have been reset in the individual components. Therefore, compositional state space generation may even yield larger (intermediate) state spaces than we obtain in our approach. Still, compositional state space generation (including for instance compositional confluence detection) is an interesting direction for future work.

9.2 Analysing MAs with the MaMa tool chain

With the MaMa tool chain, four types of properties of an MA can be computed. All are defined based on a subset of the states of the MA: the set of *goal states*. As mentioned before, SCOOP allows for a set of goal states to be generated based on the actions and properties of a MAPA specification. More specifically, we have two options for generating goal states: either (1) we provide SCOOP with a list of actions, or (2) we provide SCOOP with a condition over the MLPE's variables. In the first case, the set of goal states will consist of all states that *enable* at least one of the actions in the list. Hence, goal states in general have not executed the specified action(s) yet, but they are always able to execute at least one. In the second case, the set of goal states will consist of all states that *satisfy* the condition that was specified.

9.2.1 Analysis techniques

SCOOP's job is to provide an MA together with a set of goal states. Based on this information, IMCA is able to compute the following properties:

Unbounded reachability This type of analysis computes the probability of *eventually* reaching at least one of the goal states. It ignores all timing information, as if the MA were a PA. Unbounded reachability analysis may be used to compute properties that are not concerned with timing, but for instance with valid termination of a protocol.

Time-bounded reachability This type of analysis does take timing into account. Parameterised in a lower bound l and an upper bound u, it computes the probability that at least one goal state is visited within the interval $[l, u]$. It is not concerned with the question whether one or more goal states are also already visited before time l. Time-bounded reachability analysis may be used for properties to investigate for instance how likely a system completes a certain task within a given amount of time.

Expected time This type of analysis computes the expected time until reaching a goal state for the first time. Expected time analysis may be used for performance analysis, computing for instance how long a system is expected to work on a set of tasks.

Long-run average This type of analysis computes the fraction of time, in the long run, that an MA resides in a goal state. Long-run average analysis may be used to compute throughput, for instance, or to see how often a system is idling.

We note that time-bounded reachability grows to unbounded reachability if $l = 0$ and u goes to ∞.

9.2.2 Zeno behaviour

In principle, MAs are not allowed to have any *Zeno behaviour*. That is, no loop of internal transitions may be present. After all, since such transitions take no time and prevent Markovian delays from happening, they would prevent time from progressing—this is often not allowed.

SCOOP and IMCA do not check their input for this restriction. Hence, in contrast to many theoretical papers they do allow MAs to contain Zeno behaviour. In many cases, such MAs still allow valid reasoning. For instance, for a system first delaying according to a rate of 1, then executing an action a and then looping indefinitely, we can say that the expected time until reaching the action a is 1. On the other hand, the long-run average of being in the infinite loop may be counterintuitive. Although from some point on we stay there indefinitely, time does not progress anymore. Hence, the long-run average computed by IMCA is 0.

Both tools also allow MAs to have deadlock states. IMCA prevents Zeno behaviour by adding Markovian self-loops to such states.

9.3 Case studies

The modelling power of MAPA in combination with the reduction techniques implemented by SCOOP allow a wide variety of systems to be modelled and analysed. We worked on several, such as a handshake register, a leader election protocol, a polling system, a processor grid architecture, a perfect strategy for roulette[5], the gambler's ruin problem [Blo12], a stripped-down version of Yahtzee [Blo12], the game of the goose [Blo12], and a mutual exclusion algorithm [TKvdPS12a].

In this section we describe the first four studies mentioned above. They are modelled in the MAPA language and optimised by the latest version of SCOOP using the reduction techniques introduced in this thesis. Except for the first case study, all are also analysed by means of the IMCA tool (version 1.5 beta). These case studies should be interpreted as a proof-of-concept for our modelling language and reduction techniques. They show that all our techniques work really well in some cases, and maybe less in others. To fully investigate how well our techniques work, how scalable they are and precisely what modelling artifacts influence their impact, many more experiments are needed—we leave this for future work. Still, the current treatment is sufficient for concluding that all our techniques are valuable contributions to the field of quantitative model checking.

Measurements. All measurements were taken on a 2.4 GHz 4 GB Intel Core 2 Duo MacBook running Mac OS X 10.6.8. For each case study we first present the number of states and transitions obtained using SCOOP when not applying any of our reduction techniques. We also present the time to generate these state spaces, both with and without our basic reduction techniques. The timing measurements include the whole process of parsing, linearisation, optimisation and generation: we just measure the real time SCOOP takes to display the state space size after being invoked[6].

[5]`http://wwwhome.cs.utwente.nl/~timmer/scoop/casestudies/roulette/roulette.html`.

[6]Since our Haskell-implementation is rather inefficient in I/O at the moment, we excluded the time to write the state space to disk from our measurements.

We also present the effects of our reduction techniques on the state space sizes and the generation and analysis times. We perform three categories of measurements: one applying only dead variable reduction in combination with the basic reduction techniques, one applying only confluence reduction in combination with the basic reduction techniques, and one applying all at the same time. Increases and decreases are presented as reduction percentages with respect to the setting of only applying the basic reduction techniques.

Since all algorithms implemented by SCOOP and IMCA are deterministic, we ran all experiments only once. All other user processes were killed, for accuracy of our results. When rerunning some of the experiments to check if this approach is indeed valid, we noticed at most deviations of a few percent in the timing results.

9.3.1 Handshake register

Our first case study is a model of a *handshake register*, modelled and verified by Hesselink [Hes98]. A handshake register is a data structure that is used for communication between a single reader and a single writer. It guarantees *recentness* and *sequentiality*; any value that is read was at some point during the read action the last value written, and the values of sequential reads occur in the same order as they were written. Also, it is *waitfree*; both the reader and the writer can complete their actions within a bounded number of steps, independent of the other process.

Hesselink provided a method to implement a handshake register of an arbitrary data type based on four so-called safe registers and four atomic boolean registers. Each safe register can be used to store one value, and only guarantees to provide the actual value to the reader if no write action is in progress at the same time. Hence, the safe registers on their own do not satisfy the recentness requirement: if a safe register is read during a write operation, this may result in any value. Still, Hesselink managed to combine four copies of such a safe register with four atomic boolean registers in such a way that the result does have all the required properties for a handshake register. The algorithm is far from trivial, and Hesselink's correctness proof involves several invariants.

As an alternative to the manual proof, we present a model checking approach to show the correctness of this construction. The handshake register's specification and its construction based on the four safe register were already modelled in μCRL by Arjan Mooij, Aad Mathijssen and Jan Friso Groote. We translated these models to MAPA, generated their state spaces and compared these modulo τ^*a equivalence[7] [FM91] using μCRL's `ltscmp` tool. Indeed, as expected, the specification and implementation yield identical minimal quotients. Therefore, the construction satisfies its requirements. Below, we go into more details about the models and their analysis.

[7]The concept of τ^*a equivalence is similar to weak bisimulation. For two states p, q to be τ^*a-equivalent, each path $p \xrightarrow{\tau\ldots\tau a} p'$ has to be mimicked by a path $q \xrightarrow{\tau\ldots\tau a} q'$ such that p' and q' are again τ^*a-equivalent. Here, the sequences $\tau\ldots\tau$ may be of distinct length and are allowed to be empty; hence, the term τ^*a equivalence.

```
type D   = {1..DataSize}
type Pos = {1..3}

Register(readstatus:Pos, writestatus:Pos,v:D,vw:D,vr:D) =
    readstatus  = 1 => begin_read     . Register[readstatus  := 2          ]
 ++ readstatus  = 2 => tau            . Register[readstatus  := 3, vr := v ]
 ++ readstatus  = 3 => end_read(vr)   . Register[readstatus  := 1          ]
 ++ writestatus = 1 =>
              sum(w:D, begin_write(w) . Register[writestatus := 2, vw := w ])
 ++ writestatus = 2 => tau            . Register[writestatus := 3, v  := vw]
 ++ writestatus = 3 => end_write      . Register[writestatus := 1          ]

init Register[1,1,1,1,1]
```

Figure 9.3: A MAPA model of a handshake register's specification.

Modelling. The specification is rather straightforward, as depicted in Figure 9.3. The data type of the item that is stored is assumed to be the set of integers from 1 to `DataSize`, the upper bound being a constant that can be varied. The register allows read and write actions to be interleaved, using two status variables to keep track of the current position. Reading consists of a `begin_read` action, an internal action that copies the contents of variable `v` (which is assumed to be available to both reader and writer) to variable `vr`, and an `end_read` action parameterised in the value that was read. Writing consists of a `begin_write` action parameterised in a value of the given data type (storing this value in the variable `vw`), an internal action that copies the contents of variable `vw` to variable `v`, and an `end_write` action. Note that, indeed, this specification always reads a value that was previously written—under the assumption that the initial value of `v` is considered a written value as well. Additionally, values can never be read out of order, and readers and writers do not rely on the other to complete their operations.

The model of Hesselink's construction is more complicated: it consists of several processes, as depicted in Figure 9.4. The process `Y` models the safe registers. It is as the specification of the handshake register in Figure 9.3, except that it only correctly copies `v` to `vr` if no writing action is in progress. Otherwise, an arbitrary value is written to `vr`. The processes `A`, `B` and `C` model the atomic boolean registers. Finally, the `Reader` and `Writer` process model the ingenious part of Hesselink's construction: they one-to-one implement the algorithm to employ the atomic boolean registers and four copies of the safe register to mimic the behaviour of a handshake register [Hes98]. The system consists of the parallel composition of all these components, including (rather trivial) information on communication, hiding and encapsulation. The action names have been chosen to illustrate information flow: `w2s` means 'writer to safe register', `s2r` means 'safe register to reader', and so on.

Results. Both the model of the specification and the model of the implementation were fed to SCOOP to generate their state spaces and compare the results, for several instances of the variable `DataSize`. Indeed, the resulting state spaces

```
type D   = {1..DataSize}
type Pos = {1..3}

Y(i:Bool, j:Bool, readstatus:Pos, writestatus:Pos, v:D, vw:D, vr:D) =
    readstatus  = 1 => begin_s2r_s(i, j)     . Y[readstatus  := 2          ]
 ++ readstatus  = 2 & writestatus = 1
                    => tau                   . Y[readstatus  := 3, vr := v ]
 ++ readstatus  = 2 & not(writestatus = 1)
                    => sum(w:D, tau          . Y[readstatus  := 3, vr := w ])
 ++ readstatus  = 3 => end_s2r_s(i, j, vr)   . Y[readstatus  := 1          ]
 ++ writestatus = 1 =>
              sum(w:D, begin_w2s_s(i, j, w)  . Y[writestatus := 2, vw := w ])
 ++ writestatus = 2 => tau                   . Y[writestatus := 3, v  := vw]
 ++ writestatus = 3 => end_w2s_s(i, j)       . Y[writestatus := 1          ]

A(a:Bool)         =  readA_(a) . A[a]
                  ++ sum(la:Bool, writeA_(la) . A[la])

B(b:Bool)         =  readB_(b) . B[b]
                  ++ sum(lb:Bool, writeB_(lb) . B[lb])

C(i:Bool, c:Bool) =  readC_(i,c) . C[i,c]
                  ++ sum(lc:Bool, writeC_(i,lc) . C[i,lc])

Reader = begin_read .                               -- procedure read:
         sum(b:Bool, readA(b) . writeB(b) .         --   b := A; B := b;
         sum(c:Bool, readC(b,c) .                   --   c := C[b];
         begin_s2r_r(b,c) .                         --
         sum(y:D, end_s2r_r(b,c,y) .                --   y := Y[b,c];
         end_read(y) . Reader[])))                  -- return y;

Writer = sum(x:D, begin_write(x) .                  -- procedure write(x):
         sum(a:Bool, readB(not(a)) .                --   a := not(B);
         sum(d:Bool, readC(a,d) .                   --   d := C[a];
         begin_w2s_w(a, not(d), x) .                --   Y[a, not(d)] := x;
         end_w2s_w(a,not(d)) .                      --
         writeC(a,not(d)) .                         --   C[a] := not(d);
         writeA(a) .                                --   A    := a;
         end_write . Writer[])))                    -- return;

init Reader || Writer || Y[T,T,1,1,1,1,1] || Y[T,F,1,1,1,1,1] ||
     Y[F,T,1,1,1,1,1] || Y[F,F,1,1,1,1,1] || A[T] || B[T] || C[T,T] || C[F,T]

comm (begin_w2s_w,begin_w2s_s,begin_w2s), (end_w2s_w,end_w2s_s,end_w2s),
     (begin_s2r_s,begin_s2r_r,begin_s2r), (end_s2r_s,end_s2r_r,end_s2r),
     (readA,readA_,readA__), (writeA,writeA_,writeA__),
     (readB,readB_,readB__), (writeB,writeB_,writeB__),
     (readC,readC_,readC__), (writeC,writeC_,writeC__)

hide begin_w2s, end_w2s, begin_s2r, end_s2r,
     readA__,readB__,readC__,writeA__,writeB__,writeC__

encap begin_w2s_w, end_w2s_w, begin_s2r_s, end_s2r_s,
      begin_w2s_s, end_w2s_s, begin_s2r_r, end_s2r_r,
      readA , readB , readC , writeA , writeB ,writeC ,
      readA_, readB_, readC_, writeA_, writeB_,writeC_
```

Figure 9.4: A MAPA model of a handshake register's implementation.

	Original state space			Dead variable reduction		
`|D|`	States	Trans.	Time (s)	States	Trans.	Time (s)
1	2,064	4,128	0.5 (0.9)	2,064 (− 0%)	4,128 (− 0%)	1.6 (+220%)
2	540,736	1,115,712	132.9 (264.0)	45,504 (− 92%)	94,080 (− 92%)	12.9 (− 90%)
3	13,834,800	29,028,240	`timeout`	290,736 (− 98%)	613,008 (− 98%)	121.8
4	142,081,536	301,349,376	`timeout`	1,107,456 (− 99%)	2,365,440 (− 99%)	3,101.3

	Original state space			Confluence reduction		
`|D|`	States	Trans.	Time (s)	States	Trans.	Time (s)
1	2,064	4,128	0.5 (0.9)	1,208 (− 41%)	2,416 (− 41%)	1.0 (+100%)
2	540,736	1,115,712	132.9 (264.0)	314,048 (− 42%)	651,584 (− 42%)	145.7 (+ 10%)

	Original state space			All reductions together		
`|D|`	States	Trans.	Time (s)	States	Trans.	Time (s)
1	2,064	4,128	0.5 (0.9)	1,208 (− 41%)	2,416 (− 41%)	2.1 (+360%)
2	540,736	1,115,712	132.9 (264.0)	25,280 (− 95%)	52,608 (− 95%)	12.8 (− 90%)
3	13,834,800	29,028,240	`timeout`	155,304 (− 99%)	331,776 (− 99%)	78.5
4	142,081,536	301,349,376	`timeout`	574,976 (−100%)	1,251,328 (−100%)	322.7

Table 9.5: State space generation for Hesselink's implementation of a handshake register. Times for the original state spaces are provided with (and without) the basic MLPE reduction techniques. Negative percentages indicate reductions. State spaces that could not be generated explicitly anymore by SCOOP were generated symbolically with LTSmin [BvdPW10] to still be able to compute the reduction in state space size. The timeout was set at 10,000 seconds.

were weak trace equivalent (as implied by τ^*a equivalence), and hence the implementation also features recentness, sequentiality and waitfreeness.

Although the specification still only has 50,625 states if $|$`DataSize`$| = 25$, the implementation blows up rather quickly. Hence, we demonstrate the significance of our reduction techniques by applying them to this model and showing their impact on the number of states and transitions (Table 9.5).

Discussion. It is immediately clear from the table that the basic reduction techniques are of great help—they halve the state space generation time, even though they do not reduce the number of states or transitions. Considering for example the variation with `DataSize = 3`, we observed that constant elimination reduces the number of MLPE parameters from 50 to 32, that summation elimination reduces the number of nondeterministic summations from 23 to 6 and that expression simplification in combination with the other two basic techniques reduces the MLPE from 3,089 to 2,087 symbols.

Furthermore, dead variable reduction provides extremely large reductions: for `DataSize = 4`, less than 1 percent of the state space remains. The reason for the significant success of this technique can easily be explained from the observation that there are four safe registers (process `Y`) that each remember their values for `vr`, `vw` and `v` much longer than necessary. Resetting these when possible, the combinatoric explosion is partly avoided. Note that the reduction works relatively better for larger values of `DataSize`. This was to be expected as well, since a larger data type for values that are irrelevant results in a larger number of equivalent states.

Confluence reduction is of less help time-wise, although it does still reduce the state space a bit. This is due to the writer being the only process who writes to atomic boolean register `C`. Therefore, when it is reading a value from this register, it may interleave in any order with the system's other possible transitions without influencing the result. Confluence reduction detects this and gives priority to the τ-transition obtained from hiding `readC(a,d)`. The same holds for the τ-transitions corresponding to the actions `begin_s2r_s` and `end_s2r_s(i, j, vr)`. The former does not use or change any variables (except for using the constants `i` and `j`—but they are removed by constant elimination first). The latter does use `vr`, but this variable can only be updated in states that can never be enabled at the same time (since it requires a different value for `readstatus`). No write actions are confluent, as they change the `writestate` variable, which is used to decide whether or not to provide the correct result during a (possibly simultaneous) read action. Due to the excessive growth of the model, confluence reduction alone is not sufficient to analyze the model for $|\mathtt{D}| \geq 3$. However, the impact of confluence on larger models can be seen from its effect in combination with dead variable reduction. Then, for $|\mathtt{D}| \geq 3$, it does significantly reduce the state space generation time compared to the situation of only applying dead variable reduction.

The reason for dead variable reduction not always positively affecting the state space generation time for the smallest model is the additional overhead of deciding which variables can be reset (taking about 1.0 second for each model). For confluence reduction, there is the overhead to check which summands are confluent (taking about 0.5 seconds) and the additional overhead of recomputing representatives, as discussed above. For the larger models, though, these concerns completely vanish compared to the significant reductions that are obtained.

9.3.2 Leader election protocol

For our second case study, we investigate a leader election protocol: Algorithm $\mathcal{B}$ from [FP05]. It is an adaptation of the Itai-Rodeh protocol [IR81, IR90] for deciding on a leader in an asynchronous ring with an arbitrary number of nodes. Basically, the algorithm breaks the symmetry by rolling dice, choosing the node that rolled highest as leader. If multiple nodes roll the same high number, the process is repeated.

More precisely, each node is either *active* (still in the running to become the leader) or *passive* (only passing on messages). Initially, all nodes are active and they randomly choose a natural number within a certain interval $\{1, \dots, k\}$, i.e., they roll a k-sided die. We assume that this takes some time, governed by an exponential distribution with rate 1. After having chosen a value c, each node sends a message $(c, 1)$ to the channel on its right-hand side, with 1 representing the number of *hops* the message has taken. The hop count is included to be able to notice that a message makes it all the way around the ring, back to its original sender.

After having sent its message, each node n_i keeps reading messages (o, h) from the channel on its left-hand side. Upon receiving a message, four conditions apply from the point of view of node n_i:

- If $o < c$, then apparently a node on the left of n_i rolled a lower number than n_i did, so the message is discarded.

- If $o > c$, then apparently n_i did not roll the highest number. Since o may still be the highest number, n_i forwards the message while updating the hop count, i.e., it sends the message $(o, h + 1)$ to the channel on its right-hand side. Additionally, node n_i becomes passive.
- If $o = c$ and h is smaller than the number of nodes, then apparently some node on the left of n_i rolled the same number. To break the symmetry, n_i discards the message, chooses a new value c', sends a new message $(c', 1)$ to the channel on its right-hand side and continues as above.
- If $o = c$ and h equals the number of nodes, then apparently the message from n_i made it all the way across the ring. Considering the three conditions described above, this implies that all other nodes rolled a lower number than n_i did (and hence became passive), and therefore n_i can declare itself leader.

After a node became passive, it will only still forward messages while incrementing the hop counter.

Since each node always either discards a message or forwards it while becoming or staying passive, each message always leaves behind a trail of passive nodes. Hence, since a leader is only chosen if a message makes it all the way across the ring, it should be obvious that indeed at most one leader will be declared. It may be less obvious that a leader is eventually chosen, or how long this would take. Hence, a model checking approach comes in handy.

Modelling. The specification is depicted in Figure 9.6. We separately modelled the processes and the channels. Each `Channel` has an identification number `id` (for communication purposes) and can hold a value `val` of the type `D = {1..DataSize}` and a hop number. Additionally, due to its asynchronous nature we keep track of whether it is `set`. If not, a message can be stored. Otherwise, the message can be read and `set` is reset to `F` (false).

Each `Process` also has an identification number, as well as a `status`. The status indicates whether the process should (`1`) choose a new value, (`2`) read a message, or (`3`) passively forward all messages. After having chosen a new value from type `D` (each with probability $\frac{1}{\texttt{DataSize}}$), it is sent to the channel with the same identification number as the process itself and in the process' variable `chosen`. Messages are read from the channel that has an identification number that is one lower than the process' number, and are treated precisely as prescribed by the protocol. Passive processes just forward messages while incrementing the hop count, as required.

The system is composed of three channels and three processes. The `put` action from `Process` is made to communicate with the `putInChannel` action from `Channel`, and similarly they communicate to receive messages. We hide all irrelevant actions (`send`, `receive`) and encapsulate the actions that were needed only for communication. Additionally, for efficiency we keep the `leader` action visible only in one of the three processes. That way, confluence will be able to reduce more, while we are still able to compute the properties we are interested in.

```
constant NODES = 3

type D      = {1..DataSize}
type Id     = {0..NODES-1}
type Hop    = {1..NODES}
type Status = {1..3}

Channel(id:Id, set:Bool, val:D, hop:Hop) =
    not(set) => sum(v:D, sum(h:Hop, putInChannel(id, v, h) .
                Channel[set := T, val := v, hop := h]))
 ++ set      => getFromChannel(id, val, hop) . Channel[set := F]

Process(id:Id, status:Status, chosen:D) =
    status = 1 => <1> . roll(id) . psum(c:D, 1/DataSize : put(id, c, 1) .
                  Process[status := 2, chosen := c])

 ++ status = 2 => sum(o:D, sum(h:Hop, get(mod(id-1, NODES), o, h) .
                   (   h = NODES  => leader(id)      . Deadlock[]
                    ++ chosen < o => put(id, o, h+1) . Process[status := 3]
                    ++ o < chosen =>                   Process[]
                    ++ h < NODES & o = chosen =>       Process[status := 1])))

 ++ status = 3 => sum(o:D, sum(h:Hop, get(mod(id-1, NODES), o, h) .
                  put(id, o, h+1) . Process[]))

Deadlock = finished.Deadlock[]

init Channel[0, F, 1, 1] || Channel[1, F, 1, 1] || Channel[2, F, 1, 1] ||
                   Process[0,1,1]  ||
     hide(leader : Process[1,1,1]) ||
     hide(leader : Process[2,1,1])

comm (put, putInChannel, send), (get, getFromChannel, receive)

hide  send, receive

encap putInChannel, getFromChannel, get, put

reach leader(0)
-- reach finished
```

Figure 9.6: A MAPA model of a leader election protocol with three nodes.

Properties of interest. After having modelled the protocol, we used it to compute three properties of interest:

1. What is the probability that the first node is eventually elected to be leader? (*unbounded reachability objective*)
2. What is the probability that a leader is elected within 5 time units? (*time-bounded reachability objective*, error bound 0.01)
3. What is the expected time until a node is elected to be leader? (*expected time objective*)

Note that, in principle, the nondeterministic nature of the model implies that these properties each have no single answer but an *interval* of values. However, for this model the nondeterminism is only involved with the ordering of events. Since this ordering does not influence the observable behaviour in any way, the minimum and maximal probabilities and expected times referred to by these properties coincide. Hence, we only computed the minimal properties and expected times, and present these together with their computation times.

Results. We used the reachability condition `reach leader(0)` (as depicted in Figure 9.6) for SCOOP to generate the goal sets for the first property. For the second and third property, we changed this to `reach finished`[8]. We generated and analysed the model for different values of the variable `DataSize`, and also (easily) extended the model to incorporate four nodes instead of three. For all instances, we computed the three properties of interest in the presence and absence of our reduction techniques.

The results are presented in Table 9.7, where we denote by `leader-i-j` the variant with `i` nodes and `j`-sided dice. As expected, for N nodes the probability of becoming leader is $\frac{1}{N}$. Additionally, the expected time until a leader is chosen decreases when using a die with more sides and increases in the presence of more nodes. This is no surprise either, since larger dice decrease the probability that two nodes roll the same number, while an increase in the number of nodes increases this probability.

Discussion. Our basic reduction techniques are of significant importance to quickly generate the model's state space with SCOOP, as demonstrated by reductions of up to 98% due to these techniques. Indeed, for the variation with three nodes the number of MLPE parameters decreases from 42 to 26, and the complete MLPE decreases in size from 7,576 to 1,678 symbols. This reduces the size of the representation of states, makes it faster to compute a state's successors, and speeds up reduction techniques.

The results also indicate that dead variable reduction becomes more powerful when the size of the data type increases. This was to be expected, since variable resets obviously have more impact in this case. After all, more states are then mapped to the same state and hence relatively more reduction is obtained. We note that the resets take place in the asynchronous channels. If we altered the channel processes to not store the messages but just immediately forward them, no resets would be needed anymore. The exceptional 32% increase in state space generation time for `leader-4-2` with dead variable reduction is due to a 1.1 seconds overhead to find all variable resets. Since the model is rather small, this overhead is not fully compensated by the decrease in state space size.

Confluence reduction appears to work quite well for this model. State space reduction of up to 91% and analysis time reduction of up to 97% demonstrate the great potential of this technique. This may be attributed to the large degree of interleaving with little communication of this system. For this model it turned out

[8]This change slightly influences the reduction power of confluence in a positive way. The results we present on state space generation and reduction in the first few columns of Table 9.7 all concern the first model. Hence, they are a conservative illustration of confluence's capabilities—for the model used for checking the second and third property some more reduction is obtained.

	Original state space			Property 1		Property 2		Property 3	
Specification	States	Transitions	SCOOP (s)	Result	IMCA (s)	Result	IMCA (s)	Result	IMCA (s)
`leader-3-4`	11,714	14,814	2.0 (64.1)	0.33	0.5	0.87	51.8	2.92	2.0
`leader-3-6`	51,253	65,208	8.8 (400.7)	0.33	1.9	0.92	226.9	2.49	7.1
`leader-3-8`	148,512	189,606	26.3 (1,507.5)	0.33	5.2	0.94	725.8	2.31	19.9
`leader-3-10`	342,179	437,856	60.9 (4,281.5)	0.33	12.1	0.95	1,770.8	2.20	45.2
`leader-4-2`	9,843	13,372	2.5 (85.1)	0.25	0.7	0.36	87.7	6.67	3.6
`leader-4-3`	68,772	96,640	17.1 (827.6)	0.25	4.4	0.66	816.6	4.49	21.9
`leader-4-4`	271,773	388,460	68.1 (4,190.8)	0.25	19.0	0.77	4,062.7	3.72	87.9

	Dead variable reduction			Property 1		Property 2		Property 3	
Specification	States	Transitions	SCOOP (s)	Result	IMCA (s)	Result	IMCA (s)	Result	IMCA (s)
`leader-3-4`	7,265 (−38%)	9,339 (−37%)	1.6 (−20%)	0.33	0.3 (−40%)	0.87	30.8 (−41%)	2.92	1.2 (−40%)
`leader-3-6`	25,258 (−51%)	32,688 (−50%)	4.7 (−47%)	0.33	0.9 (−53%)	0.92	109.5 (−52%)	2.49	3.4 (−52%)
`leader-3-8`	60,795 (−59%)	78,957 (−58%)	10.8 (−59%)	0.33	2.0 (−62%)	0.94	270.5 (−63%)	2.31	7.4 (−63%)
`leader-3-10`	119,852 (−65%)	155,994 (−64%)	21.1 (−65%)	0.33	4.0 (−67%)	0.95	595.7 (−66%)	2.20	14.9 (−67%)
`leader-4-2`	8,467 (−14%)	11,600 (−13%)	3.3 (+32%)	0.25	0.6 (−14%)	0.36	74.8 (−15%)	6.67	3.1 (−14%)
`leader-4-3`	50,720 (−26%)	72,036 (−25%)	13.9 (−19%)	0.25	3.0 (−32%)	0.66	573.3 (−30%)	4.49	15.2 (−31%)
`leader-4-4`	175,401 (−35%)	253,760 (−35%)	46.1 (−32%)	0.25	12.1 (−36%)	0.77	2,608.1 (−36%)	3.72	55.6 (−37%)

	Confluence reduction (store representation map)			Property 1		Property 2		Property 3	
Specification	States	Transitions	SCOOP (s)	Result	IMCA (s)	Result	IMCA (s)	Result	IMCA (s)
`leader-3-4`	2,232 (−81%)	2,524 (−83%)	1.7 (−15%)	0.33	0.1 (−80%)	0.87	2.8 (−95%)	2.92	0.1 (−95%)
`leader-3-6`	7,440 (−85%)	8,321 (−87%)	6.1 (−31%)	0.33	0.3 (−84%)	0.92	9.5 (−96%)	2.49	0.3 (−96%)
`leader-3-8`	17,564 (−88%)	19,554 (−90%)	16.8 (−36%)	0.33	0.6 (−88%)	0.94	23.2 (−97%)	2.31	0.8 (−96%)
`leader-3-10`	34,244 (−90%)	38,031 (−91%)	38.0 (−38%)	0.33	1.2 (−90%)	0.95	62.8 (−96%)	2.20	1.8 (−96%)
`leader-4-2`	2,462 (−75%)	3,120 (−77%)	2.4 (− 4%)	0.25	0.1 (−86%)	0.36	5.8 (−93%)	6.67	0.2 (−94%)
`leader-4-3`	12,588 (−82%)	15,624 (−84%)	12.0 (−30%)	0.25	0.4 (−91%)	0.66	39.6 (−95%)	4.49	0.9 (−96%)
`leader-4-4`	40,790 (−85%)	50,614 (−87%)	42.4 (−38%)	0.25	1.4 (−93%)	0.77	138.4 (−97%)	3.72	2.6 (−97%)

	All reduction techniques			Property 1		Property 2		Property 3	
Specification	States	Transitions	SCOOP (s)	Result	IMCA (s)	Result	IMCA (s)	Result	IMCA (s)
`leader-3-4`	1,689 (−86%)	1,981 (−87%)	1.6 (−20%)	0.33	0.0 (−99%)	0.87	2.2 (−96%)	2.92	0.1 (−95%)
`leader-3-6`	5,250 (−90%)	6,131 (−91%)	4.0 (−55%)	0.33	0.1 (−95%)	0.92	6.5 (−97%)	2.49	0.2 (−97%)
`leader-3-8`	11,915 (−92%)	13,905 (−93%)	8.7 (−67%)	0.33	0.3 (−94%)	0.94	13.8 (−98%)	2.31	0.4 (−98%)
`leader-3-10`	22,652 (−93%)	26,439 (−94%)	16.6 (−73%)	0.33	0.6 (−95%)	0.95	27.2 (−98%)	2.20	0.7 (−98%)
`leader-4-2`	2,204 (−78%)	2,859 (−79%)	3.3 (+32%)	0.25	0.1 (−86%)	0.36	5.2 (−94%)	6.67	0.2 (−94%)
`leader-4-3`	10,394 (−85%)	13,366 (−86%)	10.8 (−37%)	0.25	0.4 (−91%)	0.66	33.1 (−96%)	4.49	0.7 (−97%)
`leader-4-4`	31,934 (−88%)	41,344 (−89%)	31.9 (−53%)	0.25	1.1 (−94%)	0.77	107.2 (−97%)	3.72	2.0 (−98%)

Table 9.7: State space generation and analysis for a leader election protocol.

that SCOOP's default mode—which does not store the representation map and recomputes representatives if needed again—slowed down state space generation considerably. For instance, `leader-3-6` took about 12 seconds to generate against 8.8 seconds without confluence reduction. Although the reduction of 217.4 seconds (96%) for time-bounded reachability analysis significantly makes up for this, we decided to have SCOOP remember part of the representation map—as discussed on page 207. This way, at the cost of some additional memory usage (in this case it roughly doubles), generation time is reduced to 6.1 seconds.

The combination of all our reduction techniques proves to be the most efficient. On average, IMCA analysis is sped up by a factor of more than 20 (i.e., −95%).

9.3.3 Polling system

Next, we investigate a polling system based on the description in Example 1.1. Again, inspired by [Sri91] we consider the idea of having a server processing jobs that arrive at multiple stations. As often happens in queueing theory, the arrival times as well as the service times are assumed to be exponentially distributed. As opposed to the description before, we now allow more than two arrival stations, as illustrated in Figure 9.8. These arrival stations receive jobs from the environment, and can store several of them until polled by the server in last-in-first-out manner—obviously, other strategies are conceivable.

We assume that the deletion of jobs from a station fails with some probability, and we follow previous work by *not* deterministically fixing the order in which the stations are polled by the server. However, whereas [Sri91] took a probabilistic approach to fix this order (to enable manual mathematical reasoning), we are able to omit these probabilities and rely on nondeterminism. That way, we can investigate the impact of the polling order by computing *minimal* and *maximal* values for each property of interest. Hence, this case study really requires all essential features of MAs: Markovian delays, probability and nondeterminism.

To make the system more interesting and more easily scalable, we assume

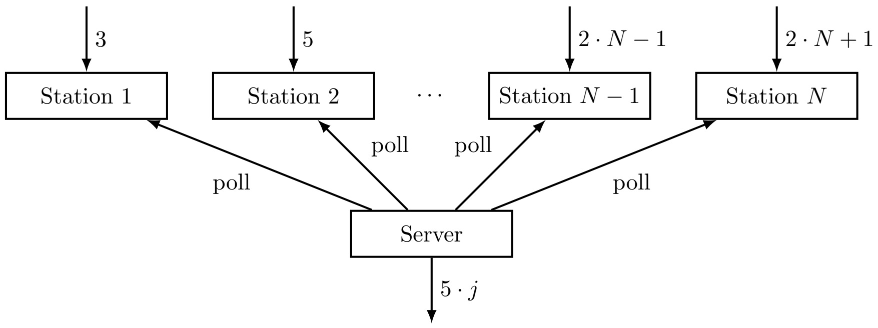


Figure 9.8: A polling system. Jobs arrive with rates that are different per arrival station. The server polls the stations for work, and processes jobs with a service rate depending on the type of job j.

```
type Stations = {1..3}
type Jobs     = {1..NrOfJobTypes}

Station(i:Stations, q:Queue) =
    size(q) < QueueSize
     => <2 * i + 1> . sum(j:Jobs, arrive(j) .
                      Station[q := add(q,j)])

 ++ size(q) > 0
     => deliver(i, head(q)) . psum(0.9 -> Station[q := tail(q)]
                              ++ 0.1 -> Station[])

Server = sum(n:Stations, sum(j:Jobs, poll(n,j)   . <5*j> .
                                       complete(j) . Server[]))

init Station[1, empty] || Station[2, empty] || Station[3, empty] || Server[]

comm (poll, deliver, copy)

hide copy, arrive, complete

encap poll, deliver

reachCondition size(q_1) = QueueSize &
               size(q_2) = QueueSize &
               size(q_3) = QueueSize
```

Figure 9.9: A MAPA model of a polling system with three arrival stations.

that there are several types of jobs. The service time of each job depends on its type, but its arrival time does not. We do allow arrival times to depend on the arrival stations; each arrival station can have a different rate of incoming jobs. We assume the failure probability of deletion to be $\frac{1}{10}$.

Modelling. The specification is depicted in Figure 9.9. We chose to have an arbitrary number of job types, depending on a variable `NrOfJobTypes`. Job types are represented as integers starting from 1, and job type j has a service rate given by $5j$. In the specification depicted here we assume the existence of three arrival stations, numbered from 1 to 3, each station i having an arrival rate given by $2i + 1$.

The `Server` process is rather simple: it nondeterministically chooses an arrival station `n` to poll a job from, and then is able to receive any job `j` by the action `poll(n,j)`. After receiving a job, it processes it with rate `5*j`, completes and tries to poll the next job. The `Station` processes are identified by a natural number and contain a queue to store incoming jobs. If a station's queue is not full yet, jobs can arrive. As mentioned before, the arrival rate is fixed per arrival station, but we do not assume anything about the incoming jobs themselves; there is a nondeterministic choice deciding their type. If a station's queue contains at least one job, it can be polled by the server. Then, with probability $\frac{9}{10}$ it is indeed removed from the queue, otherwise it erroneously remains there.

The complete system is composed of one server and (in this case) three arrival stations. Since our properties will be concerned with the situation that all arrival stations have reached their full capacity, we added a reachability condition (using the keyword `reachCondition`) that checks for precisely this situation. A `reachCondition` is specified over the global state space of the system; hence, we must refer to the `q` variables of the individual arrival stations. SCOOP has been implemented in such a way that each variable gets suffixed by an index, indicating the position of its process in the `init` specification. Hence, we can refer to the first arrival station's queue by `q_1`, and so on.

Properties of interest. Our analysis is focused on the situation that all arrival buffers are full, as in this case no jobs can arrive anymore. We use IMCA to compute three properties concerning this situation:

1. What is the expected time until reaching full capacity for the first time? (*expected time objective*)
2. What is the probability that full capacity is already reached within the first two time units? (*time-bounded reachability objective*, error bound 0.01)
3. What is the fraction of time that all arrival stations are at full capacity in the long run? (*long run objective*)

Due to nondeterminism in both the polling order and the job types that arrive, none of these properties has one definite answer. They all depend on how these two nondeterministic choices are resolved. Clearly, to keep the long-run average fraction of reaching full capacity low, the server should always first poll the arrival station with the lowest arrival rate. Additionally, the environment could help keeping this fraction low by only providing jobs that have a high service rate. Hence, for each property we provide an *interval* $[\min, \max]$ that contains all values that may be obtained for different schedulings.

Results. We investigate the effect of changes in the number of job types, the queue sizes in the arrival stations, and the number of arrival stations. For each dimension we provide three experiments, showing the intervals computed by IMCA for all three properties of interest, the time to generate and analyse these models and the reductions in these times as well as the state space sizes. Additionally, since confluence reduction seemed to be particularly powerful in the presence of many arrival stations and little data, we included a variation with nine stations all having just a single-element buffer and one type of incoming job. The results are presented in Table 9.10, where `polling-i-j-k` denotes a system with `i` stations, all having buffers of size of `j` and `k` types of jobs.

Discussion. It turns out that the model does not allow for much reduction by our main techniques dead variable reduction and confluence reduction, but still they decrease analysis time slightly. Dead variable reduction automatically deduces that the job type is irrelevant after the delay $5j$ in the `Server` process, due to the `complete(j)` action being hidden.

Confluence reduction automatically detects that the hidden `complete(j)` action is confluent and hence does not need to interleave with other actions.

	Original state space			Property 1		Property 2		Property 3	
Specification	States	Transitions	SCOOP (s)	Result	IMCA (s)	Result	IMCA (s)	Result	IMCA (s)
polling-3-2-2	2,600	4,909	0.3 (0.7)	[1.10, 3.92]	5.1	[0.37, 0.92]	61.3	[0.04, 0.47]	7.4
polling-3-2-3	21,466	43,614	2.5 (7.0)	[1.10, 9.32]	125.6	[0.17, 0.92]	759.4	[0.01, 0.47]	901.0
polling-3-2-4	109,812	232,863	13.7 (43.0)	[1.10, 22.15]	2,470.9	[0.07, 0.92]	7,828.5	timeout (10,000)	
polling-3-2-2	2,600	4,909	0.3 (0.7)	[1.10, 3.92]	5.1	[0.37, 0.92]	61.3	[0.04, 0.47]	7.4
polling-3-3-2	26,328	51,309	3.3 (7.3)	[1.48, 11.95]	176.4	[0.11, 0.81]	712.1	[0.01, 0.47]	3,345.4
polling-3-4-2	235,448	465,133	28.8 (63.9)	[1.85, 35.50]	7,630.4	timeout (10,000)		timeout (10,000)	
polling-2-2-2	331	538	0.0 (0.1)	[1.55, 4.58]	0.6	[0.32, 0.75]	3.6	[0.03, 0.29]	0.1
polling-3-2-2	2,600	4,909	0.3 (0.7)	[1.10, 3.92]	5.1	[0.37, 0.92]	61.3	[0.04, 0.47]	7.4
polling-4-2-2	20,241	42,544	3.0 (8.7)	[0.95, 3.76]	47.0	[0.38, 0.97]	1,050.9	[0.04, 0.57]	2,416.6
polling-9-1-1	3,849	7,954	1.1 (4.6)	[0.48, 0.69]	1.3	[0.97, 1.00]	1,689.4	[0.31, 0.79]	6.6

	Dead variable reduction			Property 1		Property 2		Property 3	
Specification	States	Transitions	SCOOP (s)	Result	IMCA (s)	Result	IMCA (s)	Result	IMCA (s)
polling-3-2-2	2,257 (−13%)	4,566 (−7%)	0.3 (− 0%)	[1.10, 3.92]	4.4 (−14%)	[0.37, 0.92]	53.7 (−12%)	[0.04, 0.47]	6.0 (−19%)
polling-3-2-3	17,072 (−20%)	39,220 (−10%)	2.1 (−16%)	[1.10, 9.32]	102.6 (−18%)	[0.17, 0.92]	612.1 (−19%)	[0.01, 0.47]	877.0 (− 3%)
polling-3-2-4	82,029 (−25%)	205,080 (−12%)	11.4 (−17%)	[1.10, 22.15]	1,792.9 (−27%)	[0.07, 0.92]	5,631.6 (−28%)	timeout (10,000)	
polling-3-2-2	2,257 (−13%)	4,566 (−7%)	0.3 (− 0%)	[1.10, 3.92]	4.4 (−14%)	[0.37, 0.92]	53.7 (−12%)	[0.04, 0.47]	6.0 (−19%)
polling-3-3-2	22,953 (−13%)	47,934 (− 7%)	3.0 (− 9%)	[1.48, 11.95]	153.9 (−13%)	[0.11, 0.81]	616.2 (−13%)	[0.01, 0.47]	2,994.4 (−10%)
polling-3-4-2	205,657 (−13%)	435,342 (− 6%)	26.5 (− 8%)	[1.85, 35.50]	6,674.7 (−13%)	[0.03, 0.65]	9,372.1	timeout (10,000)	
polling-2-2-2	282 (−15%)	489 (− 9%)	0.0	[1.55, 4.58]	0.4 (−33%)	[0.32, 0.75]	3.2 (−11%)	[0.03, 0.29]	0.1 (− 0%)
polling-3-2-2	2,257 (−13%)	4,566 (− 7%)	0.3 (− 0%)	[1.10, 3.92]	4.4 (−14%)	[0.37, 0.92]	53.7 (−12%)	[0.04, 0.47]	6.0 (−19%)
polling-4-2-2	17,840 (−12%)	40,143 (− 6%)	2.8 (− 7%)	[0.95, 3.76]	41.9 (−11%)	[0.38, 0.97]	937.9 (−11%)	[0.04, 0.57]	2,208.3 (− 9%)
polling-9-1-1	3,849 (− 0%)	7,954 (− 0%)	1.1 (− 0%)	[0.48, 0.69]	1.3 (− 0%)	[0.97, 1.00]	1,689.3 (− 0%)	[0.31, 0.79]	6.5 (− 2%)

	Confluence reduction			Property 1		Property 2		Property 3	
Specification	States	Transitions	SCOOP (s)	Result	IMCA (s)	Result	IMCA (s)	Result	IMCA (s)
polling-3-2-2	1,904 (−27%)	4,223 (−14%)	0.4 (+33%)	[1.10, 3.92]	3.9 (−24%)	[0.37, 0.92]	46.8 (−24%)	[0.04, 0.47]	5.5 (−26%)
polling-3-2-3	14,875 (−31%)	37,023 (−15%)	3.0 (+20%)	[1.10, 9.32]	89.5 (−29%)	[0.17, 0.92]	529.8 (−30%)	[0.01, 0.47]	743.9 (−17%)
polling-3-2-4	72,768 (−34%)	195,819 (−16%)	16.2 (+18%)	[1.10, 22.15]	1,595.3 (−35%)	[0.07, 0.92]	4,988.8 (−36%)	timeout (10,000)	
polling-3-2-2	1,904 (−27%)	4,223 (−14%)	0.4 (+33%)	[1.10, 3.92]	3.9 (−24%)	[0.37, 0.92]	46.8 (−24%)	[0.04, 0.47]	5.5 (−26%)
polling-3-3-2	19,578 (−26%)	44,559 (−13%)	4.0 (+21%)	[1.48, 11.95]	133.6 (−24%)	[0.11, 0.81]	539.0 (−24%)	[0.01, 0.47]	2,510.9 (−25%)
polling-3-4-2	175,866 (−25%)	405,551 (−13%)	36.7 (+27%)	[1.84, 35.50]	5,674.6 (−26%)	[0.03, 0.65]	7,896.7	timeout (10,000)	
polling-2-2-2	233 (−30%)	440 (−18%)	0.0	[1.55, 4.58]	0.4 (−33%)	[0.32, 0.75]	2.5 (−31%)	[0.03, 0.29]	0.1 (− 0%)
polling-3-2-2	1,914 (−26%)	4,223 (−14%)	0.4 (+33%)	[1.10, 3.92]	3.9 (−24%)	[0.37, 0.92]	46.8 (−24%)	[0.04, 0.47]	5.5 (−26%)
polling-4-2-2	15,439 (−24%)	37,742 (−11%)	3.9 (+30%)	[0.95, 3.76]	36.2 (−23%)	[0.38, 0.97]	803.7 (−24%)	[0.04, 0.57]	2,030.9 (−16%)
polling-9-1-1	1,033 (−73%)	5,138 (−35%)	1.2 (+ 9%)	[0.48. 0.69]	0.4 (−69%)	[0.97, 1.00]	439.3 (−74%)	[0.31, 0.79]	1.3 (−80%)

Table 9.10: State space generation and analysis for a polling system. The combination of dead variable reduction and confluence reduction is omitted, as these results correspond to the situation of applying only confluence reduction.

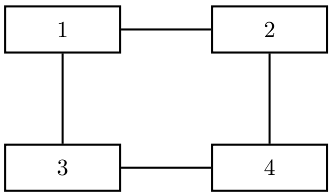


Figure 9.11: A 2×2 processor grid.

However, since there are never other actions enabled at the same time, no nondeterminism is removed due to this action being confluent—as nondeterminism is the main source of complexity of the analysis algorithms, not much reductions in analysis time are obtained. The only impact is that the intermediate states between `5*j` and `complete(j)` can be omitted. This yields some reduction in state space size (actually quite a large reduction if the number of arrival stations increases), but also renders unnecessary the reset made by dead variable reduction. Hence, the reductions made due to confluence in this case study subsume those obtained by means of dead variable resets. Therefore, confluence reduction and dead variable reduction together behave exactly the same as confluence reduction on its own, and so we excluded the results for the combination of these techniques from the table.

9.3.4 Processor grid

Our last case study is a 2×2 concurrent processor architecture, parameterised in the level k of multitasking (taken from Figure 11.7 in [ABC+94] and shown in Figure 9.12). The model was given as a Generalised Stochastic Petri Net (GSPN), so we used the GEMMA tool [Bam12] to transform it to the MAPA language.

Every processor is assumed to have k tasks to work on, each consisting of two parts. First, the processor needs to do some work itself during the task's local operation phase, which is assumed to be exponentially distributed with rate 1. Second, some help is needed from a neighbouring processor; to this end, a request is send to one of the neighbours to work on this second phase of the task. The neighbour works on the task for a period of time assumed to be exponentially distributed with rate 10. During the time that a neighbour is working on the second phase of a task, a processor can already start working on the local part of the next task. Each processor has two neighbours: one in the vertical dimension of the grid and one in the horizontal dimension (as illustrated in Figure 9.11).

The specification is based on *preemptive interaction*. That is, helping out neighbours with the second phase of their task is considered to have a higher priority than working on one's own local operation phase. Hence, if neighbours ask for help, local work is interrupted (but work for one neighbour is not interrupted by a request from another neighbour). After finishing a neighbour's job, local work is restarted[9].

[9]Due to the assumption that service times are distributed exponentially and hence are memoryless, there is no difference between restarting and resuming a job.

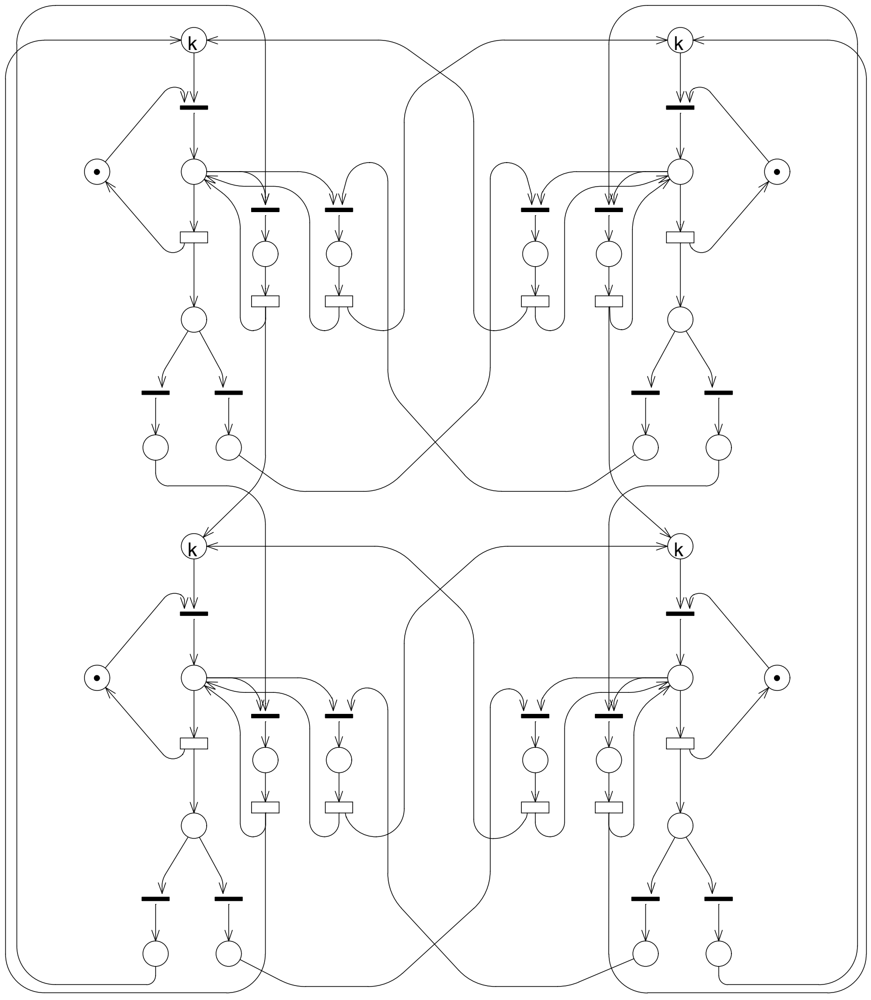


Figure 9.12: GSPN model of a 2 × 2 processor grid with multitasking and preemptive interaction (taken from [ABC+94]).

Modelling. For this case study we did not have to design a model ourselves; it was already provided by [ABC+94] in the form of the GSPN depicted in Figure 9.12. We note that the figure does not present all necessary information; the upper two timed transitions (indicated by open rectangles) have rate 1, the four timed transitions below have rate 10, and similarly for the lower half of the model. Additionally, all immediate transitions (indicated by closed rectangles) have weight 1. We do not go into details on the specifics on the model, neither do we discuss the semantics of GSPNs. Rather, we refer to [ABC+94] for both and to [Bam12] for a detailed explanation on how to translate GSPN models to

MAPA. For this chapter it suffices to say that the GSPN represents the system described above, and that we used the PIPE (Platform Independent Petri net Editor) tool [PIP13] to draw this model and the GEMMA tool to obtain an equivalent MAPA specification (actually, an MLPE). That way, every place of the GSPN is modelled as a variable and every transition as a summand.

The GSPN community is used to assign *weights* to immediate transitions, basically resolving nondeterminism by changing it into probabilistic branching. Hence, previous analysis always required weights for all immediate transitions, requiring complete knowledge of the mutual behaviour of all these transitions. That way, GSPNs reduce to CTMCs and relatively simple calculations can be employed to compute for instance the long-run average fraction of time that a GSPN is in a certain state or set of states. Analysis in [ABC$^+$94] on the GSPN of the processor grid described above indeed assumed all nondeterministic choices to be resolved uniformly. That is, if N immediate transitions are enabled simultaneously, only one of them is taken and the probability for each of them to be chosen is $\frac{1}{N}$. For the 2×2 processor grid, this most importantly specifies that a processor randomly chooses between its two neighbours for requesting work on the second phase of its tasks. This, however, may not always be the best strategy, neither may it accurately represent actual behaviour in a practical scenario.

When translating GSPNs to MAPA, we now *are* able to retain nondeterminism [Kat12, EHKZ13]. We actually even allow a weight assignment to just a (possibly empty) subset of the immediate transitions—reflecting the practical scenario of only knowing the mutual behaviour for a *selection* of the transitions.

Properties of interest. As in [ABC$^+$94], we are interested in computing the throughput of one of the processors (say processor 1 in Figure 9.11), given by the long-run average of having a token in a certain place of the GSPN (the place below the one on the top-left containing the letter `k` in Figure 9.12). In [ABC$^+$94], symmetry dictated that all processors have exactly the same behaviour. Hence, it did not matter which processor was under investigation. However, when breaking the symmetry by giving one processor a nondeterministic choice between its neighbours for service requests, differences may occur. Hence, we compute three properties for various instances of the system:

1. What is the throughput of processor 1? (*long run objective*)
2. What is the throughput of processor 2? (*long run objective*)
3. What is the throughput of processor 4? (*long run objective*)

To investigate the impact of the amount of nondeterminism on the throughput as well as the effects of the reduction techniques, we consider three variations in the assumptions on the behaviour of the grid:

- All nondeterministic choices are resolved uniformly. That is, we retain all weights from the original specification [ABC$^+$94]. This variation has no nondeterminism, only probability.
- Processors 2, 3 and 4 still use a uniform distribution to decide on their neighbour to send a service request to. Hence, these weights are retained.

All other nondeterministic behaviour, including processor 1's choice of neighbour, stays nondeterministic.

- No nondeterministic choices are resolved uniformly. That is, all weights from the original specification are omitted. This variation has no probability, only nondeterminism.

The second variation still has three of the four processors choose their neighbour to send a service request to uniformly, while the remaining processor has the liberty of choosing in any way it wants to. This situation contains nondeterminism as well as probability (and Markovian delays), and hence asks for the full spectrum of MAs. Clearly, the introduction of nondeterminism for the second and third variation implies that each answer will again be an *interval* instead of a single value. Also, note that all processors are completely equivalent in the first and third variation. Therefore, we only compute the first property for these two scenarios; processor 2 and 4 have the same throughput anyway.

Results. Table 9.13 presents the results of this case study. We denote by `grid-o-k` the original model (with all weights retained and hence all nondeterministic choices resolved probabilistically) and `k` concurrent tasks. Similarly, `grid-i-k` is the intermediate model with some weights retained and `k` concurrent tasks, and `grid-n-k` is the fully nondeterministic model with `k` concurrent tasks. The results for `grid-o-2`, `grid-o-3` and `grid-o-4` indeed correspond to the results in Table 11.5 from [ABC$^+$94]. However, existing work was not yet able to indicate the accuracy of these results in case processors do not resolve their nondeterministic choices uniformly. We now were able to investigate this, resulting in the intervals presented for the moderately and fully nondeterministic variants of the model. This shows us that for instance in the presence of two tasks, the throughput of our first processor is at least 0.81 and at most 1.00, depending on the processors' behaviour. This information is much more trustworthy than the original result of 0.903, which tells us nothing if the strong assumption on uniform resolutions is violated.

The results also indicate that the intermediate situation where only processor 1 can choose freely allows just a slight variation in throughput. This can be explained by noticing that it could send its service requests to the neighbour that is currently not already working on another service request (either coincidentally or maybe due to some shared information on who is working on what). That way, work immediately starts and no additional waiting is necessary. As service requests are dealt with rather quickly, the potential increase or decrease in throughput for processor 1 is rather small. However, the neighbouring processors may endure a much larger decrease in throughput as a result of processor 1 slightly increasing its throughput. This is explained by the fact that processor 1 may try to preempt neighbours that are working on their local operation phase. Since that phase on average takes much longer than the service requests, the preemptions may severely impact their throughput (as the memoryless aspect of the exponential distribution dictates that preempted work basically starts all over). If all processors can choose freely, processor 1 may obtain an almost perfect throughput. This presumably is due to its possibility of selecting a

	Original state space			Property 1		Property 2		Property 3	
Specification	States	Transitions	SCOOP (s)	Result	IMCA (s)	Result	IMCA (s)	Result	IMCA (s)
`grid-o-2`	2,508	3,215	14.5 (17.7)	0.9031	2.0				
`grid-o-3`	10,852	14,379	64.7 (79.5)	0.9086	32.8				
`grid-o-4`	31,832	42,879	193.0 (236.7)	0.9090	249.7				
`grid-i-2`	2,508	4,254	0.8 (0.9)	[0.9001, 0.9055]	3.0	[0.8585, 0.9479]	2.6	[0.9029, 0.9032]	3.1
`grid-i-3`	10,852	19,099	3.2 (3.9)	[0.9081, 0.9089]	84.4	[0.8633, 0.9541]	67.1	[0.9086, 0.9087]	77.2
`grid-i-4`	31,832	56,704	9.8 (11.9)	[0.9089, 0.9091]	868.2	[0.8636, 0.9545]	765.2	[0.9090, 0.9091]	937.3
`grid-n-2`	2,508	4,608	0.6 (0.6)	[0.8110, 0.9953]	2.7				
`grid-n-3`	10,852	20,872	2.7 (2.6)	[0.8173, 0.9998]	64.4				
`grid-n-4`	31,832	62,356	7.9 (8.0)	[0.8181, 1.0000]	914.4				

	Confluence reduction			Throughput		Throughput neighbours		Throughput non-neighbour	
Specification	States	Transitions	SCOOP (s)	Result	IMCA (s)	Result	IMCA (s)	Result	IMCA (s)
`grid-o-2`	1,950 (−22%)	2,657 (−17%)	18.3 (+26%)	0.9031	1.2 (−40%)				
`grid-o-3`	8,568 (−21%)	12,095 (−16%)	81.9 (+27%)	0.9086	22.0 (−33%)				
`grid-o-4`	25,177 (−21%)	36,224 (−16%)	243.8 (+26%)	0.9090	162.5 (−35%)				
`grid-i-2`	1,514 (−40%)	2,767 (−35%)	0.8 (− 0%)	[0.9001, 0.9055]	1.4 (−53%)	[0.8585, 0.9479]	1.2 (−54%)	[0.9029, 0.9032]	1.5 (−52%)
`grid-i-3`	6,509 (−40%)	12,523 (−34%)	3.5 (+ 9%)	[0.9081, 0.9089]	30.3 (−64%)	[0.8633, 0.9541]	25.2 (−62%)	[0.9086, 0.9087]	38.6 (−50%)
`grid-i-4`	19,025 (−40%)	37,418 (−34%)	10.1 (+ 3%)	[0.9089, 0.9091]	398.8 (−54%)	[0.8636, 0.9545]	348.2 (−54%)	[0.9090, 0.9091]	555.7 (−41%)
`grid-n-2`	1,514 (−40%)	3,043 (−34%)	0.7 (−30%)	[0.8110, 0.9953]	1.1 (−59%)				
`grid-n-3`	6,509 (−40%)	13,738 (−34%)	3.0 (+11%)	[0.8173, 0.9998]	22.4 (−65%)				
`grid-n-4`	19,025 (−40%)	41,018 (−34%)	8.8 (+11%)	[0.8181, 1.0000]	361.4 (−60%)				

Table 9.13: State space generation and analysis for a processor grid.

neighbour that is currently not already working on a service request, and on the neighbours' possibility of sparing processor 1 from any work. On the other hand, the neighbours 2 and 3 may also direct all their work to processor 1, lowering its throughput significantly.

Discussion. Based on the results, we can conclude that the basic reduction techniques do not help us much. This is understandable, as the construction of an MLPE based on a GSPN does not introduce any summations or constants; hence, at most some very basic expression simplifications may be possible. Additionally, dead variable reduction has no impact—therefore, these results have even been omitted. This was to be expected as well, since the only variables in an MLPE based on a GSPN are the ones that represent the number of tokens in each place. This number can only become irrelevant for places that have no outgoing transitions or that are somehow reset at some point. As neither of these situations occurs in the model at hand, indeed dead variable reduction is not applicable.

Confluence reduction may only reduce in the presence of invisible non-probabilistic transitions. Hence, in the original model not much reduction is possible. The fact that some state space reduction is still obtained is due to the fact that states having only one outgoing τ-transition may sometimes be omitted. As expected, more reduction is possible if more nondeterminism is present (since indeed some of this nondeterminism turns out to be spurious). State space reductions of about 40% and decreases in analysis time around 60% save a significant amount of time.

9.4 Contributions

In this chapter we presented our tool SCOOP, implementing the MAPA linearisation procedure, a state space generation algorithm and all reduction techniques introduced for MAPA in this thesis. We discussed four case studies, showing how to model and analyse them and illustrating the power of our reduction techniques.

Our case studies show that several types of systems can easily be modelled in MAPA. Enabling the full spectrum of MAs, and supported by IMCA, our techniques for the first time allow the efficient modelling, generation and analysis of systems incorporating both nondeterminism, probabilistic choice and Markovian delays. For the GSPN case study on a processor grid, this made it possible to compute *intervals* of probabilities, taking into account the effects of how nondeterministic choices are resolved. This was not possible in earlier work.

Our results also clearly show the significant potential of confluence reduction, dead variable reduction and the basic reduction techniques. We demonstrated that all of them may greatly reduce state space generation time. Additionally, dead variable reduction and confluence reduction often cut back the state space itself and hence the analysis time. In the remainder of this section we discuss to what extent our reduction techniques approach the smallest possible representations of our case studies. Additionally, we touch upon some trends regarding the reduction power of the individual techniques. Finally, we briefly discuss the reductions in analysis time.

9.4.1 Reduction potential

Although our techniques reduce state spaces significantly, they in general do not obtain the smallest possible model that still satisfies the same properties. After all, in general this would require the entire state space to be known upfront.

Hence, we are interested to know how much of the potential reductions are indeed obtained by means of our techniques. Since there are no tools available yet to compute the branching bisimulation minimal quotient for MAs, we actually have no means to derive the maximal possible reduction for models that exhibit nondeterminism, probability as well as Markovian delays at the same time. However, for subsets of these features we can rely on older tools that compute traditional branching bisimulation in the presence of only nondeterminism (as is the case for the handshake register), probabilistic branching bisimulation in the presence of only nondeterminism and probabilistic choice (as is the case for a slight variation of the leader election protocol) and stochastic branching bisimulation in the presence of only nondeterminism and Markovian delays (as is the case for one of the variations of the processor grid).

Handshake register. The case study of the handshake register contains no quantitative information; hence, we can just apply μCRL's `ltsmin` tool for computing its minimal quotient with respect to branching bisimulation.

For the handshake register with $|\mathtt{D}| = 2$, the state space went down from 540,736 to 25,280 states due to our techniques. In the minimal quotient, it turns out to have only 72 states. Hence, some additional reduction is still possible. Nonetheless, we managed to cut back over 95% of the maximum number of states that could have been omitted (515,456 of the maximum number of 540,664). For $|\mathtt{D}| = 3$, we went down from 13,834,800 to 155,304 states, while the minimal quotient has 189 states.

Leader election protocol. We omitted the Markovian delay from the specification of the leader election protocol to make it into a PA. Then, CADP's `bcg_min` tool can be applied to obtain a minimal quotient with respect to probabilistic branching bisimulation. Since CADP represents each probabilistic transition by means of an intermediate state with outgoing probabilistic edges, the number of states and transitions is not fully comparable to the numbers we provide. Still, we can report that the number of states in CADP of `leader-3-4` in the absence of Markovian delays is 21,232 without our reduction techniques, 2,939 when applying all reduction techniques and 242 in the branching bisimulation minimal quotient. This again shows that some additional reduction may be achieved, but also that we were able to reduce quite significantly: we omitted 87% of the maximum number of states that could be removed (18,293 out of 20,990). For `leader-3-8` the number of states without any reduction was 274,476, with reductions 21,151 and after minimisation 666—hence, here we found 93% of the maximal reduction.

Processor grid. For the GSPN case study of the processor grid, we took the variation without any weights (and hence without any discrete probabilitistic behaviour). Since that model is an IMC, we are able to apply CADP's `bcg_min` tool to obtain the minimal quotient with respect to stochastic branching bisimulation. For `grid-n-4` the number of states without any transitions was 31,832, which went down to 19,025 when applying all reduction techniques. The minimal quotient turned out to consist of 11,901 states. So, for this model our techniques managed to find 64% of the maximal reduction possible.

9.4.2 Individual reduction techniques

Based on the results for the four case studies, we draw conclusions about the applicability of the individual reduction techniques (the basic reduction techniques, dead variable reduction and confluence reduction). For each of these three categories, we also conjecture some model properties that seem to be indicators to expect a lot of reduction.

As demonstrated by the case studies, often the combination of all our techniques is most efficient. After all, this combines their forces (which are mostly rather orthogonal).

Basic reduction techniques. We noticed from the processor grid case study that the basic reduction techniques do not influence state space generation much for MAPA models generated from GSPNs. This was to be expected, due to the fact that the translation procedure does not introduce any real data. No summations are introduced and no variables can ever be constant (unless a place in the GSPN is never used). Hence, summation elimination and constant elimination are not applicable.

For the handshake register and polling system generation times roughly halved, while for the leader election protocol reduction factors of more than 70 were observed (requiring less than 1.5% of the original time to generate the state space). This could be explained from the relatively large number of parallel components, introducing a lot of constant parameters and reducible nondeterministic summations.

The reduction of the number of MLPE parameters reduces the size of the states that need to be stored in memory (vectors of values of these parameters). Additionally, smaller states and smaller specifications make it easier to check which summands are enabled for a given state. Both aspects speed up state space generation, and this speed up can be quite significant—as illustrated by our case studies.

Best results if a model is constructed as the parallel composition of many components having extensive interactions based on data parameters (introducing reducible summation and constant parameters), and if a model has lots of data and expressions over this data (introducing reducible expressions).

Dead variable reduction. The potential of dead variable reduction is most convincingly shown by the handshake register case study. There, state spaces were reduced by more than a factor of 100, leaving less than a percent of the original number of states and transitions. This way, a model could be generated that would have been much too big without dead variable reduction. On the other hand, the processor grid confirmed our expectation that GSPN-based models are not suitable for dead variable reduction. After all, these only contain variables storing the number of tokens in each place; often, this information never becomes irrelevant (as it basically encodes the control flow) and hence no variable resets are possible.

For the leader election protocol and the polling system, dead variable reduction was able to achieve significant reductions. As expected, these reductions become more powerful in case larger data types are used. After all, this implies that variable resets map more states on the same state than when using only small data types. For the polling systems moderate results were found, reducing analysis times by about 10%. This can be explained by the fact that this model only allowed one variable reset of rather limited influence. For the leader election protocol, on the other hand, more impressive reduction were obtained. Depending on the model parameters, dead variable reduction sometimes cut back analysis time to less than 3% of the time needed to analyse the unreduced model. This made the analysis of much larger models feasible in reasonable amounts of time (minutes instead of hours).

Best results if a model has many parameters with large data types, storing values that regularly become irrelevant.

Confluence reduction. Like dead variable reduction, also confluence reduction depends heavily on the structure of the model under consideration. For the handshake register state spaces reduced by approximately 40%, for the leader election protocol by approximately 80%–90%, for the polling system by approximately 25%-30% and for the processor grid by approximately 20%–40%. The case studies show little difference in the effect of confluence reduction when varying the parameters of the model. The processor grid did show that, as was to be expected, a nondeterministic model allows for more confluence reduction than a probabilistic model.

For the leader election protocol and the processor grid analysis times dropped even more significantly than the number of states of the model due to confluence reduction. This can be explained by the observation that the degree of nondeterminism reduced more than the number of states. For instance, for `leader-3-6` without confluence reduction there are 51,253 states of which 13,599 have a nondeterministic choice. With confluence reduction, 6,880 states remain, of which only 120 have a nondeterministic choice. Hence, the degree of nondeterminism goes down from 27% to 2%—apparently confluence reduction is able to detect that almost all nondeterministic choices are spurious. Since nondeterminism is the main source of complexity in the algorithms, this explains why the reduction in analysis time is even larger than the reduction in state space size.

The generation times by SCOOP are not reduced as much, due to the additional overhead of (re)computing representative states. To keep memory usage in the order of the reduced state space, the representation map is deliberately not stored by default and therefore potentially recomputed for some states. As explained in Section 9.3.2, we did store the representation map for the leader election protocol case study to save time at the cost of some additional memory usage.

Best results if a model is constructed as the parallel composition of many loosely connected components, all having many internal transitions that may interleave in any order. Since probabilistic transitions cannot be confluent, confluence reduction prefers nondeterminism over probabilistic choice.

Our heuristics underapproximate the confluent transitions, and hence may not find them all. Since no algorithm has yet been implemented to find the complete set of confluent transitions on a concrete state space, it is at this point infeasible to say if all confluent transitions were discovered. We can, however, take the state space when applying dead variable reduction as our baseline, and compare confluence reduction's performance to the maximal obtainable reduction with respect to branching bisimulation. For the handshake register with $|\mathtt{D}| = 2$, confluence brought the state space down from 45,504 to 25,280 states, with a minimal quotient of 72 states. Hence, 45% of the maximal reduction is obtained. Since confluence is more strict than branching bisimulation, we can reasonably expect more than 45% of the confluent transitions to have been detected.

For `leader-3-6` confluence got the number of states down from 44,581 to 9,267, with 430 as the absolute minimum. Hence, for this model at least 80% of the maximal possible reduction was obtained. For `grid-4` no dead variable reduction was possible, and hence the 64% discussed earlier is completely due to confluence reduction.

9.4.3 Reductions in analysis time

For the three case studies that were analysed with IMCA, we noticed (as expected) that decreases in the number of states resulted in decreases in the time needed for the computation of the various properties that we checked. Most of the algorithms used by IMCA (linear programming, value iteration) are rather expensive regarding their worst-case time complexity [ZN10], but are often reasonably fast in practice—as shown by our experiments. Therefore, no strong theoretical statements can be made about the impact of our reduction techniques on IMCA's analysis.

However, we did observe that if models were reduced by dead variable reduction, the relative reductions in analysis time corresponded quite well to the relative reductions in state space size. For the models where confluence reduction was able to actually decrease the amount of nondeterminism (leader election protocol and processor grid), the reduction in analysis time was relatively even larger.

In addition to (mostly) reducing state space generation time, our reduction techniques always reduced IMCA's analysis time. With only a few exceptions, this decrease was at least as large as the decrease in the number of states, relatively. Hence, especially if a model is to be analysed by IMCA, our techniques really pay off.

CHAPTER 10

Conclusions

"A moral being is one who is capable of reflecting on his past actions and their motives— of approving of some and disapproving of others."

Charles Darwin

We introduced MAPA: a novel process-algebraic language for specifying Markov automata (MAs). The MA is a recently introduced state-based modelling formalism incorporating nondeterminism, probabilistic choice and stochastic timing—all these features are also supported by MAPA. We defined several reduction techniques, working symbolically on specifications to optimise state space generation and analysis without ever having to generate an unreduced state space. Experimental validation shows that significant reductions are obtained, reducing the larger models between 35% and 99%.

10.1 Summary

10.1.1 The MAPA language: efficient modelling

The MAPA language is data driven, even allowing specifications that rely on dynamic data types such as stacks or lists. To simplify symbolic translations of MAPA specifications, we presented the notion of derivation-preserving bisimulation. We used it to show under which circumstances techniques can be defined for our language without even taking into account stochastic timing, while still being applicable to all specifications by means of an encoding of rates in actions.

We defined the MLPE as a restricted part of MAPA, easy to use for state space generation, parallel composition and symbolic optimisations. We demonstrated how to transform (linearise) each MAPA specification into an equivalent representation as MLPE, allowing MLPE-based reduction techniques to be applied to all specifications. The linearisation procedure is defined only on the subset of MAPA not including stochastic timing, but can be applied to the full language by means of our encoding scheme.

10.1.2 Reduction techniques: efficient generation and analysis

MAPA specifications often generate large state spaces—a well-known problem in (quantitative) model checking. This may be due to the presence of data, or due to a large number of parallel components. To alleviate this state space explosion, we presented five reduction techniques. All are defined on the MLPE, as hence are enabled through our linearisation procedure.

First we defined three basic reduction techniques: constant elimination, expression simplification and summation elimination. They syntactically clean-up an MLPE, making it more human friendly to read and speeding up state space generation. However, they do not influence the underlying state space. Our two other reduction techniques—dead variable reduction and confluence reduction—do aim to shrink a specification's MA, while preserving all properties of interest. As they work on the MLPE instead of the actual models, they prevent having to generate the unreduced state space.

Dead variable reduction automatically performs variable resets, reducing state spaces by merging several equivalent states. It incorporates a control flow analysis technique on MLPEs that does not only take into account control flow dictated by the program counters of the constituent processes, but also by control flow encoded in the data parameters. Although some of these resets may also be introduced manually, our technique relieves the user of that burden. Additionally, while manual optimisations are error-prone, our technique guarantees to only provide a bisimilar state space that is at most as large as the unreduced one.

Confluence reduction focusses on spurious interleavings. It basically gives priority to invisible transitions that do not disable any observable behaviour and hence can always be taken at the cost of other transitions emanating from the same state. We generalised a non-quantitative variant of this technique, while at the same time fixing a subtle mistake in the old definitions concerning closure under unions. We showed how to apply confluence to reduce state spaces on-the-fly, and discussed how to detect confluence symbolically based on the MLPE.

Additionally, we compared confluence reduction to partial order reduction in two ways. First, we provided a theoretical comparison between the two notions in branching time, demonstrating that confluence is slightly more powerful in that setting. We pinpointed precisely how the two techniques can be altered to make them coincide, showing that the differences we observed indeed completely characterise the gap. Second, we provided a practical comparison between confluence reduction an partial order reduction in the context of statistical model checking. We demonstrated how to apply confluence to its full potential on the partial concrete state spaces that are available during this type of model checking, and indeed showed that the advantages over partial order reduction can make a difference in practice.

10.1.3 Implementation and validation

We developed a tool called SCOOP, implementing a parser, lineariser and state space generator for our language MAPA. The tool applies all five reduction techniques presented in this thesis. It is linked to the preprocessor GEMMA to work with GSPNs as well, and to the model checker IMCA for the verification of time-bounded and unbounded reachability properties, expected times and long-run averages. Together, this tool chain (called MaMa) for the first time enables the efficient specification, generation and analysis of Markov automata.

We presented four case studies, on a handshake register, a leader election protocol, a polling system and a processor grid. They demonstrated that MAPA can be used to specify a large variety of systems. The fact that systems are represented as MAs allowed for the computation of minimal and maximal probabilities in the presence of stochastic timing, discrete probabilistic choice and nondeterminism, whereas earlier work required all nondeterminism to be resolved and hence always yielded single probabilities.

Experimental results showed that our techniques significantly reduce state space generation and analysis times. As expected, all reduction techniques turned out to have their own strengths. We were able to provide guidelines indicating for what type of systems each reduction technique is most applicable.

10.2 Discussion

The framework we presented enables the efficient modelling, generation and analysis of MAs. It contributes in several ways to the state of the art in quantitative model checking:

- The data-driven language MAPA supports the efficient modelling of systems incorporating nondeterminism, probabilistic choice and stochastic timing.
- The notion of derivation-preserving bisimulation provides the opportunity to reuse techniques developed for PAs also for specifications of MAs.
- The basic reduction techniques contribute to the efficient generation of MAs based on MAPA specifications.
- Dead variable reduction and confluence reduction alleviate the state space explosion by means of reductions in the number of states and transitions, allowing faster analysis.
- Our comparison of confluence reduction and partial order reduction resolves the long-standing uncertainty about the relation between these two concepts in branching time.

The practical applicability of our approach for specific case studies mainly depends on two aspects: (1) how accurate can the case study be represented as an MA, and (2) how severe is the state space explosion?

Accuracy of the model. Since MAs generalise LTSs, DTMCs, CTMCs, PAs and IMCs, all systems that can be modelled as one of these can also be modelled as an MA. Based on the abundant number of case studies that indeed have been modelled in one of these formalisms (for instance, [BG94a, KS94, FGK97, SvdZ98, GPW03, PS07, HHK00, FG06, KNP12, QWP99, YKNP06, KNP08, HKN$^+$08, KNS03, NS06, NM10, BCS10, BHH$^+$09]), we can conclude that there is a large target set of cases.

Already for CTMCs and IMCs, the main concern is whether the exponential distribution accurately describes the timing in a system. This is often assumed, for instance in queueing systems [BB96] and systems biology [Wil11, KNP08]. Additionally, it is well-known that every probability distribution can be approximated arbitrarily well by a phase-type distribution, i.e., a network of exponential distributions [BGdMT06, Nel95]. Although this is not always easy to achieve in practice [Bla05], a combination of exponential distributions may still be used to at least partly improve the accuracy of the model.

Surviving the state space explosion. Although our techniques mitigate the state space explosion to quite some extent, this does not yet fully solve the problem. In the presence of large data that cannot be reset, or multiple interleavings that do make a difference, the state space may still grow rather quickly. Hence, although we did lift the bar significantly, quantitative model checking of systems representable as MAs remains to be bound by the complexity of these systems. Nonetheless, our case studies (as well as many other case studies in quantitative model checking) demonstrated that interesting results can still be obtained, for instance by abstracting systems or protocols to their core or by only considering part of a system.

10.3 Future work

Our work gives rise to several directions for future work. Most obviously, the development of additional minimisation techniques preserving strong or branching bisimulation could be very useful. As our case studies demonstrated that a large fraction of the potential reduction is often already obtained by our current techniques, it would also be interesting to investigate the possibilities for reductions techniques preserving less properties, for instance only traces or even only deadlocks.

Instead of developing additional reduction techniques, future work may also focus on conducting more case studies, to evaluate the effects of our current reduction techniques and verify the generalisability of our results. Also, the potential of compositional state space generation may be explored. Additionally, the existing techniques may be investigated some more, as discussed below.

10.3.1 Reduction techniques

Dead variable reduction. It would be interesting to find additional applications for the control flow reconstructed as part of our dead variable reduction technique. One possibility is to use it for invariant generation, another is to visualise it such that process structure can be understood better. Also, it might be used to optimise confluence detection, since it could assist in determining which pairs of summands may be confluent. As also conjectured by [Pel08], such collaboration between reduction techniques could be useful.

Another direction for future work is based on the insight that the control flow graph is an abstraction of the state space. It could be investigated whether other abstractions, such as a control flow graph containing also the values of important data parameters, may result in more accurate data flow analysis.

Confluence reduction. As future work on confluence reduction we envision to search for even more powerful ways of using commutativity for state space reduction, for instance by allowing confluent transitions to be probabilistic.

Preferably, such extensions will enable more aggressive reductions that, instead of preserving the rather conservative notion of bisimulation we used, preserve the more powerful weak bisimulation from [EHZ10b]. Future work may also focus on finding more powerful heuristics for detecting confluence based on the MLPE, enabling more reduction in practice.

Another obvious direction of future work is to generalise the ample set method of partial order reduction to MAs, and prove its correctness by demonstrating again the subsumption by confluence reduction.

Finally, although we already showed that confluence reduction preserves branching bisimulation, it is still unknown precisely what types of properties are left invariant by this notion of equivalence. We conjectured (and confirmed empirically by all our experiment) that this is the case for all properties that can be verified by IMCA. Ongoing work in the Software Modeling and Verification group at RWTH Aachen University already formally showed this for strong bisimulation, and is currently concerned with also showing this for weak and branching bisimulation.

10.3.2 Long-term perspective

From a more long-term perspective, we would be interested to see if our modelling language and reduction techniques can be extended to include even more features, such as deterministic timing or non-exponential distributions.

Also, it would be useful to incorporate our techniques in a larger variety of tools. We already discussed the coupling with LTSmin, but for instance also foresee useful applications of dead variable reduction and confluence reduction in PRISM. The symbolic (BDD / MTBDD) aspect of these tools complicates matters, but in combination with our techniques may also open up even more possibilities.

Finally, whereas we now only focused on techniques based on equivalences, another interesting direction for future work is to consider abstraction. That is, instead of preserving all behaviour of the original system, only a subset is preserved. In some cases this may already be enough to show that a certain property is violated, while reducing more significantly than the type of reduction techniques discussed in this thesis.

Part V

Appendices

APPENDIX A

Proofs

"Very impressive, colleague ... but does it also work in theory?"

Fokke & Sukke

In many of the proofs in this appendix we use the fact that, given a countable set S, two distributions $\mu, \nu \in \mathrm{Distr}(S)$ and two equivalence relations R, R' over S such that $R' \supseteq R$, it holds that $\mu \equiv_R \mu'$ implies $\mu \equiv_{R'} \mu'$. This result immediately follows from Propositions 5.2.1.1 and 5.2.1.5 from [Sto02a]. We use this result in the proofs, without every time referring to these propositions again.

Many proofs of the propositions and theorems presented throughout the thesis rely on some auxiliary lemmas. These are each time presented directly before the results in which they are applied for the first time.

A.1 Proofs for Chapter 3

A.1.1 Proof of Proposition 3.32

Lemma A.1. *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L \rangle$ be an MA, and let $s \in S$, $\alpha \in A^\chi$, and $\mu \in \mathrm{Distr}(S)$. Then,*

$$s \xrightarrow{\alpha} \mu \quad \Longrightarrow \quad s \overset{\alpha}{\Longrightarrow}_R \mu$$

for any equivalence relation R over S.

Proof. Assume that $s \xrightarrow{\alpha} \mu$. To show that $s \overset{\alpha}{\Longrightarrow}_R \mu$, we need to provide a scheduler $\mathcal{S}$ such that

- $F^{\mathcal{S}}_{\mathcal{M}}(s) = \mu$
- For every maximal path

  $$s \xrightarrow{\alpha_1, \mu_1} s_1 \xrightarrow{\alpha_2, \mu_2} \dots \xrightarrow{\alpha_n, \mu_n} s_n \in \mathit{maxpaths}^{\mathcal{S}}_{\mathcal{M}}(s)$$

  it holds that $\alpha_n = \alpha$. Moreover, for every $1 \leq i < n$ we have $\alpha_i = \tau$, $(s, s_i) \in R$ and $L(s) = L(s_i)$.

Let $\mathcal{S}$ be defined by $\mathcal{S}(s) = \{(s, \alpha, \mu) \mapsto 1\}$ and $\mathcal{S}(\pi) = \mathbb{1}_\perp$ for every path $\pi \neq s$. Clearly, $F^{\mathcal{S}}_{\mathcal{M}}(s) = \mu$. Also, all maximal path in

$$\mathit{maxpaths}^{\mathcal{S}}_{\mathcal{M}}(s) = \{s \xrightarrow{\alpha, \mu} s' \mid s' \in \mathit{supp}(\mu)\}$$

indeed trivially satisfy the requirements above for any equivalence relation R over S (since $n = 1$ for all of these maximal paths). □

Lemma A.2. *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L\rangle$ be an MA, $R \subseteq S \times S$ a branching bisimulation for $\mathcal{M}$, and $R' \supseteq R$ an equivalence relation such that $L(p) = L(q)$ for every $(p, q) \in R'$. Then, for every $(s, t) \in R$ and all $\alpha \in A^{\chi}, \mu \in$ Distr(S), it holds that*

$$s \overset{\alpha}{\Longrightarrow}_{R'} \mu \implies \exists \mu' \in \mathrm{Distr}(S) \;.\; t \overset{\alpha}{\Longrightarrow}_{R'} \mu' \wedge \mu \equiv_{R'} \mu'.$$

Proof. Let $(s, t) \in R$, and assume that $s \overset{\alpha}{\Longrightarrow}_{R'} \mu$. If $\alpha = \tau$ and $\mu = \mathbb{1}_s$, then the trivial branching transition $t \overset{\tau}{\Longrightarrow}_{R'} \mathbb{1}_t$ completes the proof. Otherwise, assume that there exists a scheduler $\mathcal{S}$ such that

- $F^{\mathcal{S}}_{\mathcal{M}}(s) = \mu$
- For every maximal path

$$s \xrightarrow{\alpha_1,\mu_1} s_1 \xrightarrow{\alpha_2,\mu_2} \dots \xrightarrow{\alpha_n,\mu_n} s_n \in \mathit{maxpaths}^{\mathcal{S}}_{\mathcal{M}}(s)$$

  it holds that $\alpha_n = \alpha$. Moreover, for every $1 \leq i < n$ we have $\alpha_i = \tau$, $(s, s_i) \in R'$ and $L(s) = L(s_i)$.

As every single transition of s can be mimicked by t (due to (s, t) being in a bisimulation relation R), we can define a scheduler $\mathcal{S}'$ that mimics every choice of $\mathcal{S}$. So, assume that $\mathcal{S}(s) = \{s \xrightarrow{\alpha_1} \mu_1 \mapsto p_1, \dots, s \xrightarrow{\alpha_n} \mu_n \mapsto p_n\}$. Then, we let $\mathcal{S}'$ schedule the transitions necessary for $t \overset{\alpha_i}{\Longrightarrow}_R \mu'_i$ (with $\mu_i \equiv_R \mu'_i$) with probability p_i. That is, if for instance $t \xrightarrow{\alpha_2} \nu_1$ and $t \xrightarrow{\alpha_3} \nu_2$ should both be assigned probability 0.5 as a first step to obtaining $t \overset{\alpha_1}{\Longrightarrow}_R \mu'_1$, we let $\mathcal{S}'$ schedule them each with probability $0.5p_1$. This way, with probability p_1 the tree starting from t reaches a distribution over states that is R-equivalent to μ_1. As we can then again mimic the transitions of $\mathcal{S}$ from there, and this can continue until the end of each maximal path of $\mathcal{S}$, we obtain a scheduler $\mathcal{S}'$ for which $F^{\mathcal{S}'}_{\mathcal{M}}(t) = \mu'$ with $\mu \equiv_R \mu'$. Moreover, all states visited before the α-actions in the tree starting from t also remain in the same R' equivalence class because of the restrictions of the $\Longrightarrow_{R'}$ relation and the fact that the mimicked steps yield an R-equivalent distribution. Therefore, indeed $t \overset{\alpha}{\Longrightarrow}_{R'} \mu' \wedge \mu \equiv_R \mu'$. Since $R' \supseteq R$, this implies $\mu \equiv_{R'} \mu'$.

Figure 1.1 illustrates the proof for a simplified non-probabilistic setting. □

Proposition 3.32. *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L\rangle$ be an MA. Then, an equivalence relation $R \subseteq S \times S$ is a branching bisimulation for $\mathcal{M}$ if and only if for every $(s, t) \in R$ and all $\alpha \in A^{\chi}, \mu \in$ Distr(S), it holds that $L(s) = L(t)$ and*

$$s \overset{\alpha}{\Longrightarrow}_R \mu \implies \exists \mu' \in \mathrm{Distr}(S) \;.\; t \overset{\alpha}{\Longrightarrow}_R \mu' \wedge \mu \equiv_R \mu'$$

Proof. (if) First, assume that R is an equivalence relation satisfying the requirements of this proposition. Hence, if $(s, t) \in R$, then $L(s) = L(t)$ and every branching transition $s \overset{\alpha}{\Longrightarrow}_R \mu$ can be mimicked by t. By Lemma A.1, this immediately implies that also every transition $s \xrightarrow{\alpha} \mu$ can be mimicked. Therefore, R is a bisimulation relation.

$$\begin{tikzcd}
s \arrow[r, dash, dashed, "R"] \arrow[dr, dash, dashed, "R"] \arrow[dd, "\tau"] \arrow[dd, dash, dashed, bend right=60, "R'"'] & t \arrow[d, "\tau"] \\
 & t_1 \arrow[d, "\tau"] \\
s_1 \arrow[r, dash, dashed, "R"] \arrow[dr, dash, dashed, "R"] \arrow[dd, "\tau"] \arrow[dd, dash, dashed, bend right=60, "R'"'] & t_2 \arrow[d, "\tau"] \\
 & t_3 \arrow[d, "\tau"] \\
s_2 \arrow[r, dash, dashed, "R"] \arrow[dr, dash, dashed, "R"] \arrow[dd, "\alpha"] & t_4 \arrow[d, "\tau"] \\
 & t_5 \arrow[d, "\alpha"] \\
s_3 \arrow[r, dash, dashed, "R"] & t_6
\end{tikzcd}$$

Figure 1.1: Visualisation of the proof of Lemma A.2. Since R is an equivalence relation, $(s, t) \in R$ and $(s, t_1) \in R$ imply that $(t, t_1) \in R$, and hence $(t, t_1) \in R'$ since $R' \supseteq R$. Then, $(s, t) \in R$, $(s, s_1) \in R'$ and $(s_1, t_2) \in R$ together with $R' \supseteq R$ imply that $(t, t_2) \in R'$. This reasoning can be continued to see that $t \overset{\alpha}{\Longrightarrow}_{R'} t_6$.

(only if) If R is a bisimulation relation for $\mathcal{M}$ and $(s, t) \in R$, then $L(s) = L(t)$ by definition of bisimulation, and the implication immediately follows from Lemma A.2 (instantiating it with $R' = R$; as R is a bisimulation relation, by definition indeed $L(p) = L(q)$ for every $(p, q) \in R'$). $\square$

A.1.2 Proof of Proposition 3.33

Lemma A.3. *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L\rangle$ be an MA, and let $s \in S$, $\alpha \in A^{\chi}$, and $\mu \in \mathrm{Distr}(S)$. Also, let R, R' be two equivalence relations over S such that $R' \supseteq R$. Then*

$$s \overset{\alpha}{\Longrightarrow}_R \mu \quad \Longrightarrow \quad s \overset{\alpha}{\Longrightarrow}_{R'} \mu$$

Proof. Assume that $s \overset{\alpha}{\Longrightarrow}_R$. If $\alpha = \tau$ and $\mu = \mathbb{1}_s$, then by definition of the branching transition we immediately obtain $s \overset{\alpha}{\Longrightarrow}_{R'} \mu$ for any R'. Otherwise, there exists a scheduler $\mathcal{S}$ such that

- $F^{\mathcal{S}}_{\mathcal{M}}(s) = \mu$
- For every maximal path

$$s \xrightarrow{\alpha_1,\mu_1} s_1 \xrightarrow{\alpha_2,\mu_2} \dots \xrightarrow{\alpha_n,\mu_n} s_n \in \mathit{maxpaths}^{\mathcal{S}}_{\mathcal{M}}(s)$$

it holds that $\alpha_n = \alpha$. Moreover, for every $1 \leq i < n$ we have $\alpha_i = \tau$, $(s, s_i) \in R$ and $L(s) = L(s_i)$.

The same scheduler proofs the validity of $s \overset{\alpha}{\Longrightarrow}_{R'} \mu$. After all, the only thing that has to be checked is that $(s, s_i) \in R'$ still holds for all $1 \leq i < n$. As $(s, s_i) \in R$ is assumed and $R' \supseteq R$, this is immediate. □

Proposition 3.33. *The relation $\approx_{\mathrm{b}}$ is an equivalence relation.*

Proof. Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L\rangle$ be an MA. We show that $\approx_{\mathrm{b}}$ is an equivalence relation, by showing reflexivity, symmetry and transitivity.

Reflexivity holds, as the identity relation $\{(s, s) \mid s \in S\}$ is a branching bisimulation. This immediately follows by definition of branching bisimulation, Lemma A.1 and the fact that $\equiv_R$ is reflexive.

For symmetry, let $s, t \in S$ and assume that $s \approx_{\mathrm{b}} t$. Then, there must exist a branching bisimulation $R \subseteq S \times S$ such that $(s, t) \in R$. As every branching bisimulation is an equivalence relation, also $(t, s) \in R$, and hence $t \approx_{\mathrm{b}} s$.

For transitivity, let $(p, q) \in S$ and assume that $p \approx_{\mathrm{b}} q$ as well as $q \approx_{\mathrm{b}} r$. Then, using Proposition 3.32, there exists an equivalence relation $R_1 \subseteq S \times S$ such that $(p, q) \in R_1$, and for all $(s, t) \in R_1$ it holds that

$$s \overset{\alpha}{\Longrightarrow}_{R_1} \mu \implies \exists \mu' \in \mathrm{Distr}(S) \, . \, t \overset{\alpha}{\Longrightarrow}_{R_1} \mu' \wedge \mu \equiv_{R_1} \mu'.$$

Similarly, there exists an equivalence relation $R_2 \subseteq S \times S$ such that $(q, r) \in R_2$, and for all $(s, t) \in R_2$ it holds that

$$s \overset{\alpha}{\Longrightarrow}_{R_2} \mu \implies \exists \mu' \in \mathrm{Distr}(S) \, . \, t \overset{\alpha}{\Longrightarrow}_{R_2} \mu' \wedge \mu \equiv_{R_2} \mu'.$$

We define $R_3 = (R_2 \circ R_1) \cup (R_1 \circ R_2)$, and let R be the transitive closure of R_3. We first prove that R is an equivalence relation by showing (1) reflexivity, (2) symmetry, and (3) transitivity.

(1) As R_1 are R_2 are equivalence relations, they are reflexive; thus, for every state $s \in S$ it holds that $(s, s) \in R_1$ and $(s, s) \in R_2$. Therefore, also $(s, s) \in R_2 \circ R_1$ and thus $(s, s) \in R$.

2) First observe that when $(x, z) \in R_2 \circ R_1$, then there must be a state $y \in S$ such that $(x, y) \in R_1$ and $(y, z) \in R_2$, and therefore by symmetry of R_1 and R_2 also $(y, x) \in R_1$ and $(z, y) \in R_2$, and thus $(z, x) \in R_1 \circ R_2$.

Now let $(s, t) \in R$. Then there is an integer $n \geq 2$ such that there exists a sequence of states $s_1, s_2, \dots, s_n$ such that $s_1 = s$ and $s_n = t$, and for all $1 \leq i < n$ it holds that $(s_i, s_{i+1}) \in (R_2 \circ R_1)$ or $(s_i, s_{i+1}) \in (R_1 \circ R_2)$. By the observation above we can reverse the order of the states, obtaining the sequence $s_n, s_{n-1}, \dots, s_1$ such that still $s_n = t$ and $s_1 = s$, and for all $1 \leq i < n$ it holds that $(s_i, s_{i+1}) \in (R_2 \circ R_1)$ or $(s_i, s_{i+1}) \in (R_1 \circ R_2)$. To be precise, when $(s_i, s_{i+1}) \in (R_2 \circ R_1)$, then $(s_{i+1}, s_i) \in (R_1 \circ R_2)$, and when $(s_i, s_{i+1}) \in (R_1 \circ R_2)$, then $(s_{i+1}, s_i) \in (R_2 \circ R_1)$. The sequence obtained in this way proves that $(t, s) \in R$.

(3) By definition.

We now prove that $p \approx_{\mathrm{b}} r$, by showing that $(p, r) \in R$, and that for all $(s, u) \in R$

$$s \xrightarrow{\alpha} \mu \implies \exists \mu' \in \mathrm{Distr}(S) \, . \, u \overset{\alpha}{\Longrightarrow}_R \mu' \wedge \mu \equiv_R \mu'$$

As $(p, q) \in R_1$ and $(q, r) \in R_2$, it follows immediately that $(p, r) \in R_2 \circ R_1$ and therefore indeed $(p, r) \in R$. We prove the second part by induction on the number of transitive steps needed to include (s, u) in R.

Base case. Let $(s, u) \in R$ because $(s, u) \in R_2 \circ R_1$ (the case that $(s, u) \in R$ because $(s, u) \in R_1 \circ R_2$ can be proven symmetrically). This implies that there exists a state t such that $(s, t) \in R_1$ and $(t, u) \in R_2$. Let $s \xrightarrow{\alpha} \mu$. Then, by definition of bisimulation we know that there exists a $\mu' \in \mathrm{Distr}(S)$ such that $t \overset{\alpha}{\Longrightarrow}_{R_1} \mu'$ and $\mu \equiv_{R_1} \mu'$. By Lemma A.3 it follows that $t \overset{\alpha}{\Longrightarrow}_R \mu'$, and since $R \supseteq R_1$, we obtain $\mu \equiv_R \mu'$.

Note that $R \supseteq R_2$ and $L(p) = L(q)$ for every $(p, q) \in R$. The latter follows from the fact that, if $(p, q) \in R$, there is a sequence of states $s_1, s_2, \ldots, s_n$ such that $s_1 = s$ and $s_n = t$, and for all $1 \leq i < n$ it holds that $(s_i, s_{i+1}) \in (R_2 \circ R_1)$ or $(s_i, s_{i+1}) \in (R_1 \circ R_2)$. Since R_1 and R_2 only relate equally-labelled states, this immediately implies that also p and q are equally labelled. Now, Lemma A.2 applies, and hence for all $(t, u) \in R_2$ it holds that

$$t \overset{\alpha}{\Longrightarrow}_R \mu' \implies \exists \mu'' \in \mathrm{Distr}(S) \, . \, u \overset{\alpha}{\Longrightarrow}_R \mu'' \wedge \mu' \equiv_R \mu''.$$

We thus showed that $s \xrightarrow{\alpha} \mu$ implies $t \overset{\alpha}{\Longrightarrow}_R \mu'$ (with $\mu \equiv_R \mu'$), and that $t \overset{\alpha}{\Longrightarrow}_R \mu'$ implies $u \overset{\alpha}{\Longrightarrow}_R \mu''$ (with $\mu' \equiv_R \mu''$). Therefore, it follows that if $s \xrightarrow{\alpha} \mu$, indeed there exists a $\mu'' \in \mathrm{Distr}(S)$ such that $u \overset{\alpha}{\Longrightarrow}_R \mu''$. As $\equiv_R$ is an equivalence relation, $\mu \equiv_R \mu''$ follows by transitivity.

Inductive step. Assume that if $(s, t) \in R$ by k transitive steps,

$$s \xrightarrow{\alpha} \mu \implies \exists \mu' \in \mathrm{Distr}(S) \, . \, t \overset{\alpha}{\Longrightarrow}_R \mu' \wedge \mu \equiv_R \mu'.$$

Now, let $(s, u) \in R$ by $k + 1$ transitive steps. That is, there exists a t such that $(s, t) \in R$ by means of k transitive steps, and either $(t, u) \in R_2 \circ R_1$ or $(t, u) \in R_1 \circ R_2$. We then need to show that

$$s \xrightarrow{\alpha} \mu \implies \exists \mu'' \in \mathrm{Distr}(S) \, . \, u \overset{\alpha}{\Longrightarrow}_R \mu'' \wedge \mu \equiv_R \mu''.$$

By the induction hypothesis we already know that $s \xrightarrow{\alpha} \mu$ implies $t \overset{\alpha}{\Longrightarrow}_R \mu'$ for some $\mu' \equiv_R \mu$. Moreover, using the same reasoning as for the base case, we know that $t \overset{a}{\Longrightarrow}_R \mu'$ implies that $u \overset{a}{\Longrightarrow}_R \mu''$ for some $\mu'' \equiv_R \mu$. Therefore, by transitivity of $\equiv_R$ the statement holds. □

A.2 Proofs for Chapter 4

A.2.1 Proof of Proposition 4.18

Proposition 4.18. *The target μ of every transition derived using the SOS-rule* PSum *is a probability distribution over closed process terms.*

Proof. For μ to be a probability distribution function over closed process terms, it should hold that $\mu\colon S \to [0,1]$ such that $\sum_{s \in S} \mu(s) = 1$, where the sample space S consists of all process terms without free variables.

Note that μ is only defined to be nonzero for process terms p' that can be found by evaluating $p[\boldsymbol{x} := \boldsymbol{d}]$ for some $\boldsymbol{d} \in \boldsymbol{D}$. Since we assumed $a(\boldsymbol{t}) \boldsymbol{\sum}_{\boldsymbol{x}:\boldsymbol{D}} f : p$ to be a closed process term, the only variables in p that are allowed to be free are variables from $\boldsymbol{x}$. Hence, every process term $p[\boldsymbol{x} := \boldsymbol{d}]$ is again closed. Let $P = \{p[\boldsymbol{x} := \boldsymbol{d}] \mid \boldsymbol{d} \in \boldsymbol{D}\}$ be the set of these process terms. Now, indeed,

$$\begin{aligned}
\sum_{p' \in P} \mu(p') &= \sum_{p' \in P} \sum_{\substack{\boldsymbol{d}' \in \boldsymbol{D} \\ p' = p[\boldsymbol{x} := \boldsymbol{d}']}} f[\boldsymbol{x} := \boldsymbol{d}'] \\
&= \sum_{\boldsymbol{d}' \in \boldsymbol{D}} \sum_{\substack{p' \in P \\ p' = p[\boldsymbol{x} := \boldsymbol{d}']}} f[\boldsymbol{x} := \boldsymbol{d}'] \\
&= \sum_{\boldsymbol{d}' \in \boldsymbol{D}} f[\boldsymbol{x} := \boldsymbol{d}'] = 1
\end{aligned}$$

In the first step we apply the definition of μ from Figure 4.5; in the second we interchange the summand indices (which is allowed because $f[\boldsymbol{x} := \boldsymbol{d}']$ is always non-negative); in the third we omit the second summation as for every $\boldsymbol{d}' \in \boldsymbol{D}$ there is exactly one $p' \in P$ satisfying $p' = p[\boldsymbol{x} := \boldsymbol{d}']$; in the fourth we use the fact that f is a real-valued expression yielding values in $[0,1]$ such that $\sum_{\boldsymbol{d} \in \boldsymbol{D}} f[\boldsymbol{x} := \boldsymbol{d}] = 1$ (due to Definitions 4.10 and 4.16 on syntax and well-formedness). □

A.2.2 Proof of Theorem 4.35

Lemma A.4. *Let S be any set and R, R' two equivalence relations over $S \times S$ such that $R \subseteq R'$. Let $[r]_{R'} \in S/R'$ be an arbitrary equivalence class of R'. Then, $[r]_{R'} \in S/R'$ can be partitioned into equivalence classes of R.*

Proof. Let $[p]_R \in S/R$ be one of the equivalence classes of R. We show that either $[p]_R \subseteq [r]_{R'}$ or $[p]_R \cap [r]_{R'} = \varnothing$. To see this, let $s \in [p]_R$, so $(s,p) \in R$. Since $R \subseteq R'$, this implies $(s,p) \in R'$. Now, we make a case distinction based on whether or not $p \in [r]_{R'}$.

- Let $p \in [r]_{R'}$, and hence, $(p,r) \in R'$. Since $(s,p) \in R'$, by transitivity we obtain $(s,r) \in R'$ and thus $s \in [r]_{R'}$.
- Let $p \notin [r]_{R'}$. If $s \in [r]_{R'}$, then $(s,r) \in R'$ and hence by transitivity also $(p,r) \in R'$ and thus $p \in [r]_{R'}$. As this is a contradiction, $s \notin [r]_{R'}$.

Hence, if $p \in [r]_{R'}$ then every element of $[p]_R$ is in $[r]_{R'}$ and thus $[p]_R \subseteq [r]_{R'}$, and if $p \notin [r]_{R'}$ then no element of $[p]_R$ is in $[r]_{R'}$ and thus $[p]_R \cap [r]_{R'} = \varnothing$. Since every equivalence class of R is either fully contained in $[r]_{R'}$ or does not overlap with it at all, $[r]_{R'}$ indeed can be partitioned into equivalence classes of R. □

We now show that derivation-preserving bisimulation is a congruence for all prCRL operators. We allow prCRL process terms to contain free variables. In that case, we require the bisimulation relation to be valid under all possible valuations for these variables. For instance, $d > 5 \Rightarrow p$ and $d > 5 \wedge d > 3 \Rightarrow p$ are clearly bisimilar for every value of d, but $d > 5 \Rightarrow p$ and $d > 3 \Rightarrow p$ are not bisimilar if for example d is substituted by 4.

Theorem 4.35. *Derivation-preserving bisimulation is a congruence for all operators in prCRL.*

Proof. Let $p, p', q,$ and q' be prCRL process terms (possibly containing unbound data variables) such that $p \sim_{\mathrm{dp}} p'$ and $q \sim_{\mathrm{dp}} q'$ for every valuation of their free variables. We show that, for every such valuation and every $\boldsymbol{D}$, c, a, $\boldsymbol{t}$ and f, also

$$p + q \quad \sim_{\mathrm{dp}} \quad p' + q' \tag{A.1}$$

$$\sum_{\boldsymbol{x}:\boldsymbol{D}} p \quad \sim_{\mathrm{dp}} \quad \sum_{\boldsymbol{x}:\boldsymbol{D}} p' \tag{A.2}$$

$$c \Rightarrow p \quad \sim_{\mathrm{dp}} \quad c \Rightarrow p' \tag{A.3}$$

$$a(\boldsymbol{t}) \sum_{\boldsymbol{x}:\boldsymbol{D}}\!\!\bullet\, f : p \quad \sim_{\mathrm{dp}} \quad a(\boldsymbol{t}) \sum_{\boldsymbol{x}:\boldsymbol{D}}\!\!\bullet\, f : p' \tag{A.4}$$

$$Y(\boldsymbol{t}) \quad \sim_{\mathrm{dp}} \quad Y'(\boldsymbol{t}) \tag{A.5}$$

where $Y(\boldsymbol{g} : \boldsymbol{G}) = p$ and $Y'(\boldsymbol{g} : \boldsymbol{G}) = p'$ (assuming that no non-decodable construct is introduced).

Let R_p and R_q be the derivation-preserving bisimulation relations relating p and p', and q and q', respectively. Also, assume some arbitrary valuation for all free variables of p, p', q and q'.

For each of the statements above, we construct a relation R and show that it is a derivation-preserving bisimulation relation. In each case, we first prove that (a) R is a bisimulation relation relating the left-hand side and right-hand side of the equation, and then that (b) it is derivation preserving.

(A.1). We choose R to be the symmetric, reflexive, transitive closure of the set

$$R_p \cup R_q \cup \{(p + q, p' + q')\}$$

(a) Let $p + q \xrightarrow{\alpha} \mu$. We show that $p' + q' \xrightarrow{\alpha} \mu'$ such that $\mu \equiv_R \mu'$. By the operational semantics, either $p \xrightarrow{\alpha} \mu$ or $q \xrightarrow{\alpha} \mu$. We assume the first possibility without loss of generality. Since $p \sim_{\mathrm{dp}} p'$ (by the bisimulation relation R_p), we know that $p' \xrightarrow{\alpha} \mu'$ for some μ' such that $\mu \equiv_{R_p} \mu'$, and therefore, by the operational semantics, also $p' + q' \xrightarrow{\alpha} \mu'$. Since $R_p \subseteq R$, by Proposition 5.2.1 of [Sto02a] we obtain that $\mu \equiv_R \mu'$. The fact that transitions of $p' + q'$ can be mimicked by $p + q$ follows by symmetry.

For any other element $(s,t) \in R$, the required implications follow from the assumption that R_p and R_q are bisimulation relations. Since R is the smallest set containing R_p, R_q and $(p+q, p'+q')$ such that $(s,s) \in R$, $(s,t) \in R \implies (t,s) \in R$ and $(s,t) \in R \wedge (t,u) \in R \implies (s,u) \in R$, we can do induction over the number of applications of these closure rules for (s,t) to be in R. The base case, $(s,t) \in R_p$ or $(s,t) \in R_q$, follows immediate from the fact that R_p and R_q are bisimulation relations and Proposition 5.2.1 of [Sto02a]. Otherwise, $(s,t) \in R$ is due to reflexivity, symmetry or transitivity. For reflexivity, $s = t$, and they trivially mimic each other. For symmetry, $(t,s) \in R$ can mimic each other by the induction hypothesis, and therefore (s,t) also satisfy the requirements because of symmetry of mimicking. If $(s,t) \in R$ since $(s,u) \in R$ and $(u,t) \in R$, then by the induction hypothesis any transition $s \xrightarrow{\alpha} \mu$ can be mimicked by a transition $u \xrightarrow{\alpha} \mu'$ such that $\mu \equiv_R \mu'$, which in turn can be mimicked by a transition $t \xrightarrow{\alpha} \mu''$ such that $\mu' \equiv_R \mu''$. By transitivity of $\equiv_R$, indeed $\mu \equiv_R \mu''$.

(b) Let $[r]_R$ be any equivalence class of R, and λ an arbitrary rate. Also, let

$$X = \{\mathcal{D} \in \Delta \mid \exists r' \in [r]_R \,.\, p \ + q \ \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mathbb{1}_{r'}\}$$
$$X' = \{\mathcal{D} \in \Delta \mid \exists r' \in [r]_R \,.\, p' + q' \ \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mathbb{1}_{r'}\}$$

be the sets of all derivations from $p + q$ and $p' + q'$, respectively, with action $\mathsf{rate}(\lambda)$ to a state in $[r]_R$. We need to show that $|X| = |X'|$. Note that $|X| < \infty$ and $|X'| < \infty$ since infinite outgoing rates are prohibited.

Note that X can be partitioned into two sets: $X = X_p \cup X_q$, where X_p contains all derivations that start with NChoiceL (and hence correspond to transitions of p), and X_q contains all derivations that start with NChoiceR (corresponding to transitions of q). That is:

$$X_p = \{\mathcal{D} \in \Delta \mid \exists \mathcal{D}' \in \Delta \,.\, \mathcal{D} = \text{NC{\scriptsize HOICE}L} + \mathcal{D}' \wedge \exists r' \in [r]_R \,.\, p \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}'} \mathbb{1}_{r'}\}$$
$$X_q = \{\mathcal{D} \in \Delta \mid \exists \mathcal{D}' \in \Delta \,.\, \mathcal{D} = \text{NC{\scriptsize HOICE}R} + \mathcal{D}' \wedge \exists r' \in [r]_R \,.\, q \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}'} \mathbb{1}_{r'}\}$$

Similarly, we can partition X' into two sets X'_p and X'_q. Since every derivation in X_p corresponds to exactly one derivation of p, it follows that the size of X_p is given by

$$|X_p| = |\{\mathcal{D} \in \Delta \mid \exists r' \in [r]_R \,.\, p \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mathbb{1}_{r'}\}|$$

and similarly for X_q, X'_p and X'_q.

Since $R_p \subseteq R$, by Lemma A.4 we know that $[r]_R$ can be partitioned into $[p_1]_{R_p}, [p_2]_{R_p}, \ldots, [p_n]_{R_p}$ for some $p_1, p_2, \ldots p_n$. Therefore:

$$|X_p| = \sum_{i=1}^{n} |\{\mathcal{D} \in \Delta \mid \exists r' \in [p_i]_{R_p} \,.\, p \ \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mathbb{1}_{r'}\}|$$

$$= \sum_{i=1}^{n} |\{\mathcal{D} \in \Delta \mid \exists r' \in [p_i]_{R_p} \,.\, p' \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mathbb{1}_{r'}\}| = |X'_p|$$

where the second equality is due to the fact that $(p, p') \in R_p$ and R_p is derivation preserving. In the same way, we find that $|X_q| = |X'_q|$, and hence $|X| = |X'|$.

The fact that all other elements of R satisfy the derivation preservation property follows from an easy inductive argument and the fact that R_p and R_q are derivation preserving (in the same way as above for (a)).

(A.2). We choose R to be the symmetric, reflexive, and transitive closure of the set

$$R_p \cup \left\{ \left(\sum_{\boldsymbol{x}:\boldsymbol{D}} p, \sum_{\boldsymbol{x}:\boldsymbol{D}} p' \right) \right\}$$

(a) Let $\sum_{\boldsymbol{x}:\boldsymbol{D}} p \xrightarrow{\alpha} \mu$. Then, by the operational semantics, there is a $\boldsymbol{d} \in \boldsymbol{D}$ such that $p[\boldsymbol{x} := \boldsymbol{d}] \xrightarrow{\alpha} \mu$. From the assumption that $p \sim_{\text{dp}} p'$ for all valuations, it immediately follows that $p[\boldsymbol{x} := \boldsymbol{d}] \sim_{\text{dp}} p'[\boldsymbol{x} := \boldsymbol{d}]$ for any $\boldsymbol{d} \in \boldsymbol{D}$, so if we have $p[\boldsymbol{x} := \boldsymbol{d}] \xrightarrow{\alpha} \mu$, then also $p'[\boldsymbol{x} := \boldsymbol{d}] \xrightarrow{\alpha} \mu'$ and hence $\sum_{\boldsymbol{x}:\boldsymbol{D}} p' \xrightarrow{\alpha} \mu'$ with $\mu \equiv_{R_p} \mu'$ and thus $\mu \equiv_R \mu'$ due to $R \supseteq R_p$ and Proposition 5.2.1 of [Sto02a]. The fact that transitions of $\sum_{\boldsymbol{x}:\boldsymbol{D}} p'$ can be mimicked by $\sum_{\boldsymbol{x}:\boldsymbol{D}} p$ follows by symmetry. For all other elements of R, the required implications follow from the assumption that R_p is a bisimulation relation as above.

(b) Let $[r]_R$ be any equivalence class of R, and λ an arbitrary rate. Also, let

$$X = \{\mathcal{D} \in \Delta \mid \exists r' \in [r]_R \,.\, \sum_{\boldsymbol{x}:\boldsymbol{D}} p \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mathbb{1}_{r'}\}$$
$$X' = \{\mathcal{D} \in \Delta \mid \exists r' \in [r]_R \,.\, \sum_{\boldsymbol{x}:\boldsymbol{D}} p \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mathbb{1}_{r'}\}$$

be the sets of all derivations from $\sum_{\boldsymbol{x}:\boldsymbol{D}} p$ and $\sum_{\boldsymbol{x}:\boldsymbol{D}} p'$, respectively, with action $\mathsf{rate}(\lambda)$ to a state in $[r]_R$. We need to show that $|X| = |X'|$. Again, neither X nor X' can be infinite.

Note that X can be partitioned into as many sets as there are elements in the set $\boldsymbol{D}$: $X = \bigcup_{\boldsymbol{d} \in \boldsymbol{D}} X_{\boldsymbol{d}}$, where $X_{\boldsymbol{d}}$ contains all derivations that start with $\text{NSUM}(\boldsymbol{d})$ (and hence correspond to transitions of p with $\boldsymbol{d}$ substituted for $\boldsymbol{x}$). That is:

$$X_{\boldsymbol{d}} = \{\mathcal{D} \in \Delta \mid \exists \mathcal{D}' \in \Delta \,.\, \mathcal{D} = \text{NSUM}(\boldsymbol{d}) + \mathcal{D}' \wedge$$
$$\exists r' \in [r]_R \,.\, p[\boldsymbol{x} := \boldsymbol{d}] \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}'} \mathbb{1}_{r'}\}$$

Similarly, we can partition X' into sets $X'_{\boldsymbol{d}}$. Since every derivation in $X_{\boldsymbol{d}}$ corresponds precisely to one derivation of $p[\boldsymbol{x} := \boldsymbol{d}]$, it follows that the size of $X_{\boldsymbol{d}}$ is given by

$$|X_{\boldsymbol{d}}| = |\{\mathcal{D} \in \Delta \mid \exists r' \in [r]_R \,.\, p[\boldsymbol{x} := \boldsymbol{d}] \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mathbb{1}_{r'}\}|$$

and similarly for $X'_{\boldsymbol{d}}$.

Since $R_p \subseteq R$, by Lemma A.4 we know that $[r]_R$ can be partitioned into $[p_1]_{R_p}, [p_2]_{R_p}, \ldots, [p_n]_{R_p}$ for some $p_1, p_2, \ldots p_n$. Therefore:

$$\begin{aligned}|X_{\boldsymbol{d}}| &= \sum_{i=1}^{n} |\{\mathcal{D} \in \Delta \mid \exists r' \in [p_i]_{R_p} \,.\, p\,[\boldsymbol{x} := \boldsymbol{d}] \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mathbb{1}_{r'}\}| \\ &= \sum_{i=1}^{n} |\{\mathcal{D} \in \Delta \mid \exists r' \in [p_i]_{R_p} \,.\, p'[\boldsymbol{x} := \boldsymbol{d}] \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mathbb{1}_{r'}\}| = |X'_{\boldsymbol{d}}|\end{aligned}$$

where the second equality is due to the fact that $(p, p') \in R_p$ and R_p is derivation preserving for every valuation. As this holds for all $X_{\boldsymbol{d}}$, we obtain $|X| = |X'|$.

The fact that all other elements of R satisfy the derivation preservation property follows again from the fact that R_p is derivation preserving.

(A.3). We choose R to be the symmetric, reflexive, and transitive closure of the set

$$R_p \cup \{(c \Rightarrow p, c \Rightarrow p')\}$$

(a) Let $(c \Rightarrow p) \xrightarrow{\alpha} \mu$. By the operational semantics, this implies that c holds for the given valuation and $p \xrightarrow{\alpha} \mu$. Now, since $p \sim_{\mathrm{dp}} p'$ (by the bisimulation relation R_p), we know that $p' \xrightarrow{\alpha} \mu'$, and therefore also $(c \Rightarrow p') \xrightarrow{\alpha} \mu'$, such that $\mu \equiv_{R_p} \mu'$. Since $R_p \subseteq R$, by Proposition 5.2.1 of [Sto02a] we obtain that $\mu \equiv_R \mu'$. The fact that transitions of $c \Rightarrow p'$ can be mimicked by $c \Rightarrow p$ follows by symmetry. For all other elements of R, the required implications follow from the assumption that R_p is a bisimulation relation, as above.

(b) If c does not hold for the given valuation, then both $c \Rightarrow p$ and $c \Rightarrow p'$ have no derivations at all. If c does hold, the proof is analogous to the proof of 1(b) and 2(b).

(A.4). We choose R to be the symmetric, reflexive, and transitive closure of the set

$$R_p \cup \left\{ \left(a(\boldsymbol{t}) \sum_{\boldsymbol{x}:\boldsymbol{D}} f : p, a(\boldsymbol{t}) \sum_{\boldsymbol{x}:\boldsymbol{D}} f : p' \right) \right\}$$

(a) Let $(a(\boldsymbol{t}) \sum_{\boldsymbol{x}:\boldsymbol{D}} f : p) \xrightarrow{\alpha} \mu$. Then, by the operational semantics $\alpha = a(\boldsymbol{t})$, and

$$\forall \boldsymbol{d} \in \boldsymbol{D} \,.\, \mu(p[\boldsymbol{x} := \boldsymbol{d}]) = \sum_{\substack{\boldsymbol{d}' \in \boldsymbol{D} \\ p[\boldsymbol{x}:=\boldsymbol{d}]=p[\boldsymbol{x}:=\boldsymbol{d}']}} f[\boldsymbol{x} := \boldsymbol{d}']$$

Then, also $(a(\boldsymbol{t}) \sum_{\boldsymbol{x}:\boldsymbol{D}} f : p') \xrightarrow{\alpha} \mu'$, where $\alpha = a(\boldsymbol{t})$ and

$$\forall \boldsymbol{d} \in \boldsymbol{D} \,.\, \mu'(p'[\boldsymbol{x} := \boldsymbol{d}]) = \sum_{\substack{\boldsymbol{d}' \in \boldsymbol{D} \\ p'[\boldsymbol{x}:=\boldsymbol{d}]=p'[\boldsymbol{x}:=\boldsymbol{d}']}} f[\boldsymbol{x} := \boldsymbol{d}']$$

From the assumption that $p \sim_{\mathrm{dp}} p'$ for all valuations (by the bisimulation relation R_p), it immediately follows that $p[\boldsymbol{x} := \boldsymbol{d}] \sim_{\mathrm{dp}} p'[\boldsymbol{x} := \boldsymbol{d}]$ for any $\boldsymbol{d} \in \boldsymbol{D}$, so $(p[\boldsymbol{x} := \boldsymbol{d}], p'[\boldsymbol{x} := \boldsymbol{d}]) \in R_p$. Since μ and μ' both assign probability $f[\boldsymbol{x} := \boldsymbol{d}]$ to these process terms, they assign equal probabilities to each equivalence class of R_p; hence, $\mu \equiv_{R_p} \mu'$ and thus $\mu \equiv_R \mu'$ due to $R \supseteq R_p$ and Proposition 5.2.1 of [Sto02a]. (Note that for instance μ might have $p[\boldsymbol{x} := \boldsymbol{d}] = p[\boldsymbol{x} := \boldsymbol{d}']$ and therefore assign probability $f[\boldsymbol{x} := \boldsymbol{d}] + f[\boldsymbol{x} := \boldsymbol{d}']$ to this term. However, even if $p'[\boldsymbol{x} := \boldsymbol{d}] \neq p'[\boldsymbol{x} := \boldsymbol{d}']$ and therefore μ' does not combine these probabilities, still all terms are in the same equivalence class, and therefore everything still matches.)

Again, the mimicking the other way around follows by symmetry. For all other elements of R, the required implications follow from the assumption that R_p is a bisimulation relation, as above.

(b) The proof is analogous to the proof of 1(b) and 2(b).

(A.5). We choose R to be the symmetric, reflexive, transitive closure of the set

$$R_p \cup \{(Y(\boldsymbol{t}), Y'(\boldsymbol{t})\}$$

Recall that $Y(\boldsymbol{g} : \boldsymbol{G}) = p$, $Y'(\boldsymbol{g} : \boldsymbol{G}) = p'$ and $p \sim_{\mathrm{dp}} p'$ for all valuations.

(a) Let $Y(\boldsymbol{t}) \xrightarrow{\alpha} \mu$. Then, by the operational semantics, also $p[\boldsymbol{x} := \boldsymbol{t}] \xrightarrow{\alpha} \mu$. From the assumption that $p \sim_{\mathrm{dp}} p'$ for all valuations, it immediately follows that $p[\boldsymbol{x} := \boldsymbol{t}] \sim_{\mathrm{dp}} p'[\boldsymbol{x} := \boldsymbol{t}]$. Therefore, also $p'[\boldsymbol{x} := \boldsymbol{t}] \xrightarrow{\alpha} \mu'$ with $\mu \equiv_{R_p} \mu'$ and thus $\mu \equiv_R \mu'$ due to $R \supseteq R_p$ and Proposition 5.2.1 of [Sto02a]. The fact that transitions of $Y'(\boldsymbol{t})$ can be mimicked by $Y(\boldsymbol{t})$ follows by symmetry. For all other elements of R, the required implications follow from the assumption that R_p is a bisimulation relation.

(b) The proof is analogous to the proof of 1(b) and 2(b). □

A.2.3 Proof of Theorem 4.36

The following lemma states that, if $\mu \equiv_R \mu'$, then also $\mu_f \equiv_{R_f} \mu'_f$, where R_f is the lifting of R over a bijective function f.

Lemma A.5. *Let S, T be countable sets, $\mu, \mu' \in \mathrm{Distr}(S)$, and $R \subseteq S \times S$ an equivalence relation such that $\mu \equiv_R \mu'$. Given a bijective function $f \colon S \to T$, the set*

$$R_f = \{(t, t') \in T^2 \mid (f^{-1}(t)), f^{-1}(t')) \in R\}$$

is an equivalence relation and $\mu_f \equiv_{R_f} \mu'_f$.

Proof. For any $t \in T$, we have $(t, t) \in R_f$ since $(f^{-1}(t), f^{-1}(t)) \in R$ due to reflexivity of R; hence, R_f is also reflexive. For any $(t, t') \in R_f$ it holds that $(f^{-1}(t), f^{-1}(t')) \in R$, so by symmetry of R also $(f^{-1}(t'), f^{-1}(t)) \in R$ and hence $(t', t) \in R_f$. Therefore, R_f is also symmetric. For any $(t, t') \in R_f$ and $(t', t'') \in R_f$, we find $(f^{-1}(t), f^{-1}(t')) \in R$ and $(f^{-1}(t'), f^{-1}(t'')) \in R$, so $(f^{-1}(t), f^{-1}(t'')) \in R$ by transitivity of R, and hence also $(t, t'') \in R_f$. Therefore, R_f is also transitive.

Now, let $[t]_{R_f}$ be an arbitrary equivalence class of R_f, then

$$
\begin{aligned}
&\mu_f([t]_{R_f}) && \{ \text{ Def. of probability of sets } \}\\
&= \sum_{t' \in [t]_{R_f}} \mu_f(t') && \{ \text{ Def. of lifting of distributions } \}\\
&= \sum_{t' \in [t]_{R_f}} \mu(f^{-1}(t')) && \{ \text{ Disjointness of inverses } \}\\
&= \mu(\bigcup_{t' \in [t]_{R_f}} \{f^{-1}(t')\}) && \{ \text{ Def. of inverse } \}\\
&= \mu(\bigcup_{t' \in [t]_{R_f}} \{s \in S \mid f(s) = t'\}) && \{ \text{ Easy rewriting } \}\\
&= \mu(\{s \in S \mid f(s) \in [t]_{R_f}\}) && \{ \text{ See below } \}\\
&= \sum_{\substack{[s]_R \in S/R\\ f(s) \in [t]_{R_f}}} \mu([s]_R)
\end{aligned}
$$

To see why the final equality holds, we show that $f(s) \in [t]_{R_f}$ if and only if $f(s') \in [t]_{R_f}$ for every $s' \in [s]_R$ (note that the 'if' part of this statement is trivial, since $s \in [s]_R$). Then, the total probability of all states s such that $f(s) \in [t]_{R_f}$ clearly corresponds to the total probability of all classes of states for which at least one state has this property.

Let $s \in S$ such that $f(s) \in [t]_{R_f}$, and let $s' \in [s]_R$. So, by definition of equivalence classes, $(s, s') \in R$. Hence, by definition of R_f also $(f(s), f(s')) \in R_f$. Since $f(s) \in [t]_{R_f}$, therefore by definition of equivalence classes $(f(s), t) \in R_f$. Finally, by symmetry and transitivity of R_f we obtain $(f(s'), t) \in R_f$ and thus $f(s') \in [t]_{R_f}$.

In exactly the same way as above, we can show that

$$
\mu'_f([t]_{R_f}) = \sum_{\substack{[s]_R \in S/R\\ f(s) \in [t]_{R_f}}} \mu'([s]_R)
$$

Now, since $\mu([s]_R) = \mu'([s]_R)$ for every $s \in S$ (by definition of $\equiv$ and due to the assumption $\mu \equiv_R \mu'$), we obtain $\mu_f([t]_{R_f}) = \mu'_f([t]_{R_f})$ and hence $\mu_f \equiv_{R_f} \mu'_f$. $\square$

Based on the encoding and decoding rules, we can prove the following results. Note that the Lemma A.6 implies that dec and enc are bijective (when restricting to decodable prCRL specifications).

Lemma A.6. *Restricting to MAPA specifications without any* rate*-actions and decodable prCRL specifications, the functions* dec *and* enc *are each others' inverse. That is,*

$$
\mathsf{dec} \circ \mathsf{enc} = \mathsf{id_m} \qquad \textit{and} \qquad \mathsf{enc} \circ \mathsf{dec} = \mathsf{id_p}
$$

where $\mathsf{id_m}$ *is the identity function on MAPA process terms and* $\mathsf{id_p}$ *is the identity function on prCRL process terms.*

Proof. We show that $\mathsf{dec}\,(\mathsf{enc}\,(p)) = p$ for every MAPA process term p, by induction on the structure of p. It can be shown similarly that $\mathsf{enc}\,(\mathsf{dec}\,(p)) = p$ for every prCRL term p (using the assumption that p is decodable).

Base case. Let $p = Y(\boldsymbol{t})$. Then,

$$\mathsf{dec}\,(\mathsf{enc}\,(p)) = \mathsf{dec}\,(\mathsf{enc}\,(Y(\boldsymbol{t}))) = \mathsf{dec}\,(Y(\boldsymbol{t})) = Y(\boldsymbol{t}) = p$$

Inductive case. Let $\mathsf{dec}\,(\mathsf{enc}\,(p)) = p$ and $\mathsf{dec}\,(\mathsf{enc}\,(q)) = q$. Now:

$$\begin{aligned}
&\mathsf{dec}\,(\mathsf{enc}\,(c \Rightarrow p)) && \{\text{ Def. of } \mathsf{enc}\,() \}\\
&= \mathsf{dec}\,(c \Rightarrow \mathsf{enc}\,(p)) && \{\text{ Def. of } \mathsf{dec}\,() \}\\
&= c \Rightarrow \mathsf{dec}\,(\mathsf{enc}\,(p)) && \{\text{ Induction hypothesis } \}\\
&= c \Rightarrow p
\end{aligned}$$

We can show in exactly the same way that

$$\begin{aligned}
\mathsf{dec}\,(\mathsf{enc}\,(p + q)) &= p + q\\
\mathsf{dec}\,(\mathsf{enc}\,(\textstyle\sum_{\boldsymbol{x}:\boldsymbol{D}} p)) &= \textstyle\sum_{\boldsymbol{x}:\boldsymbol{D}} p\\
\mathsf{dec}\,(\mathsf{enc}\,(a(\boldsymbol{t})\textstyle\sum\!\!\!\!\bullet_{\boldsymbol{x}:\boldsymbol{D}} f : p)) &= a(\boldsymbol{t})\textstyle\sum\!\!\!\!\bullet_{\boldsymbol{x}:\boldsymbol{D}} f : p
\end{aligned}$$

where for the last equation, we need the assumption that $a \neq \mathsf{rate}$. Finally,

$$\begin{aligned}
\mathsf{dec}\,(\mathsf{enc}\,((\lambda) \cdot p)) &= \mathsf{dec}\,(\mathsf{rate}(\lambda) \sum\!\!\!\!\!\bullet_{x:\{*\}} 1 : \mathsf{enc}\,(p))\\
&= (\lambda) \cdot \mathsf{dec}\,(\mathsf{enc}\,(p)) = (\lambda) \cdot p
\end{aligned}$$ □

The following lemma states that enc is similar to a functional bisimulation (i.e., a bisimulation relation that is actually a function), except that it relates MAPA process terms to prCRL process terms.

Lemma A.7. *Let m be a MAPA process term. Then, for every action $a \neq \mathsf{rate}$ and distribution μ,*

$$m \overset{a}{\hookrightarrow} \mu \quad \Longleftrightarrow \quad \mathsf{enc}\,(m) \xrightarrow{a} \mu_{\mathsf{enc}}$$

Proof. Let $m \overset{a}{\hookrightarrow} \mu$. We prove that $\mathsf{enc}\,(m) \xrightarrow{a} \mu_{\mathsf{enc}}$ by induction on the structure of m. The reverse can be proven symmetrically, noting that dec indeed decodes a transition like $\mathsf{enc}\,(m) \xrightarrow{a} \mu_{\mathsf{enc}}$ to an interactive transition if $a \neq \mathsf{rate}$.

Base case. Let $m = b(\boldsymbol{t})\sum_{\boldsymbol{x}:\boldsymbol{D}} f : m'$. Since $m \xhookrightarrow{a} \mu$, by the SOS rules it must hold that $a = b(\boldsymbol{t})$ and

$$\forall \boldsymbol{d} \in \boldsymbol{D} \,.\, \mu(m'[\boldsymbol{x} := \boldsymbol{d}]) = \sum_{\substack{\boldsymbol{d}' \in \boldsymbol{D} \\ m'[\boldsymbol{x}:=\boldsymbol{d}]=m'[\boldsymbol{x}:=\boldsymbol{d}']}} f[\boldsymbol{x} := \boldsymbol{d}']$$

Now, by definition of enc we have $\mathsf{enc}\,(m) = b(\boldsymbol{t})\sum_{\boldsymbol{x}:\boldsymbol{D}} f : \mathsf{enc}\,(m')$. Hence, by the SOS rules for prCRL it holds that $\mathsf{enc}\,(m) \xrightarrow{a} \mu'$, where

$$\forall \boldsymbol{d} \in \boldsymbol{D} \,.\, \mu'(\mathsf{enc}\,(m')\,[\boldsymbol{x} := \boldsymbol{d}]) = \sum_{\substack{\boldsymbol{d}' \in \boldsymbol{D} \\ \mathsf{enc}(m')[\boldsymbol{x}:=\boldsymbol{d}]=\mathsf{enc}(m')[\boldsymbol{x}:=\boldsymbol{d}']}} f[\boldsymbol{x} := \boldsymbol{d}']$$

Since the enc function does neither introduce nor remove variables, it follows that, for every $\boldsymbol{d}' \in \boldsymbol{D}$, $\mathsf{enc}\,(m')\,[\boldsymbol{x} := \boldsymbol{d}] = \mathsf{enc}\,(m')\,[\boldsymbol{x} := \boldsymbol{d}']$ holds if and only if $m'[\boldsymbol{x} := \boldsymbol{d}] = m'[\boldsymbol{x} := \boldsymbol{d}']$ holds. Hence, the right-hand sides of the two equations coincide. Also, note that $\mathsf{enc}\,(m')\,[\boldsymbol{x} := \boldsymbol{d}] = \mathsf{enc}\,(m'[\boldsymbol{x} := \boldsymbol{d}])$. Therefore $\mu'(\mathsf{enc}\,(m'')) = \mu(m'')$ for every MAPA process term m''. By definition, this implies that $\mu' = \mu_{\mathsf{enc}}$.

Inductive case. Let $m = m' + m''$. Since $m \xhookrightarrow{a} \mu$, by the SOS rules it must hold that either $m' \xhookrightarrow{a} \mu$ or $m'' \xhookrightarrow{a} \mu$. By induction, this implies that either $\mathsf{enc}\,(m') \xrightarrow{a} \mu_{\mathsf{enc}}$ or $\mathsf{enc}\,(m'') \xrightarrow{a} \mu_{\mathsf{enc}}$. Since $\mathsf{enc}\,(m) = \mathsf{enc}\,(m') + \mathsf{enc}\,(m'')$, the SOS rules for prCRL imply that $\mathsf{enc}\,(m) \xrightarrow{a} \mu_{\mathsf{enc}}$.

The cases where $m = Y(\boldsymbol{t})$, $m = c \Rightarrow m'$ or $m = \sum_{\boldsymbol{x}:\boldsymbol{D}} m'$ are proven in the same way. Note that $m \neq (\lambda) \cdot m'$, since such a process term cannot do an interactive transition. □

Lemma A.8. *Let m be a MAPA process term. Then, for every process term m', rate λ and Markovian derivation $\mathcal{D}$,*

$$m \xrightarrow{\lambda}_{\mathcal{D}} m' \quad \Longleftrightarrow \quad \mathsf{enc}\,(m) \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}'} \mathbb{1}_{\mathsf{enc}(m')}$$

where $\mathcal{D}'$ is obtained from $\mathcal{D}$ by substituting PSum *for* MStep.

Proof. Let $m \xrightarrow{\lambda}_{\mathcal{D}} m'$. We prove that $\mathsf{enc}\,(m) \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}'} \mathbb{1}_{\mathsf{enc}(m')}$, by induction on the structure of m. The reverse can be proven symmetrically.

Base case. Let $m = (\kappa) \cdot m'$. Since $m \xrightarrow{\lambda}_{\mathcal{D}} m'$, by the SOS rules it must hold that $\kappa = \lambda$ and $\mathcal{D} = \langle \text{MStep} \rangle$. Hence, $\mathsf{enc}\,(m) = \mathsf{rate}(\lambda)\sum_{x:\{*\}} 1 : \mathsf{enc}\,(m')$.

The derivation $\mathcal{D}'$, corresponding to $\mathcal{D}$, is $\langle \text{PSum} \rangle$. Note that, by the SOS rules of prCRL and the fact that x does not occur in $\mathsf{enc}\,(m')$ by definition of enc, indeed $\mathsf{enc}\,(m) \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}'} \mathbb{1}_{\mathsf{enc}(m')}$.

Inductive case. Let $m = m_1 + m_2$. Since $m \xrightarrow{\lambda}_{\mathcal{D}} m'$, by the SOS rules it must hold that either $m_1 \xrightarrow{\lambda}_{\mathcal{D}_1} m'$ and $\mathcal{D} = \langle \text{NChoiceL} \rangle + \mathcal{D}_1$ or $m_2 \xrightarrow{\lambda}_{\mathcal{D}_2} m'$

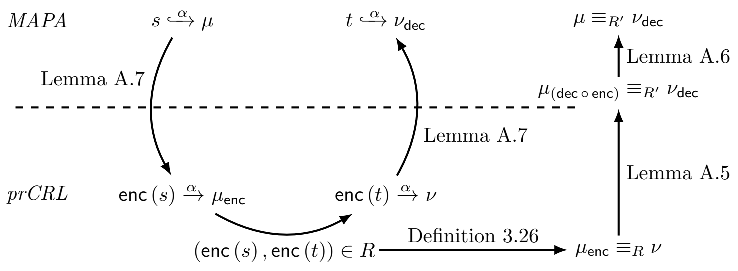

MAPA
$s \xhookrightarrow{\alpha} \mu$
$t \xhookrightarrow{\alpha} \nu_{\mathsf{dec}}$
$\mu \equiv_{R'} \nu_{\mathsf{dec}}$
Lemma A.6
Lemma A.7
$\mu_{(\mathsf{dec} \circ \mathsf{enc})} \equiv_{R'} \nu_{\mathsf{dec}}$
Lemma A.7
Lemma A.5
prCRL
$\mathsf{enc}(s) \xrightarrow{\alpha} \mu_{\mathsf{enc}}$
$\mathsf{enc}(t) \xrightarrow{\alpha} \nu$
Definition 3.26
$(\mathsf{enc}(s), \mathsf{enc}(t)) \in R$
$\mu_{\mathsf{enc}} \equiv_R \nu$

Figure 1.2: Visualisation of the proof of Lemma A.9 part (1).

and $\mathcal{D} = \langle \text{NChoiceR} \rangle + \mathcal{D}_2$. Assume the first (the proof for the other option is symmetrical). By induction, this implies that $\mathsf{enc}(m_1) \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}_1'} \mathbb{1}_{\mathsf{enc}(m')}$. Since $\mathsf{enc}(m) = \mathsf{enc}(m_1) + \mathsf{enc}(m_2)$, the SOS rules for prCRL imply that $\mathsf{enc}(m) \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}_1''} \mathbb{1}_{\mathsf{enc}(m')}$, where $\mathcal{D}_1'' = \langle \text{NChoiceL} \rangle + \mathcal{D}_1'$. Since we already saw that $\mathcal{D} = \langle \text{NChoiceL} \rangle + \mathcal{D}_1$, indeed $\mathcal{D}_1'' = \mathcal{D}'$.

The cases where $m = Y(\boldsymbol{t})$, $m = c \Rightarrow m'$ or $m = \sum_{\boldsymbol{x}:\boldsymbol{D}} m'$ are proven in the same way. Note that $m \neq b(\boldsymbol{t}) \sum_{\boldsymbol{x}:\boldsymbol{D}} f : m'$, since such a process term cannot do a Markovian transition. $\square$

Lemma A.9. *Let P_1, P_2 be decodable prCRL specifications. Then,*

$$P_1 \sim_{\text{dp}} P_2 \quad \Rightarrow \quad \mathsf{dec}(P_1) \approx_{\text{s}} \mathsf{dec}(P_2).$$

Proof. Assume $P_1 \sim_{\text{dp}} P_2$, and let $\mathcal{M}_1 = \langle S, s_1^0, A, \hookrightarrow, \rightsquigarrow \rangle$ and $\mathcal{M}_2 = \langle S, s_2^0, A, \hookrightarrow, \rightsquigarrow \rangle$ be the MAs that represent the semantics of $\mathsf{dec}(P_1)$ and $\mathsf{dec}(P_2)$. Let R be the derivation-preserving bisimulation relating P_1 and P_2.

Now, consider the bisimulation relation R' over MAPA terms, given by $R' = \{(\mathsf{dec}(p), \mathsf{dec}(q)) \mid (p, q) \in R\}$. It is easy to see that R' is an equivalence relation, since R is one. We now show that it is a bisimulation relation relating $\mathcal{M}_1$ and $\mathcal{M}_2$, and therefore proving the result.

First, since the initial states of P_1 and P_2 are related by R, the initial states of $\mathsf{dec}(P_1)$ and $\mathsf{dec}(P_2)$ are related by R' by definition of dec. Second, let $(s, t) \in R'$ and assume that $s \xrightarrow{\alpha} \mu$. We show that $t \xrightarrow{\alpha} \mu'$ such that $\mu \equiv_{R'} \mu'$. Note that either (1) $\alpha \in A$ and $s \xhookrightarrow{\alpha} \mu$, or (2) $\alpha = \chi(\mathit{rate}(s))$, $\mathit{rate}(s) > 0$, $\mu = \mathbb{P}_s$ and there is no μ' such that $s \xhookrightarrow{\tau} \mu'$. Also note that $\alpha \neq \mathsf{rate}$, by definition of dec.

(1) Let $s \xhookrightarrow{\alpha} \mu$ for some $\alpha \in A$ such that $\alpha \neq \mathsf{rate}$. We need to show that $t \xhookrightarrow{\alpha} \mu'$ such that $\mu \equiv_{R'} \mu'$. First note that, by Lemma A.7, we have $\mathsf{enc}(s) \xrightarrow{\alpha} \mu_{\mathsf{enc}}$. We know that $(s, t) \in R'$, so $(\mathsf{enc}(s), \mathsf{enc}(t)) \in R$. Since $\mathsf{enc}(s) \xrightarrow{\alpha} \mu_{\mathsf{enc}}$ and R is a bisimulation relation, this implies that $\mathsf{enc}(t) \xrightarrow{\alpha} \nu$ such that $\mu_{\mathsf{enc}} \equiv_R \nu$. Then, $t \xhookrightarrow{\alpha} \nu_{\mathsf{dec}}$ by Lemma A.7.

Now, note that R' can be seen as R_{dec} as defined in Lemma A.5. Hence, by this lemma $\mu_{\mathsf{enc}} \equiv_R \nu$ implies $\mu_{(\mathsf{dec} \circ \mathsf{enc})} \equiv_{R'} \nu_{\mathsf{dec}}$. By Lemma A.6, this reduces to $\mu \equiv_{R'} \nu_{\mathsf{dec}}$, which is what we wanted to show.

Figure 1.2 illustrates this part of the proof.

(2) Let $\alpha = \chi(\mathit{rate}(s))$, $\mathit{rate}(s) > 0$, $\mu = \mathbb{P}_s$ and let there be no μ' such that $s \xhookrightarrow{\tau} \mu'$. We need to show that $t \xrightarrow{\alpha} \nu$ such that $\mu \equiv_{R'} \nu$, i.e., that (a) $\mathit{rate}(t) = \mathit{rate}(s)$, that (b) $\mathbb{P}_s \equiv_{R'} \mathbb{P}_t$ and that (c) there is no μ' such that $t \xhookrightarrow{\tau} \mu'$.

For (a), note that

$$\mathit{rate}(s) = \sum_{s' \in S} \mathit{rate}(s, s') = \sum_{s' \in S} \sum_{(s,\lambda,s') \in \rightsquigarrow} \lambda$$

by Definition 3.5, and that by the operational semantics, we have $(s, \lambda, s') \in \rightsquigarrow$ if and only if $\mathsf{MD}(s, s') \neq \varnothing$ and $\lambda = \sum_{(\lambda_i,\mathcal{D}) \in \mathsf{MD}(s,s')} \lambda_i$. Combining this, we obtain

$$\mathit{rate}(s) = \sum_{s' \in S} \sum_{(\lambda_i,\mathcal{D}) \in \mathsf{MD}(s,s')} \lambda_i = \sum_{(\lambda_i,\mathcal{D}) \in \mathsf{MD}(s)} \lambda_i$$

where $\mathsf{MD}(s) = \bigcup_{s' \in S} \mathsf{MD}(s, s') = \{(\lambda_i, \mathcal{D}) \in \mathbb{R} \times \Delta \mid \exists s' \in S \,.\, s \xrightarrow{\lambda_i}_{\mathcal{D}} s'\}$. This rewriting is valid since all sets $\mathsf{MD}(s, s')$ are disjoint, because every Markovian derivation $\mathcal{D}$ yields a single target state s'.

Now, let

$$\mathsf{MD}'(\mathsf{enc}\,(s)) = \{(\lambda_i, \mathcal{D}) \in \mathbb{R} \times \Delta \mid \exists s' \in S \,.\, \mathsf{enc}\,(s) \xrightarrow{\mathsf{rate}(\lambda_i)}_{\mathcal{D}'} \mathbb{1}_{\mathsf{enc}(s')}\}$$

where $\mathcal{D}'$ is again obtained from $\mathcal{D}$ by substituting PSum for MStep. By Lemma A.8, it follows that $\mathsf{MD}(s) = \mathsf{MD}'(\mathsf{enc}\,(s))$. Hence, $\mathsf{enc}\,(s)$ has the same outgoing transitions as s, except that the derivations are slightly different and that the target states are encoded too.

Similarly, $\mathit{rate}(t) = \sum_{(\lambda_i,\mathcal{D}) \in \mathsf{MD}(t)} \lambda_i$ with $\mathsf{MD}(t) = \{(\lambda_i, \mathcal{D}) \in \mathbb{R} \times \Delta \mid \exists t' \in S \,.\, t \xrightarrow{\lambda_i}_{\mathcal{D}} t'\}$, and $\mathsf{MD}(t) = \mathsf{MD}'(\mathsf{enc}\,(t)) = \{(\lambda_i, \mathcal{D}) \in \mathbb{R} \times \Delta \mid \exists t' \in S \,.\, \mathsf{enc}\,(t) \xrightarrow{\mathsf{rate}(\lambda_i)}_{\mathcal{D}'} \mathbb{1}_{\mathsf{enc}(t')}\}$.

To show that $\mathit{rate}(s) = \mathit{rate}(t)$, it therefore remains to show that

$$\sum_{(\lambda_i,\mathcal{D}) \in \mathsf{MD}'(\mathsf{enc}(s))} \lambda_i = \sum_{(\lambda_i,\mathcal{D}) \in \mathsf{MD}'(\mathsf{enc}(t))} \lambda_i$$

To see why this is the case, first note that, since the bisimulation relation R is derivation-preserving, by definition we know that for every $(p, q) \in R$, every equivalence equivalence class $[r]_R$ and every rate λ, it holds that

$$|\{\mathcal{D} \in \Delta \mid \exists r' \in [r]_R \,.\, p \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mathbb{1}_{r'}\}| = |\{\mathcal{D} \in \Delta \mid \exists r' \in [r]_R \,.\, q \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mathbb{1}_{r'}\}|$$

Hence, as $(\mathsf{enc}\,(s), \mathsf{enc}\,(t)) \in R$, this also holds for $\mathsf{enc}\,(s), \mathsf{enc}\,(t)$, and therefore

$$|\{\mathcal{D} \in \Delta \mid \exists \mathsf{enc}\,(r') \in [\mathsf{enc}\,(r)]_R \,.\, \mathsf{enc}\,(s) \xrightarrow{\mathsf{rate}(\lambda_i)}_{\mathcal{D}} \mathbb{1}_{\mathsf{enc}(r')}\}| =$$
$$|\{\mathcal{D} \in \Delta \mid \exists \mathsf{enc}\,(r') \in [\mathsf{enc}\,(r)]_R \,.\, \mathsf{enc}\,(t) \xrightarrow{\mathsf{rate}(\lambda_i)}_{\mathcal{D}} \mathbb{1}_{\mathsf{enc}(r')}\}|$$

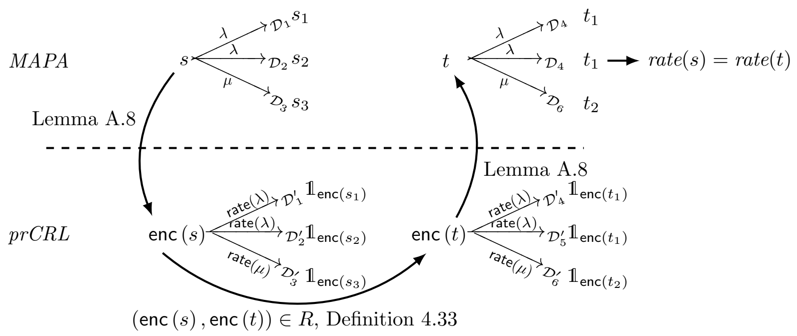


Figure 1.3: Visualisation of the proof of Lemma A.9 part (2).

for every action $\mathsf{rate}(\lambda_i)$ and every equivalence class $[\mathsf{enc}(r)]_R$.

Note that the size of each of these sets is equal to the total number of derivations from $\mathsf{enc}(s)$ and $\mathsf{enc}(t)$ with action $\mathsf{rate}(\lambda_i)$ to a certain equivalence class $[\mathsf{enc}(r)]_R$. This equality immediately implies that, for every action $\mathsf{rate}(\lambda_i)$, the total number of $\mathsf{rate}(\lambda_i)$-derivations from $\mathsf{enc}(s)$ and $\mathsf{enc}(t)$ is also equal. Clearly, if there are as many $\mathsf{rate}(\lambda_i)$-derivations from $\mathsf{enc}(s)$ and $\mathsf{enc}(t)$ for every $\mathsf{rate}(\lambda_i)$, then by definition

$$\sum_{(\lambda_i,\mathcal{D})\in\mathsf{MD}'(\mathsf{enc}(s))} \lambda_i = \sum_{(\lambda_i,\mathcal{D})\in\mathsf{MD}'(\mathsf{enc}(t))} \lambda_i$$

which is what we needed to show.

Figure 1.3 illustrates this part of the proof, and can be helpful for next part as well.

For (b), note that $\mathbb{P}_s(s') = \frac{rate(s,s')}{rate(s)}$ and $\mathbb{P}_t(s') = \frac{rate(t,s')}{rate(t)}$. To show $\mathbb{P}_s \equiv_{R'} \mathbb{P}_t$, we need to prove that $\mathbb{P}_s([p]_{R'}) = \mathbb{P}_t([p]_{R'})$ for every equivalence class $[p]_{R'} \in S/R'$.

Let $[p]_{R'} \in S/R'$, then

$$\mathbb{P}_s([p]_{R'}) = \sum_{p'\in[p]_{R'}} \mathbb{P}_s(p') = \sum_{p'\in[p]_{R'}} \frac{rate(s,p')}{rate(s)} = \frac{\sum_{p'\in[p]_{R'}} rate(s,p')}{rate(s)}$$

Similarly, $\mathbb{P}_t([p]_{R'}) = \frac{\sum_{p'\in[p]_{R'}} rate(t,p')}{rate(t)}$. Since we already showed in part (b) that $rate(s) = rate(t)$, it remains to show that the numerators of these two fractions coincide.

As above, we can derive

$$\sum_{p'\in[p]_{R'}} rate(s,p') = \sum_{p'\in[p]_{R'}} \sum_{(\lambda_i,\mathcal{D})\in\mathsf{MD}(s,p')} \lambda_i = \sum_{(\lambda_i,\mathcal{D})\in\mathsf{MD}(s,[p]_{R'})} \lambda_i$$

where $\mathsf{MD}(s,[p]_{R'}) = \{(\lambda_i,\mathcal{D}) \in \mathbb{R}\times\Delta \mid \exists s' \in [p]_{R'} \;.\; s \xrightarrow{\lambda_i}_{\mathcal{D}} s'\}$. Using

Lemma A.8 we find that $\mathsf{MD}(s, [p]_{R'}) = \mathsf{MD}'(\mathsf{enc}\,(s), [\mathsf{enc}\,(p)]_R)$, where

$$\begin{aligned}&\mathsf{MD}'(\mathsf{enc}\,(s), [\mathsf{enc}\,(p)]_R)\\ &\qquad = \{(\lambda_i, \mathcal{D}) \in \mathbb{R} \times \Delta \mid \exists \mathsf{enc}\,(p') \in [\mathsf{enc}\,(p)]_R \,.\, \mathsf{enc}\,(s) \xrightarrow{\mathsf{rate}(\lambda_i)}_{\mathcal{D}'} \mathbb{1}_{\mathsf{enc}(p')}\}\end{aligned}$$

Similar derivations can be made for t, so it remains to show that

$$\sum_{(\lambda_i,\mathcal{D})\in\mathsf{MD}'(\mathsf{enc}(s),[\mathsf{enc}(p)]_R)} \lambda_i = \sum_{(\lambda_i,\mathcal{D})\in\mathsf{MD}'(\mathsf{enc}(t),[\mathsf{enc}(p)]_R)} \lambda_i$$

Just as in part (b), by assumption we have

$$\begin{aligned}&|\{\mathcal{D} \in \Delta \mid \exists \mathsf{enc}\,(p') \in [\mathsf{enc}\,(p)]_R \,.\, \mathsf{enc}\,(s) \xrightarrow{\mathsf{rate}(\lambda_i)}_{\mathcal{D}} \mathbb{1}_{\mathsf{enc}(p')}\}| =\\ &|\{\mathcal{D} \in \Delta \mid \exists \mathsf{enc}\,(p') \in [\mathsf{enc}\,(p)]_R \,.\, \mathsf{enc}\,(t) \xrightarrow{\mathsf{rate}(\lambda_i)}_{\mathcal{D}} \mathbb{1}_{\mathsf{enc}(p')}\}|\end{aligned}$$

for every action $\mathsf{rate}(\lambda_i)$.

Note again that the size of each of these sets is equal to the total number of derivations from $\mathsf{enc}\,(s)$ and $\mathsf{enc}\,(t)$ with action $\mathsf{rate}(\lambda_i)$ to a certain equivalence class $[\mathsf{enc}\,(p)]_R$. Hence, the fact that these two sets are of equal size implies that every transition $\mathsf{enc}\,(s) \xrightarrow{\mathsf{rate}(\lambda_i)}_{\mathcal{D}} \mathbb{1}_{\mathsf{enc}(s')}$ with $\mathsf{enc}\,(s') \in [\mathsf{enc}\,(p)]_R$ corresponds one-to-one to a transition $\mathsf{enc}\,(t) \xrightarrow{\mathsf{rate}(\lambda_i)}_{\mathcal{D}'} \mathbb{1}_{\mathsf{enc}(t')}$ such that $\mathsf{enc}\,(t') \in [\mathsf{enc}\,(p)]_R$.

This immediately implies that, for every action $\mathsf{rate}(\lambda_i)$, the total number of $\mathsf{rate}(\lambda_i)$-derivations to $[\mathsf{enc}\,(p)]_R$ from $\mathsf{enc}\,(s)$ and $\mathsf{enc}\,(t)$ is also equal. Therefore, by definition

$$\sum_{(\lambda_i,\mathcal{D})\in\mathsf{MD}'(\mathsf{enc}(s),[\mathsf{enc}(p)]_R)} \lambda_i = \sum_{(\lambda_i,\mathcal{D})\in\mathsf{MD}'(\mathsf{enc}(t),[\mathsf{enc}(p)]_R)} \lambda_i$$

which is what we needed to show.

For (c), note that $(\mathsf{enc}\,(s), \mathsf{enc}\,(t)) \in R$ since $(s,t) \in R'$. As there is no τ-transition from s, by Lemma A.7 there is also no τ-transition from $\mathsf{enc}\,(s)$. Since R is a bisimulation relation, also $\mathsf{enc}\,(t)$ does not have a τ-transition, and applying Lemma A.7 again also t does not have a τ-transition. □

Theorem 4.36. *Let $f\colon$ prCRL $\to$ prCRL be a function such that $f(P) \sim_{\mathrm{dp}} P$ for every prCRL specification P, and such that if P is decodable, then so is $f(P)$. Then, $\mathsf{dec}\,(f(\mathsf{enc}\,(M))) \approx_{\mathrm{s}} M$ for every MAPA specification M without any actions labelled by* rate.

Proof. Let M be an arbitrary MAPA specification without any rate action. Since $f(P) \sim_{\mathrm{dp}} P$ for every prCRL specification P, also $f(\mathsf{enc}\,(M)) \sim_{\mathrm{dp}} \mathsf{enc}\,(M)$. Additionally, since $\mathsf{enc}\,(M)$ is decodable by construction, by the assumption on f also $f(\mathsf{enc}\,(M))$ is. Lemma A.9 therefore yields $\mathsf{dec}\,(f(\mathsf{enc}\,(M))) \approx_{\mathrm{s}} \mathsf{dec}\,(\mathsf{enc}\,(M))$. Moreover, $M = \mathsf{dec}\,(\mathsf{enc}\,(M))$ by Lemma A.6, and thus also $M \approx_{\mathrm{s}} \mathsf{dec}\,(\mathsf{enc}\,(M))$. By transitivity of strong bisimulation, the result follows. □

A.2.4 Proof of Theorem 4.45

Lemma A.10. *Algorithm 1 terminates for every finite specification P.*

Proof. The algorithm terminates when *toTransform* eventually becomes empty. Note that every iteration removes exactly one element from *toTransform*. So, if the total number of additions to *toTransform* is finite (and the call to Algorithm 2 never goes into an infinite recursion), the algorithm will terminate.

The elements that are added to *toTransform* are of the form $X'_i(\mathit{pars}) = p_i$, where $p_i \in \text{subterms}(P)$ (recall that subterms were introduced in Definition 4.52). Since P has a finite set of equations with finite right-hand sides, there exists only a finite number of such p_i. Moreover, every process equation $X'_i(\mathit{pars}) = p_i$ that is added to *toTransform* is also added to *bindings*. This makes sure that no process equation $X'_k(\mathit{pars}) = p_i$ is ever added to *toTransform* again, as can be observed from line 3 of Algorithm 2. Hence, the total number of possible additions to *toTransform* is finite.

The fact that Algorithm 2 always terminates relies on not allowing specifications with unguarded recursion. After all, the base case of Algorithm 2 is the action prefix. Therefore, when every recursion in a specification is guarded at some point by an action prefix, this base case is always reached eventually. □

Lemma A.11. *Let $P = (E, X_1(\boldsymbol{v}))$ be a decodable input prCRL specification for Algorithm 1 (having unique variable names), and let $\boldsymbol{v}'$ be the computed new initial vector. Then, before and after an arbitrary iteration of the algorithm's while loop,*

$$(E \cup \text{done} \cup \text{toTransform}, X'_1(\boldsymbol{v}')) \sim_{\text{dp}} P$$

and $(E \cup \text{done} \cup \text{toTransform}, X'_1(\boldsymbol{v}'))$ is decodable.

Proof. The fact that $(E \cup \mathit{done} \cup \mathit{toTransform}, X'_1(\boldsymbol{v}'))$ stays decodable is immediate from the observation that the algorithm uses the process terms from P (which is assumed to be decodable) and never changes or introduces any new probabilistic sums. Hence, from now on we only focus on the $\sim_{\text{dp}}$ part of the lemma.

For brevity, in this proof we write 'bisimilar' if we actually mean 'derivation-preserving bisimilar for all valuations' (since there may be unbound variables). Also, the notation $p \sim_{\text{dp}} q$ will be used for this.

We prove that $(E \cup \mathit{done} \cup \mathit{toTransform}, X'_1(\boldsymbol{v}')) \sim_{\text{dp}} P$ before and after an arbitrary iteration of the algorithm's while loop, by induction on the number of iterations that have already been performed. We let

$$E = \{X_1(\boldsymbol{x_1} : \boldsymbol{D_1}) = p_1, \ldots, X_n(\boldsymbol{x_n} : \boldsymbol{D_n}) = p_n\}$$

and hence we have

$$P = (\{X_1(\boldsymbol{x_1} : \boldsymbol{D_1}) = p_1, \ldots, X_n(\boldsymbol{x_n} : \boldsymbol{D_n}) = p_n\}, X_1(\boldsymbol{v}))$$

Base case. Before the first iteration, the parameters of the new processes are determined. Every process will have the same parameters: $\boldsymbol{x_1} : \boldsymbol{D_1}, \boldsymbol{x'} : \boldsymbol{D'}$. This is the union of all process variables of the original processes, extended with a parameter for every nondeterministic or probabilistic sum binding a variable that is used later on. Also, the new initial state vector $\boldsymbol{v'}$ is computed by taking the original initial vector $\boldsymbol{v}$, and appending dummy values for all added parameters. Furthermore, *done* is set to $\varnothing$ and *toTransform* to $\{X'_1(\boldsymbol{x_1} : \boldsymbol{D_1}, \boldsymbol{x'} : \boldsymbol{D'}) = p_1\}$.

Clearly, $X'_1(\boldsymbol{v'})$ is identical to $X_1(\boldsymbol{v})$, except that it has more global variables (without overlap, as we assumed specifications to have unique variable names). However, these additional global variables are not used in p_1, otherwise they would be free in $X_1(\boldsymbol{x_1} : \boldsymbol{D_1}) = p_1$ (which is not allowed by Definition 4.16). Therefore, $(E \cup \mathit{done} \cup \mathit{toTransform}, X'_1(\boldsymbol{v'}))$ and P are obviously bisimilar (by the smallest equivalence relation that relates $X_1(\boldsymbol{v})$ to $X'_1(\boldsymbol{v'})$ and contains the identity relation).

Since the additional unused variables do not affect the possible derivations in any way, this bisimulation is derivation preserving.

Inductive case. Now assume that $k \geq 0$ iterations have passed. Without loss of generality, assume that each time a process $(X'_i(\mathit{pars}) = p_i) \in \mathit{toTransform}$ had to be chosen, it was the one with the smallest i. Then, after these k iterations,

$$\mathit{done} = \{X'_1(\boldsymbol{x_1} : \boldsymbol{D_1}, \boldsymbol{x'} : \boldsymbol{D'}) = p'_1, \dots, X'_k(\boldsymbol{x_1} : \boldsymbol{D_1}, \boldsymbol{x'} : \boldsymbol{D'}) = p'_k\}$$

Also,

$$\mathit{toTransform} = \{X'_{k+1}(\boldsymbol{x_1} : \boldsymbol{D_1}, \boldsymbol{x'} : \boldsymbol{D'}) = p'_{k+1}, \dots, X'_l(\boldsymbol{x_1} : \boldsymbol{D_1}, \boldsymbol{x'} : \boldsymbol{D'}) = p'_l\}$$

for some $l \geq k$. We have $(E \cup \mathit{done} \cup \mathit{toTransform}, X'_1(\boldsymbol{v'})) \sim_{\mathrm{dp}} P$ by the induction hypothesis.

We prove that after $k+1$ iterations, $(E \cup \mathit{done} \cup \mathit{toTransform}, X'_1(\boldsymbol{v'}))$ is still derivation-preserving bisimilar to P. During iteration $k+1$ three things happen:

1. The process equation $X'_{k+1}(\boldsymbol{x_1} : \boldsymbol{D_1}, \boldsymbol{x'} : \boldsymbol{D'}) = p'_{k+1}$ is removed from *toTransform*;
2. An equation $X'_{k+1}(\boldsymbol{x_1} : \boldsymbol{D_1}, \boldsymbol{x'} : \boldsymbol{D'}) = p''_{k+1}$ is added to *done*;
3. potentially, one or more equations of the form

$$X'_{l+1}(\boldsymbol{x_1} : \boldsymbol{D_1}, \boldsymbol{x'} : \boldsymbol{D'}) = p'_{l+1}, \dots, X'_m(\boldsymbol{x_1} : \boldsymbol{D_1}, \boldsymbol{x'} : \boldsymbol{D'}) = p'_m$$

   are added to *toTransform*.

As the other equations in $E \cup \mathit{done} \cup \mathit{toTransform}$ do not change, Theorem 4.35 implies that $(E \cup \mathit{done} \cup \mathit{toTransform}, X'_1(\boldsymbol{v'})) \sim_{\mathrm{dp}} P$ still holds if $p'_{k+1} \sim_{\mathrm{dp}} p''_{k+1}$. We show this by induction on the number of recursive calls to Algorithm 2 (which depends on the structure of p'_{k+1}). Since unguarded recursion is not allowed by our well-formedness criteria, indeed every process term needs only a finite number of recursive calls.

The base case is when Algorithm 2 terminates without any recursive calls; this is when $p'_{k+1} = a(\boldsymbol{t})\sum_{\boldsymbol{x}:\boldsymbol{D}} f : q$. We now make a case distinction based on whether there already is a process equation in either *done* or *toTransform* whose right-hand side is an IRF corresponding to the normal form of q (which is just q when q is not a process instantiation, otherwise it is the right-hand side of the process it instantiates), as indicated by the variable *bindings*.

Case 1a. *There does not already exist a process equation* $X'_j(\mathit{pars}) = q'$ *in* bindings *such that* q' *is the normal form of* q.

A new process equation $X'_{l+1}(\mathit{pars}) = q'$ is added to *toTransform* via line 6 of Algorithm 2, and $p''_{k+1} = a(\boldsymbol{t})\sum_{\boldsymbol{x}:\boldsymbol{D}} f : X'_{l+1}(\mathit{actualPars})$.

When q was *not* a process instantiation, the actual parameters for X'_{l+1} are just the unchanged global variables, with those that are not used in q reset (line 4 of Algorithm 3). Since (by definition of the normal form) the right-hand side of X'_{l+1} is identical to q and *actualPars* takes care that all data parameters keep the same value, clearly $X'_{l+1}(\mathit{actualPars})$ is derivation-preserving bisimilar to q: every derivation $\mathcal{D}$ of q corresponds to the derivation $\textsc{Inst} + \mathcal{D}$ of $X'_{l+1}(\mathit{actualPars})$. Therefore, by Theorem 4.35 indeed also $p''_{k+1} \sim_{\mathrm{dp}} p'_{k+1}$.

When $q = Y(t_1, t_2, \ldots, t_n)$, there should occur some substitutions to ascertain that $X'_{l+1}(\mathit{actualPars})$ is bisimilar to q. Since we know that $X'_{l+1}(\mathit{actualPars}) = q'$, with q' the right-hand side of Y, the actual parameters to be provided to X'_{l+1} should include $t_1, t_2, \ldots, t_n$ for the global variables of X'_{l+1} that correspond to the original global variables of Y. All other global variables can be reset, as they cannot be used by Y anyway. This indeed happens in line 2 of Algorithm 3, so all behaviours of q are present in $X'_{l+1}(\mathit{actualPars})$ are vice versa, and all derivations of q map one-to-one to the derivations of $X'_{l+1}(\mathit{actualPars})$. So, we find that $q \sim_{\mathrm{dp}} X'_{l+1}(\mathit{actualPars})$, and therefore, by Theorem 4.35, indeed also $p''_{k+1} \sim_{\mathrm{dp}} p'_{k+1}$.

Case 1b. *There exists a process equation* $X'_j(\mathit{pars}) = q'$ *in* bindings *such that* q' *is the normal form of* q.

We obtain

$$p''_{k+1} = a(\boldsymbol{t})\sum_{\boldsymbol{x}:\boldsymbol{D}} f : X'_j(\mathit{actualPars})$$

from line 4 of Algorithm 2. By Theorem 4.35, we only need to show that q is derivation-preserving bisimilar to $X'_j(\mathit{actualPars})$.

Note that the fact that $X'_j(\mathit{pars}) = q'$ is in *bindings* implies that at some point $X'_j(\mathit{pars}) = q'$ was in *toTransform*. In case it was already transformed in an earlier iteration there is now a process $X'_j(\mathit{pars}) = q''$ in *done* such that $q'' \sim_{\mathrm{dp}} q'$ (by induction). Otherwise, $X'_j(\mathit{pars}) = q'$ is still in *toTransform*. In both cases, *done* $\cup$ *toTransform* $\cup\, P$ contains a process $X'_j(\mathit{pars}) = q''$ such that $q'' \sim_{\mathrm{dp}} q'$, and therefore it is correct to take $p''_{k+1} = a(\boldsymbol{t})\sum_{\boldsymbol{x}:\boldsymbol{D}} f : X'_j(\mathit{actualPars})$. The reasoning to see that indeed $p''_{k+1} \sim_{\text{dp}} p'_{k+1}$ then only depends on the choice of *actualPars*, and is the same as for Case 1a.

Now, we assume that Algorithm 2 functions correctly (i.e., provides a derivation-preserving bisimilar result) for all process terms that require k or less recursive calls. Let q_1 and q_2 be two such process terms, for which Algorithm 2 provided the bisimilar process terms p'''_{k+1} and p''''_{k+1} in k steps. Also, assume that p'_{k+1} yields $k+1$ recursive calls. We prove that p''_{k+1} (as obtained from Algorithm 2) is bisimilar to p'_{k+1} by means of a case distinction over the possible structures of p'_{k+1}. (Note that p'_{k+1} cannot be an action prefix, since that would not require any recursive calls.)

Case 2. $p'_{k+1} = c \Rightarrow q_1$.

Algorithm 2 yields $p''_{k+1} = c \Rightarrow p'''_{k+1}$, which according to Theorem 4.35 is bisimilar to p'_{k+1}, since $q_1 \sim_{\text{dp}} p'''_{k+1}$.

Case 3. $p'_{k+1} = q_1 + q_2$.

Algorithm 2 yields $p''_{k+1} = p'''_{k+1} + p''''_{k+1}$, which according to Theorem 4.35 is bisimilar to p'_{k+1}, since $q_1 \sim_{\text{dp}} p'''_{k+1}$ and $q_2 \sim_{\text{dp}} p''''_{k+1}$.

Case 4. $p'_{k+1} = Y(\boldsymbol{t})$, *where we assume that* $Y(\boldsymbol{x} : \boldsymbol{D}) = q_1$.

Algorithm 2 yields $p''_{k+1} = p'''_{k+1}$, with $\boldsymbol{x}$ substituted by $\boldsymbol{t}$, which is bisimilar to p'_{k+1} (as it precisely follows the SOS rule INST). To see that the bisimulation preserves derivations, note that every derivation $\mathcal{D}$ of p'''_{k+1} corresponds one-to-one to a derivation INST $+ \mathcal{D}$ of p'_{k+1}.

Case 5. $p'_{k+1} = \sum_{x:D} q_1$.

In this case, Algorithm 2 yields $p''_{k+1} = \sum_{x:D} p'''_{k+1}$, which according to Theorem 4.35 is bisimilar to p'_{k+1}, since $q_1 \sim_{\text{dp}} p'''_{k+1}$. □

Since in all cases the process term p''_{k+1} obtained from Algorithm 2 is derivation-preserving bisimilar to p'_{k+1} for all valuations, the lemma holds.

Theorem 4.45. *Let P be a decodable prCRL specification such that all variables are named uniquely. Given this input, Algorithm 1 terminates and provides a specification P' such that $P' \sim_{\text{dp}} P$, P' is in IRF and P' is decodable.*

Proof. Let $P = (E, X_1(\boldsymbol{v}))$ be an arbitrary input prCRL specification for Algorithm 1 (having unique variable names), and let $\boldsymbol{v}'$ be the computed new initial vector. Lemma A.10 already provided termination, and Lemma A.11 provided the invariant that $(E \cup \textit{done} \cup \textit{toTransform}, X'_1(\boldsymbol{v}')) \sim_{\text{dp}} P$ and that $(E \cup \textit{done} \cup \textit{toTransform}, X'_1(\boldsymbol{v}'))$ is decodable. As at the end of the algorithm only the equations in *done* are returned, it remains to prove that upon termination $X'_1(\boldsymbol{v}')$ in *done* does not depend on any of the process equations in $E \cup \textit{toTransform}$, and that *done* is in IRF.

First of all, note that upon termination *toTransform* $= \varnothing$ by the condition of the while loop. Moreover, note that the processes that are added to *done* all have a right-hand side determined by Algorithm 2, which only produces process terms that refer to processes in *done* or *toTransform* (in line 4 and line 6). Therefore, $X'_1(\boldsymbol{v}')$ in *done* indeed can only depend on process equations in *done*.

Finally, to show that *done* is indeed in IRF, we need to prove that all probabilistic sums immediately go to a process instantiation, and that process instantiations do not occur in any other way. This is immediately clear from Algorithm 2, as process instantiations are only constructed in line 4 and line 6; there, they indeed are always preceded by a probabilistic sum. Moreover, probabilistic sums are also only constructed by these lines, and are, as required, always succeeded by a process instantiation. Finally, all processes clearly have the same list of global variables (because they are created on line 10 on Algorithm 1 using *pars*, and *pars* never changes). □

A.2.5 Proof of Theorem 4.48

Theorem 4.48. *Let P' be a decodable specification in IRF without a variable pc, and let the output of Algorithm 4 applied to P' be the specification X. Then, $P' \sim_{\text{dp}} X$ and X is decodable.*

Let Y be like X, except that for each summand all nondeterministic sums have been moved to the beginning while substituting their variables by fresh names, and all separate nondeterministic sums and separate conditions have been merged (using vectors and conjunctions, respectively). Then, Y is an LPPE, $Y \sim_{\text{dp}} X$ and Y is decodable.

Proof. The fact that X and Y stay decodable is immediate from the observation that the algorithm uses the process terms from P' (which is assumed to be decodable) and never changes or introduces any new probabilistic sums. Hence, from now on we only focus on the $\sim_{\text{dp}}$ part of the theorem.

Algorithm 4 transforms a specification

$$P' = (\{X'_1(\boldsymbol{x} : \boldsymbol{D}) = p'_1, \ldots, X'_k(\boldsymbol{x} : \boldsymbol{D}) = p'_k\}, X'_1(\boldsymbol{v}))$$

to an LPPE

$$X = (\{X(pc : \{1, \ldots, k\}, \boldsymbol{x} : \boldsymbol{D})\}, X(1, \boldsymbol{v}))$$

by constructing one or more summands for X for every process in P'. Basically, the algorithm just introduces a program counter pc to keep track of the process that is currently active. That is, instead of starting in $X'_1(\boldsymbol{v})$, the system will start in $X(1, \boldsymbol{v})$. Moreover, instead of advancing to $X'_j(\boldsymbol{v})$, the system will advance to $X(j, \boldsymbol{v})$.

The bisimulation relation R to prove the theorem is the smallest equivalence relation containing

$$R = \{(X'_i(\boldsymbol{u}), X(i, \boldsymbol{u})) \mid 1 \leq i \leq k, \boldsymbol{u} \in \boldsymbol{D}\}$$

By definition the initial states are related: $(X'_1(\boldsymbol{v}), X(1, \boldsymbol{v})) \in R$. We additionally need to prove that

$$X'_i(\boldsymbol{u}) \xrightarrow{\alpha} \mu \Longrightarrow \exists \mu' \,.\, X(i, \boldsymbol{u}) \xrightarrow{\alpha} \mu' \wedge \mu \equiv_R \mu'$$

and vice versa, for all $1 \leq i \leq k$ and $\boldsymbol{u} \in \boldsymbol{D}$, and that these pairs $(X'_i(\boldsymbol{u}), X(i, \boldsymbol{u}))$ have the same number of $\mathsf{rate}(\lambda)$-transitions to each equivalence class. We do so by assuming an arbitrary $X'_l(\boldsymbol{u})$, and showing that each derivation (and hence each transition) of $X'_l(\boldsymbol{u})$ maps one-to-one to a derivation of $X(l, \boldsymbol{u})$, by induction over the structure of $X'_l(\boldsymbol{x} : \boldsymbol{D})$.

The base case is $X'_l(\boldsymbol{x} : \boldsymbol{D}) = a(\boldsymbol{t}) \sum_{\boldsymbol{y}:\boldsymbol{E}} f : X'_j(t'_1, \ldots, t'_k)$. For this process, Algorithm 4 constructs the summand $pc = l \Rightarrow a(\boldsymbol{t}) \sum_{\boldsymbol{y}:\boldsymbol{E}} f : X(j, t'_1, \ldots, t'_k)$. As every summand constructed by the algorithm contains a condition $pc = i$, and the summands produced for $X'_l(\boldsymbol{x} : \boldsymbol{D})$ are the only ones producing a summand with $i = l$ (all others require pc to have a different value), it follows that $X'_l(\boldsymbol{u})$ and $X(l, \boldsymbol{u})$ have precisely the same derivations, except that each target state $X'_j(\boldsymbol{u}')$ of $X'_l(\boldsymbol{u})$ is mimicked by $X(l, \boldsymbol{u})$ using a target state $X(j, \boldsymbol{u}')$. Since these are all R-related, indeed $X'_l(\boldsymbol{u})$ and $X(l, \boldsymbol{u})$ have the same derivations (and hence transitions) modulo R.

Now assume that $X'_l(\boldsymbol{x} : \boldsymbol{D}) = c \Rightarrow q$. By induction, $X''_l(\boldsymbol{x} : \boldsymbol{D}) = q$ would result in the construction of one or more summands such that $X''_l(\boldsymbol{u})$ and $X(l, \boldsymbol{u})$ have the same derivations. For $X'_l(\boldsymbol{x} : \boldsymbol{D})$ the algorithm takes those summands, and adds the condition c to all of them. Therefore, $X'_l(\boldsymbol{u})$ and $X(l, \boldsymbol{u})$ also have the same derivations (and hence transitions). Similar arguments can be used for $X'_l(\boldsymbol{x} : \boldsymbol{D}) = q_1 + q_2$ or $X'_l(\boldsymbol{x} : \boldsymbol{D}) = \sum_{\boldsymbol{x}:\boldsymbol{D}} q$. Hence, $P' \sim_{\mathrm{dp}} X$.

Now, let Y be equal to X, except that within each summand all nondeterministic sums have been moved to the beginning while substituting their variables by fresh names, and all separate nondeterministic sums and separate conditions have been merged (using vectors and conjunctions, respectively).

To see that Y is an LPPE, first observe that X already was a single process equation consisting of a set of summands. Each of these contains a number of nondeterministic sums and conditions, followed by a probabilistic sum. Furthermore, each probabilistic sum is indeed followed by a process instantiation, as can be seen from line 6 of Algorithm 4.

The only discrepancy for X to be an LPPE is that the nondeterministic sums and the conditions are not yet necessarily in the right order, and there may be several of them. For instance, we may have something like

$$d > 0 \Rightarrow \sum_{d:D_1} e > 0 \Rightarrow \sum_{f:D_2} act(d).X(d, f)$$

However, since the nondeterministic sums and conditions are swapped in Y such that all nondeterministic sums precede all conditions, and since conditions are merged using conjunctions, and summations using vectors, Y is an LPPE. In case of the example before, we obtain

$$\sum_{(d',f):D_1 \times D_2} d > 0 \wedge e > 0 \Rightarrow act(d').X(d', f)$$

Note that, indeed, some renaming had to be done such that $Y \sim_{\text{dp}} X$. After all, by renaming the summation variables, we can rely on the obvious equality of the process terms $c \Rightarrow \sum_{d:D} p$ and $\sum_{d:D} c \Rightarrow p$, given that d is not a free variable in c. Thus, swapping the conditions and nondeterministic sums this way does not modify the process' semantics in any way. Also, it is easy to see from the SOS rules in Table 4.5 that merging nondeterministic sums and conditions does not influence the transitions or the number of $\mathsf{rate}(\lambda)$-derivations of the process in any way. Therefore, indeed $Y \sim_{\text{dp}} X$. $\square$

A.2.6 Proof of Proposition 4.51

Proposition 4.51. *Let P be a prCRL specification such that size$(P) = n$. Then, the worst-case time complexity of linearising P is $O(n^3)$. The worst-case size of the resulting LPPE is in $O(n^2)$.*

Proof. Let $P = (E, I)$ be a specification such that *size*$(P) = n$. First of all, note that

$$\left|pars\right| \leq \sum_{(X_i(\boldsymbol{x_i} : \boldsymbol{D_i}) = p_i) \in E} |\boldsymbol{x_i}| + \left|\text{subterms}'(P)\right| \leq n \tag{A.6}$$

after the initialisation of Algorithm 1, where $|pars|$ denotes the numbers of new global variables and *subterms*$'(P)$ denotes the *multiset* containing all subterms of P (counting a process term that occurs twice as two subterms, and including nondeterministic and probabilistic choices over a vector of k variables k times). When mentioning the subterms of P in this proof, we will be referring to this multiset (for a formal definition of subterms, see Definition 4.52 on page 95).

The first inequality follows from the fact that *pars* is defined to be the sequence of all $\boldsymbol{x_i}$ appended by all local variables of P (that are syntactically used), and the observation that there are at most as many local variables as there are subterms. The second inequality follows from the definition of *size* and the observation that *size*(p_i) counts the number of subterms of p plus the size of their expressions.

Time complexity. We first determine the worst-case time complexity of Algorithm 1. As the function *transform* is called at most once for every subterm of P, it follows from Equation (A.6) that the number of times this happens is in $O(n)$. The time complexity of every such call is governed by the call to *normalForm*.

The function *normalForm* checks for each global variable in *pars* whether or not it can be reset; from Equation (A.6) we know that the number of such variables is in $O(n)$. To check whether a global variable can be reset given a process term p, we have to examine every expression in p; as the size of the expressions is accounted for by n, this is also in $O(n)$. So, the worst-case time complexity of *normalForm* is in $O(n^2)$. Therefore, the worst-case time complexity of Algorithm 1 is in $O(n^3)$.

As the transformation from IRF to LPPE by Algorithm 4 is easily seen to be in $O(n)$, we find that, in total, linearisation has a worst-case time complexity in $O(n^3)$.

LPPE size complexity. Every summand of the LPPE X that is constructed has a size in $O(n)$. After all, each contains a process instantiation with an expression for every global variable in *pars*, and we already saw that the number of them is in $O(n)$. Furthermore, the number of summands is bound from above by the number of subterms of P, so this is in $O(n)$. Therefore, the size of X is in $O(n^2)$. □

A.2.7 Proof of Proposition 4.56

Proposition 4.56. *For all $\boldsymbol{v} \in \boldsymbol{G}, \boldsymbol{v}' \in \boldsymbol{G}'$, it holds that*

$$Z(\boldsymbol{v}, \boldsymbol{v}') \approx_{\text{iso}} X(\boldsymbol{v}) \,||\, Y(\boldsymbol{v}')$$

Proof. The only processes an MLPE $Z(\boldsymbol{v}, \boldsymbol{v}')$ can become, are of the form $Z(\hat{\boldsymbol{v}}, \hat{\boldsymbol{v}}')$, and the only processes a parallel composition $X(\boldsymbol{v}) \,||\, Y(\boldsymbol{v}')$ can become, are of the form $X(\hat{\boldsymbol{v}}) \,||\, Y(\hat{\boldsymbol{v}}')$. Therefore, the isomorphism h needed to prove the proposition is as follows:

$$h(X(\boldsymbol{v}) \,||\, Y(\boldsymbol{v}')) = Z(\boldsymbol{v}, \boldsymbol{v}') \qquad \text{and} \qquad h(Z(\boldsymbol{v}, \boldsymbol{v}')) = X(\boldsymbol{v}) \,||\, Y(\boldsymbol{v}')$$

for all $\boldsymbol{v} \in \boldsymbol{G}, \boldsymbol{v}' \in \boldsymbol{G}'$, and $h(p) = p$ for every process term p of a different form. Clearly, h is bijective. We will now show that indeed $X(\boldsymbol{v}) \,||\, Y(\boldsymbol{v}') \xrightarrow{\alpha} \mu$ if and only if $Z(\boldsymbol{v}, \boldsymbol{v}') \xrightarrow{\alpha} \mu_h$. We do so by showing that

$$X(\boldsymbol{v}) \,||\, Y(\boldsymbol{v}') \xhookrightarrow{a(\boldsymbol{q})} \mu \text{ if and only if } Z(\boldsymbol{v}, \boldsymbol{v}') \xhookrightarrow{a(\boldsymbol{q})} \mu_h$$

and

$$X(\boldsymbol{v}) \,||\, Y(\boldsymbol{v}') \overset{\lambda}{\rightsquigarrow} X(\hat{\boldsymbol{v}}') \,||\, Y(\hat{\boldsymbol{v}}') \text{ if and only if } Z(\boldsymbol{v}, \boldsymbol{v}') \overset{\lambda}{\rightsquigarrow} \mu(X(\hat{\boldsymbol{v}}') \,||\, Y(\hat{\boldsymbol{v}}'))$$

Let $\boldsymbol{v} \in \boldsymbol{G}$ and $\boldsymbol{v}' \in \boldsymbol{G}'$ be arbitrary global variables vectors for X and Y. Then, by the operational semantics of parallel MAPA, $X(\boldsymbol{v}) \,||\, Y(\boldsymbol{v}') \xhookrightarrow{a(\boldsymbol{q})} \mu$ is enabled if and only if at least one of the following three conditions holds.

$$\begin{array}{ll}
(1) & X(\boldsymbol{v}) \xhookrightarrow{a(\boldsymbol{q})} \mu' \wedge \forall \hat{\boldsymbol{v}} \in \boldsymbol{G} \,.\, \mu(X(\hat{\boldsymbol{v}}) \,||\, Y(\boldsymbol{v}')) = \mu'(\hat{\boldsymbol{v}}) \\
(2) & Y(\boldsymbol{v}') \xhookrightarrow{a(\boldsymbol{q})} \mu' \wedge \forall \hat{\boldsymbol{v}}' \in \boldsymbol{G}' \,.\, \mu(X(\boldsymbol{v}) \,||\, Y(\hat{\boldsymbol{v}}')) = \mu'(\hat{\boldsymbol{v}}') \\
(3) & X(\boldsymbol{v}) \xhookrightarrow{a'(\boldsymbol{q})} \mu' \wedge Y(\boldsymbol{v}') \xhookrightarrow{a''(\boldsymbol{q})} \mu'' \wedge \gamma(a', a'') = a \wedge \\
 & \forall \hat{\boldsymbol{v}} \in \boldsymbol{G}, \hat{\boldsymbol{v}}' \in \boldsymbol{G}' \,.\, \mu(X(\hat{\boldsymbol{v}}) \,||\, Y(\hat{\boldsymbol{v}}')) = \mu'(\hat{\boldsymbol{v}}) \cdot \mu''(\hat{\boldsymbol{v}}')
\end{array}$$

It immediately follows from the construction of Z that $Z(\boldsymbol{v}, \boldsymbol{v}') \xhookrightarrow{a(\boldsymbol{q})} \mu_h$ is enabled under exactly the same conditions, as condition (1) is covered by the first set of summands of Z, condition (2) is covered by the second set of summands of Z, and condition (3) is covered by the third set of summands of Z.

We now show that $X(\boldsymbol{v}) \,||\, Y(\boldsymbol{v}') \overset{\lambda}{\rightsquigarrow} X(\hat{\boldsymbol{v}}) \,||\, Y(\hat{\boldsymbol{v}}')$ if and only if $Z(\boldsymbol{v}, \boldsymbol{v}') \overset{\lambda}{\rightsquigarrow} h(X(\hat{\boldsymbol{v}}) \,||\, Y(\hat{\boldsymbol{v}}'))$, by making a case distinction between (1) $\hat{\boldsymbol{v}} = \boldsymbol{v}$ and $\hat{\boldsymbol{v}}' = \boldsymbol{v}'$, (2) $\hat{\boldsymbol{v}} \neq \boldsymbol{v}$ and $\hat{\boldsymbol{v}}' = \boldsymbol{v}'$, (3) $\hat{\boldsymbol{v}} = \boldsymbol{v}$ and $\hat{\boldsymbol{v}}' \neq \boldsymbol{v}'$, and (4) $\hat{\boldsymbol{v}} \neq \boldsymbol{v}$ and $\hat{\boldsymbol{v}}' \neq \boldsymbol{v}'$.

(1) If $\hat{\boldsymbol{v}} = \boldsymbol{v}$ and $\hat{\boldsymbol{v}}' = \boldsymbol{v}'$, then by the operational semantics a transition $X(\boldsymbol{v}) \,||\, Y(\boldsymbol{v}') \overset{\lambda}{\rightsquigarrow} X(\hat{\boldsymbol{v}}) \,||\, Y(\hat{\boldsymbol{v}}')$ is enabled if λ is the sum of all rates λ_i

such that there is a derivation $X(\boldsymbol{v}) \xrightarrow{\lambda_i}_{\mathcal{D}} X(\boldsymbol{v})$ or $Y(\boldsymbol{v'}) \xrightarrow{\lambda_i}_{\mathcal{D}} Y(\boldsymbol{v'})$. By construction of Z, it follows immediately that $Z(\boldsymbol{v}, \boldsymbol{v'})$ has precisely the same number of derivations with the same rates to go to $Z(\boldsymbol{v}, \boldsymbol{v'})$, via its fourth and fifth set of summands. Hence, $Z(\boldsymbol{v}, \boldsymbol{v'}) \overset{\lambda}{\rightsquigarrow} Z(\boldsymbol{v}, \boldsymbol{v'})$ is enabled under exactly the same conditions as $X(\boldsymbol{v}) \,||\, Y(\boldsymbol{v'}) \overset{\lambda}{\rightsquigarrow} X(\hat{\boldsymbol{v}}) \,||\, Y(\hat{\boldsymbol{v}}')$. Since $Z(\boldsymbol{v}, \boldsymbol{v'}) = h(X(\boldsymbol{v}) \,||\, Y(\boldsymbol{v'})) = h(X(\hat{\boldsymbol{v}}) \,||\, Y(\hat{\boldsymbol{v}}'))$, this completes the proof.

(2) If $\hat{\boldsymbol{v}} \neq \boldsymbol{v}$ and $\hat{\boldsymbol{v}}' = \boldsymbol{v'}$, then by the operational semantics a transition $X(\boldsymbol{v}) \,||\, Y(\boldsymbol{v'}) \overset{\lambda}{\rightsquigarrow} X(\hat{\boldsymbol{v}}) \,||\, Y(\hat{\boldsymbol{v}}')$ is enabled if λ is the sum of all rates λ_i such that there is a derivation $X(\boldsymbol{v}) \xrightarrow{\lambda_i}_{\mathcal{D}} X(\hat{\boldsymbol{v}})$. By construction of Z, it follows immediately that $Z(\boldsymbol{v}, \boldsymbol{v'})$ has precisely the same number of derivations with the same rates to go to $Z(\hat{\boldsymbol{v}}, \boldsymbol{v'})$, via its fourth set of summands. Hence, $Z(\boldsymbol{v}, \boldsymbol{v'}) \overset{\lambda}{\rightsquigarrow} Z(\hat{\boldsymbol{v}}, \boldsymbol{v'})$ is enabled under exactly the same conditions as $X(\boldsymbol{v}) \,||\, Y(\boldsymbol{v'}) \overset{\lambda}{\rightsquigarrow} X(\hat{\boldsymbol{v}}) \,||\, Y(\hat{\boldsymbol{v}}')$. Since $Z(\hat{\boldsymbol{v}}, \boldsymbol{v'}) = h(X(\hat{\boldsymbol{v}}) \,||\, Y(\boldsymbol{v'})) = h(X(\hat{\boldsymbol{v}}) \,||\, Y(\hat{\boldsymbol{v}}'))$, this completes the proof.

(3) Symmetric to the previous case.

(4) It immediately follows from the operational semantics and the construction of Z that neither $X(\boldsymbol{v}) \,||\, Y(\boldsymbol{v'})$ nor $Z(\boldsymbol{v}, \boldsymbol{v'})$ has any Markovian transitions that change both $\boldsymbol{v}$ and $\boldsymbol{v'}$. So, if $\hat{\boldsymbol{v}} \neq \boldsymbol{v}$ and $\hat{\boldsymbol{v}}' \neq \boldsymbol{v'}$, then trivially $X(\boldsymbol{v}) \,||\, Y(\boldsymbol{v'}) \overset{\lambda}{\rightsquigarrow} X(\hat{\boldsymbol{v}}) \,||\, Y(\hat{\boldsymbol{v}}')$ if and only if $Z(\boldsymbol{v}, \boldsymbol{v'}) \overset{\lambda}{\rightsquigarrow} h(X(\hat{\boldsymbol{v}}) \,||\, Y(\hat{\boldsymbol{v}}'))$.□

A.2.8 Proof of Proposition 4.59

Proposition 4.59. *For all $\boldsymbol{v} \in \boldsymbol{G}$, $U(\boldsymbol{v}) \approx_{\text{iso}} \tau_H(X(\boldsymbol{v}))$, $V(\boldsymbol{v}) \approx_{\text{iso}} \rho_R(X(\boldsymbol{v}))$, and $W(\boldsymbol{v}) \approx_{\text{iso}} \partial_E(X(\boldsymbol{v}))$.*

Proof. We will prove that $U(\boldsymbol{v}) \approx_{\text{iso}} \tau_H(X(\boldsymbol{v}))$ for all $\boldsymbol{v} \in \boldsymbol{G}$; the other two statements are proven similarly.

The only processes an MLPE $X(\boldsymbol{v})$ can become are of the form $X(\boldsymbol{v'})$. Moreover, as hiding does not change the process structure, the only processes that $\tau_H(X(\boldsymbol{v}))$ can become, are processes of the form $\tau_H(X(\boldsymbol{v'}))$. Therefore, the isomorphism h needed to prove the proposition is easy: for all $\boldsymbol{v} \in \boldsymbol{G}$, we define $h(\tau_H(X(\boldsymbol{v}))) = U(\boldsymbol{v})$ and $h(U(\boldsymbol{v})) = \tau_H(X(\boldsymbol{v}))$, and $h(p) = p$ for every p of a different form. Clearly, h is bijective.

We now show that h indeed is an isomorphism. To do so, we first show that (1) $\tau_H(X(\boldsymbol{v})) \xhookrightarrow{a(\boldsymbol{q})} \mu$ if and only if $h(\tau_H(X(\boldsymbol{v}))) \xhookrightarrow{a(\boldsymbol{q})} \mu_h$, i.e., if and only if $U(\boldsymbol{v}) \xhookrightarrow{a(\boldsymbol{q})} \mu_h$. Second, we show that (2) $\tau_H(X(\boldsymbol{v})) \overset{\lambda}{\rightsquigarrow} \tau_H(X(\boldsymbol{v'}))$ if and only if $U(\boldsymbol{v}) \overset{\lambda}{\rightsquigarrow} U(\boldsymbol{v'})$.

(1) First, recall that $X(\boldsymbol{v}) \xhookrightarrow{a(\boldsymbol{q})} \mu$ is enabled if and only if there is a summand $i \in I$ and a local variables vector $\boldsymbol{d_i'} \in \boldsymbol{D_i}$ such that

$$c_i(\boldsymbol{v}, \boldsymbol{d_i'}) \wedge a_i(\boldsymbol{b_i}(\boldsymbol{v}, \boldsymbol{d_i'})) = a(\boldsymbol{q}) \wedge \mu = \mathit{target}_i(\boldsymbol{v}, \boldsymbol{d_i})$$

where $\mathit{target}_i(\boldsymbol{v}, \boldsymbol{d_i'})$ is the distribution μ' such that

$$\forall \boldsymbol{e_i'} \in \boldsymbol{E_i} \,.\, \mu'(\boldsymbol{n_i}(\boldsymbol{v}, \boldsymbol{d_i'}, \boldsymbol{e_i'})) = \sum_{\substack{\boldsymbol{e_i''} \in \boldsymbol{E_i} \\ \boldsymbol{n_i}(\boldsymbol{v}, \boldsymbol{d_i'}, \boldsymbol{e_i'}) = \boldsymbol{n_i}(\boldsymbol{v}, \boldsymbol{d_i'}, \boldsymbol{e_i''})}} f_i(\boldsymbol{v}, \boldsymbol{d_i'}, \boldsymbol{e_i''})$$

We now make a case distinction between (a) $a \neq \tau$ or $\boldsymbol{q} \neq (\,)$, and (b) $a = \tau$ and $\boldsymbol{q} = (\,)$.

(a) If $a \neq \tau$ or $\boldsymbol{q} \neq (\,)$, then the operational semantics imply that $\tau_H(X(\boldsymbol{v})) \xhookrightarrow{a(\boldsymbol{q})} \mu$ is enabled if and only if $X(\boldsymbol{v}) \xhookrightarrow{a(\boldsymbol{q})} \mu \wedge a \notin H$. Moreover, $U(\boldsymbol{v}) \xhookrightarrow{a(\boldsymbol{q})} \mu_h$ is enabled if and only if there is a summand $i \in I$ and a local variables vector $\boldsymbol{d'_i} \in \boldsymbol{D_i}$ such that

$$c_i(\boldsymbol{v}, \boldsymbol{d'_i}) \wedge a'_i(\boldsymbol{b'_i}(\boldsymbol{v}, \boldsymbol{d'_i})) = a(\boldsymbol{q}) \wedge \mu = \mathit{target}_i(\boldsymbol{v}, \boldsymbol{d_i})$$

which indeed corresponds to $X(\boldsymbol{v}) \xhookrightarrow{a(\boldsymbol{q})} \mu \wedge a \notin H$ by definition of a'_i and $\boldsymbol{b'_i}$ and the assumption that $a \neq \tau$ or $\boldsymbol{q} \neq (\,)$.

(b) If $a = \tau$ and $\boldsymbol{q} = (\,)$, then the operational semantics imply that $\tau_H(X(\boldsymbol{v})) \xhookrightarrow{\tau} \mu$ is enabled if and only if $X(\boldsymbol{v}) \xhookrightarrow{\tau} \mu$ is enabled or there exists some $a \in H$ with parameters $\boldsymbol{q'}$ such that $X(\boldsymbol{v}) \xhookrightarrow{a(\boldsymbol{q'})} \mu$ is enabled. It immediately follows by definition of a'_i and $\boldsymbol{b'_i}$ that $U(\boldsymbol{v}) \xhookrightarrow{\tau} \mu_h$ is enabled under exactly these conditions.

(2) Second, note that, since Markovian transitions cannot be hidden, we have $\tau_H(X(\boldsymbol{v})) \overset{\lambda}{\rightsquigarrow} \tau_H(X(\boldsymbol{v'}))$ if and only if $X(\boldsymbol{v}) \overset{\lambda}{\rightsquigarrow} X(\boldsymbol{v'})$ and there is no μ such that $\tau_H(X(\boldsymbol{v})) \xhookrightarrow{\tau} \mu$. Additionally, as $X(\boldsymbol{g} : \boldsymbol{G})$ and $U(\boldsymbol{g} : \boldsymbol{G})$ have exactly the same Markovian summands, we find that $U(\boldsymbol{v}) \overset{\lambda}{\rightsquigarrow} U(\boldsymbol{v'})$ if and only if $X(\boldsymbol{v}) \overset{\lambda}{\rightsquigarrow} X(\boldsymbol{v'})$ and there is no μ such that $U(\boldsymbol{v}) \xhookrightarrow{\tau} \mu$ (the 'only if' part is due to the fact that $X(\boldsymbol{v}) \xhookrightarrow{\tau} \mu$ implies that there is a μ' such that $U(\boldsymbol{v}) \xhookrightarrow{\tau} \mu'$).

By the first part of this proof, there is no μ such that $\tau_H(X(\boldsymbol{v})) \xhookrightarrow{\tau} \mu$ if and only if there is no μ such that $U(\boldsymbol{v}) \xhookrightarrow{\tau} \mu$. Hence, when combining this with the findings above, we see that

$$\tau_H(X(\boldsymbol{v})) \overset{\lambda}{\rightsquigarrow} \tau_H(X(\boldsymbol{v'})) \text{ if and only if } U(\boldsymbol{v}) \overset{\lambda}{\rightsquigarrow} U(\boldsymbol{v'})$$

Since we defined $U(\boldsymbol{v'}) = h(\tau_H(X(\boldsymbol{v'})))$, this completes the proof. □

A.2.9 Proof of Proposition 4.61

Proposition 4.61. *The underlying MAs of an MLPE before and after constant elimination are isomorphic.*

Proof (sketch). We first show that every parameter that is marked constant by the procedure, indeed has the same value in every reachable state of the state space. Let x be a parameter with initial value x^0 that is not constant, so there exists a state $\boldsymbol{v}$ where x is not equal to x^0. Then there must be a summand that semantically changes x to a value different from x^0. If the next state of x in this summand is syntactically given by the expression x, and x is changed because it is bound to a value $x' \neq x^0$ by a nondeterministic or probabilistic sum, this is detected by the procedure in the first iteration. If the next state is given by an expression e not equal to x, then e must also be different from x^0 (otherwise the value of x would still not change). This will also be detected in the first iteration, except if e is the name of another parameter y that is initially, but not constantly, equal to x^0. Either the first iteration detects y to be non-constant and the second iteration will then see that x is also non-constant, or y is also non-constant because of a valuation based on another parameter. In the latter case, it may take some more iterations, but as y is non-constant this recursion clearly ends at some point (as a cyclicity would imply that x is constant, violating our assumption).

Now, as the procedure correctly computes the parameters of an MLPE that are constant, it is trivial that changing all their occurrences to their initial values is a valid transformation; it does not change the semantics of the MLPE and therefore leaves the reachable state space (including its transitions) untouched. Moreover, as the constant parameters are not used anymore after this step, they have no influence on the semantics of the MLPE, and removing them also does also not change anything (except for state names, which is allowed by isomorphism). □

A.2.10 Proof of Proposition 4.65

Proposition 4.65. *The underlying MAs of an MLPE before and after summation elimination are isomorphic.*

Proof (sketch). Assume a summand with a summation $\sum_{d:D}$ and a condition c. Clearly, when c is given by $d = e$ or $e = d$, and e does not contain d, the condition can indeed only be satisfied when d is equal to e, so, for any other value the summand would not be enabled. Given a condition $e_1 \wedge e_2$, the summand is only enabled when both e_1 and e_2 hold, so clearly indeed only when d has a value in the intersection of the sets containing the values for those two conditions to hold. For a disjunction $e_1 \vee e_2$, knowing that for e_i to hold d should have a value in S_i, clearly it should have a value in $S_1 \cup S_2$ for the disjunction to hold. However, when either S_1 or S_2 is empty, this implies that we don't know the values that satisfy this disjunct, so then we also don't know anything about the complete disjunction.

If in the end there is precisely one value d' that enables the condition c, this means that the summand can only be taken for this value. Therefore, clearly we can just as well omit the nondeterministic choice and substitute d' for every free occurrence of d in the resulting summand. This obviously changes nothing about the underlying MA, so the transformation preserves isomorphism.

It should also be immediately obvious from the operational semantics that summations over unused variables can be omitted in interactive summands, and that summations in Markovian summands over variables that are only used directly after the summation can be changed as explained in Section 4.5.3. □

A.3 Proofs for Chapter 5

To simplify the proofs for this chapter, we first define a notation for the target of a summand. As in Section 4.2.4, we sometimes use state vectors $\boldsymbol{v}$ as abbreviations for process terms $X(\boldsymbol{v})$ when the process is clear from the context.

Definition A.12 (LPPE target function). *Given an LPPE*

$$X(\boldsymbol{g} : \boldsymbol{G}) = \sum_{i \in I} \sum_{\boldsymbol{d_i} : \boldsymbol{D_i}} c_i \Rightarrow a_i(\boldsymbol{b_i}) \sum_{\boldsymbol{e_i} : \boldsymbol{E_i}} f_i : X(\boldsymbol{n_i})$$

we define, for each $i \in I$, the function $target_i \colon \boldsymbol{G} \times \boldsymbol{D_i} \to \mathrm{Distr}(\boldsymbol{G})$ by

$target_i(\boldsymbol{v}, \boldsymbol{d'_i}) = \mu$ such that

$$\forall \boldsymbol{e'_i} \in \boldsymbol{E_i} \;.\; \mu(\boldsymbol{n_i}(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e'_i})) = \sum_{\substack{\boldsymbol{e''_i} \in \boldsymbol{E_i} \\ \boldsymbol{n_i}(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e'_i}) = \boldsymbol{n_i}(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e''_i})}} f_i(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e''_i})$$

Hence, $target_i(\boldsymbol{v}, \boldsymbol{d'_i})$ provides the next-state distribution of $X(\boldsymbol{v})$ after taking summand i with a local variable vector $\boldsymbol{d'_i}$ (assuming that summand i is indeed enabled given $\boldsymbol{v}$ and $\boldsymbol{d'_i}$).

A.3.1 Proof of Proposition 5.11

Proposition 5.11. *After executing Algorithm 5, the set $Reach_j$ contains all values that a CFP g_j may obtain.*

Proof. The proof is by induction on the number of transitions by summands in the cluster of g_j that have to be taken to reach a value. We prove that all values that g_j may take in n such transitions are included before or during iteration n (where the first iteration has index 0). Note that the number of values g_j can take is finite, as it is limited by the number of summands.

The only value g_j can have after 0 transitions is $init_j$. By the first line of the algorithm, this value is indeed included, so all values g_j may take in 0 transitions are included in $Reach_j$ before or during iteration 0.

Now assume that all values g_j may take in k transitions are included in $Reach_j$ before or during iteration k. Furthermore, let v be a value g_j can obtain in $k+1$ transitions, so $init_j = v^0 \to v^1 \to \cdots \to v^k \to v^{k+1} = v$. By the induction hypothesis $v^k \in Reach_j$ during iteration $k+1$. Furthermore, since $v^k \to v$, there must be some summand $i \in R_{g_j}$ such that $v^k = source(i, g_j)$ and $v = dest(i, g_j)$. Also, $v^k \notin Prev$ during iteration $k+1$, otherwise v would have been reachable in less than $k+1$ steps. Hence, v^{k+1} is added during iteration $k+1$ by the innermost statement of the algorithm.

Termination of the algorithm immediately follows from the observation that the number of values a CFP may take is finite. □

A.3.2 Proof of Theorem 5.21

Lemma A.13. *Given a decodable LPPE*

$$X(\boldsymbol{g} : \boldsymbol{G}) = \sum_{i \in I} \sum_{\boldsymbol{d_i} : \boldsymbol{D_i}} c_i \Rightarrow a_i(\boldsymbol{b_i}) \sum_{\boldsymbol{e_i} : \boldsymbol{E_i}} f_i : X(\boldsymbol{n_i})$$

and an equivalence relation $R \subseteq \boldsymbol{G} \times \boldsymbol{G}$, if

$$\begin{aligned} c_i(\boldsymbol{v}, \boldsymbol{d'_i}) \implies & c_i(\boldsymbol{v'}, \boldsymbol{d'_i}) \wedge \boldsymbol{b_i}(\boldsymbol{v}, \boldsymbol{d'_i}) = \boldsymbol{b_i}(\boldsymbol{v'}, \boldsymbol{d'_i}) \wedge \\ & \mathit{target}_i(\boldsymbol{v}, \boldsymbol{d'_i}) \equiv_R \mathit{target}_i(\boldsymbol{v'}, \boldsymbol{d'_i}) \end{aligned}$$

for every $(\boldsymbol{v}, \boldsymbol{v'}) \in R$, $i \in I$ and $\boldsymbol{d'_i} \in \boldsymbol{D_i}$, then R is a derivation-preserving bisimulation.

Proof. To see that R is a strong bisimulation relation, take an arbitrary pair $(\boldsymbol{v}, \boldsymbol{v'}) \in R$ and let $\boldsymbol{v} \xhookrightarrow{\alpha} \mu$. Note that, by the operational semantics, this implies that for at least one summand $i \in I$, there is a local choice $\boldsymbol{d'_i} \in \boldsymbol{D_i}$ such that $c_i(\boldsymbol{v}, \boldsymbol{d'_i})$ holds, $\alpha = a_i(\boldsymbol{b_i}(\boldsymbol{v}, \boldsymbol{d'_i}))$ and $\mathit{target}_i(\boldsymbol{v}, \boldsymbol{d'_i}) = \mu$.

Then, by the assumption on R, we have

$$c_i(\boldsymbol{v'}, \boldsymbol{d'_i}) \wedge \boldsymbol{b_i}(\boldsymbol{v}, \boldsymbol{d'_i}) = \boldsymbol{b_i}(\boldsymbol{v'}, \boldsymbol{d'_i}) \wedge \mathit{target}_i(\boldsymbol{v}, \boldsymbol{d'_i}) \equiv_R \mathit{target}_i(\boldsymbol{v'}, \boldsymbol{d'_i})$$

By the operational semantics this immediately implies that $\boldsymbol{v'} \xhookrightarrow{\alpha} \mu'$ such that $\mu \equiv_R \mu'$, which is what we needed to show for strong bisimulation.

It remains to show that for every equivalence equivalence class $[\boldsymbol{w}]_R$ and every rate λ:

$$\begin{aligned} &|\{\mathcal{D} \in \Delta \mid \exists \boldsymbol{w'} \in [\boldsymbol{w}]_R \,.\, \boldsymbol{v} \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mathbb{1}_{\boldsymbol{w'}}\}| = \\ &|\{\mathcal{D} \in \Delta \mid \exists \boldsymbol{w''} \in [\boldsymbol{w}]_R \,.\, \boldsymbol{v'} \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mathbb{1}_{\boldsymbol{w''}}\}| \end{aligned}$$

It is easy to see from the LPPE's structure and the operational semantics that every derivation $\boldsymbol{v} \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mathbb{1}_{\boldsymbol{w'}}$ corresponds to selecting a summand i and a local variable vector $\boldsymbol{d'_i}$ such that $c_i(\boldsymbol{v}, \boldsymbol{d'_i})$ holds, $\mathsf{rate}(\lambda) = a_i(\boldsymbol{b_i}(\boldsymbol{v}, \boldsymbol{d'_i}))$ and $\mathit{target}_i(\boldsymbol{v}, \boldsymbol{d'_i}) = \mathbb{1}_{\boldsymbol{w'}}$. Hence, we can specify a derivation by a pair $(i, \boldsymbol{d'_i})$.

Given such a derivation $(i, \boldsymbol{d'_i})$, the assumption on R implies that $(i, \boldsymbol{d'_i})$ is also a derivation for $\boldsymbol{v'}$, yielding a transition $\boldsymbol{v'} \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mu$ such that $\mu \equiv_R \mathbb{1}_{\boldsymbol{w'}}$. Since we assumed that the LPPE is decodable, this implies that $\mu = \mathbb{1}_{\boldsymbol{w''}}$ for some $\boldsymbol{w''} \in [\boldsymbol{w'}]_R$. Hence, every derivation for a transition $\boldsymbol{v} \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mathbb{1}_{\boldsymbol{w'}}$ for some $\boldsymbol{w'} \in [\boldsymbol{w}]_R$ is also a derivation for $\boldsymbol{v'} \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mathbb{1}_{\boldsymbol{w''}}$ for some $\boldsymbol{w''} \in [\boldsymbol{w}]_R$. Therefore,

$$\begin{aligned} &|\{\mathcal{D} \in \Delta \mid \exists \boldsymbol{w'} \in [\boldsymbol{w}]_R \,.\, \boldsymbol{v} \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mathbb{1}_{\boldsymbol{w'}}\}| \leq \\ &|\{\mathcal{D} \in \Delta \mid \exists \boldsymbol{w''} \in [\boldsymbol{w}]_R \,.\, \boldsymbol{v'} \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mathbb{1}_{\boldsymbol{w''}}\}| \end{aligned}$$

Now, by symmetry we obtain the inequality in the other direction, and hence these two sets have the same cardinality. □

The next lemma states that if a CFP g_j rules a summand i, and i is enabled for some state vector $\boldsymbol{v} = (v_1, \ldots, v_j, \ldots, v_n)$ and local variable vector $\boldsymbol{d'_i} \in \boldsymbol{D_i}$, then the control flow graph of g_j contains an edge from v_j to $n_{i,j}(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e'_i})$ (where $\boldsymbol{e'_i} \in \boldsymbol{E_i}$ can be taken arbitrarily due to the uniqueness of the destination).

Lemma A.14. *Let g_j be a CFP, $\boldsymbol{v}$ a state vector, $\boldsymbol{d'_i} \in \boldsymbol{D_i}$ a local variable vector and $\boldsymbol{e'_i} \in \boldsymbol{E_i}$ a probabilistic choice. Then, if g_j rules i and $c_i(\boldsymbol{v}, \boldsymbol{d'_i})$ holds,* *it follows that*

$$(v_j, i, n_{i,j}(\boldsymbol{v}, \boldsymbol{d_i'}, \boldsymbol{e_i'})) \in E_{g_j}$$

Proof. Since g_j rules i, by definition $source(i, g_j) \neq \bot$. Then, by definition of *source*, $c_i(\boldsymbol{v}, \boldsymbol{d_i'})$ implies that $v_j = source(i, g_j)$. By definition of rules also $dest(i, g_j) \neq \bot$, and by the definition of *dest* we then know that $c_i(\boldsymbol{v}, \boldsymbol{d_i'})$ implies that $dest(i, g_j) = n_{i,j}(\boldsymbol{v}, \boldsymbol{d_i'}, \boldsymbol{e_i'})$. Thus, using the definition of the control flow graph, indeed $(v_j, i, n_{i,j}(\boldsymbol{v}, \boldsymbol{d_i'}, \boldsymbol{e_i'})) \in E_{g_j}$. □

Lemma A.15. *Let $\boldsymbol{v}$ and $\boldsymbol{v'}$ be state vectors such that $\boldsymbol{v} \cong \boldsymbol{v'}$, and assume that Relevant$(g_k, \boldsymbol{v'})$ for some g_k. Then, also Relevant$(g_k, \boldsymbol{v})$.*

Proof. If g_k is a CFP, then *Relevant*$(g_k, \boldsymbol{v})$ by definition. From now on we therefore assume that it is a DP.

Assume that $\boldsymbol{v} \cong \boldsymbol{v'}$ and *Relevant*$(g_k, \boldsymbol{v'})$. Let g_j be one of the CFPs g_k belongs to. Then, by definition of *Relevant*, we have $R(g_k, g_j, v_j')$. Because g_j is a CFP we know that *Relevant*$(g_j, \boldsymbol{v})$, so by the definition of $\cong$ we have $v_j = v_j'$. Since $R(g_k, g_j, v_j')$, this immediately implies $R(g_k, g_j, v_j)$. Since this argument holds for all g_j that g_k belongs to, we obtain *Relevant*$(g_k, \boldsymbol{v})$. □

Lemma A.16. *The relation $\cong$ is an equivalence relation.*

Proof. Reflexivity is trivial. For symmetry, assume that $\boldsymbol{v} \cong \boldsymbol{v'}$. For all $g_k \in J$, if *Relevant*$(g_k, \boldsymbol{v'})$, then by Lemma A.15 also *Relevant*$(g_k, \boldsymbol{v})$. Therefore, by definition of $\cong$ and the assumption that $\boldsymbol{v} \cong \boldsymbol{v'}$, we obtain $v_k = v_k'$, hence $\boldsymbol{v'} \cong \boldsymbol{v}$.

For transitivity, assume that $\boldsymbol{v} \cong \boldsymbol{v'}$ and $\boldsymbol{v'} \cong \boldsymbol{v''}$. If *Relevant*$(g_k, \boldsymbol{v})$, then by definition $v_k = v_k'$. Using symmetry and Lemma A.15 it follows that *Relevant*$(g_k, \boldsymbol{v'})$, and hence $v_k' = v_k''$. Therefore, $\boldsymbol{v} \cong \boldsymbol{v''}$. □

Now, we show that if a summand i is enabled given some state vector $\boldsymbol{v}$, then it is also enabled given a state vector $\boldsymbol{v'}$ such that $\boldsymbol{v} \cong \boldsymbol{v'}$.

Lemma A.17. *Let $\boldsymbol{v}$ and $\boldsymbol{v'}$ be state vectors such that $\boldsymbol{v} \cong \boldsymbol{v'}$, $i \in I$ a summand and $\boldsymbol{d_i'} \in \boldsymbol{D_i}$ a local variable vector for i. Then, $c_i(\boldsymbol{v}, \boldsymbol{d_i'})$ implies $c_i(\boldsymbol{v'}, \boldsymbol{d_i'})$.*

Proof. It has to be shown that for all $g_k \in \text{pars}(c_i)$ it holds that $v_k = v_k'$. Since this is trivially true for CFPs, we from now on assume that g_k is a DP. Assume that $c_i(\boldsymbol{v}, \boldsymbol{d_i'})$ holds. Let an arbitrary $g_k \in \text{pars}(c_i)$ be given, and let g_j be a CFP that g_k belongs to. Then, since g_k is directly used in i, by definition of belongs-to g_j rules i. Therefore, by Lemma A.14 and Definition 5.9, $v_j = source(i, g_j)$, and because g_k is used directly in i by definition $R(g_k, g_j, v_j)$. Since this holds for all g_j to which g_k belongs, we obtain *Relevant*$(g_k, \boldsymbol{v})$, and, using the definition of $\cong$, also $v_k = v_k'$. Since g_k was chosen arbitrary, this is the case for all $g_k \in \text{pars}(c_i)$, so $c_i(\boldsymbol{v'}, \boldsymbol{d_i'})$ also holds. □

We can also show that if a summand i is taken given some state vector $\boldsymbol{v}$, the resulting action parameters are identical to when i is taken given a state vector $\boldsymbol{v'}$ such that $\boldsymbol{v} \cong \boldsymbol{v'}$.

Lemma A.18. *Let $\boldsymbol{v}$ and $\boldsymbol{v}'$ be state vectors such that $\boldsymbol{v} \cong \boldsymbol{v}'$, $i \in I$ a summand and $\boldsymbol{d}'_{\boldsymbol{i}} \in \boldsymbol{D}_{\boldsymbol{i}}$ a local variable vector for i. Then, $\boldsymbol{b}_{\boldsymbol{i}}(\boldsymbol{v}, \boldsymbol{d}'_{\boldsymbol{i}}) = \boldsymbol{b}_{\boldsymbol{i}}(\boldsymbol{v}', \boldsymbol{d}'_{\boldsymbol{i}})$.*

Proof. Identical to the proof of Lemma A.17, when taking the elements of $\boldsymbol{b}_{\boldsymbol{i}}$ instead of c_i. □

Lemma A.19. *Let $\boldsymbol{v}$ and $\boldsymbol{v}'$ be state vectors such that $\boldsymbol{v} \cong \boldsymbol{v}'$, $i \in I$ a summand, $\boldsymbol{d}'_{\boldsymbol{i}} \in \boldsymbol{D}_{\boldsymbol{i}}$ a local variable vector for i and $\boldsymbol{e}'_{\boldsymbol{i}} \in \boldsymbol{E}_{\boldsymbol{i}}$ a probabilistic choice for i. Then, $f_i(\boldsymbol{v}, \boldsymbol{d}'_{\boldsymbol{i}}, \boldsymbol{e}'_{\boldsymbol{i}}) = f_i(\boldsymbol{v}', \boldsymbol{d}'_{\boldsymbol{i}}, \boldsymbol{e}'_{\boldsymbol{i}})$.*

Proof. Identical to the proof of Lemma A.17, when taking the elements of f_i instead of c_i. □

Finally, we show that taking a summand i given some state vector $\boldsymbol{v}$ and taking it given a state vector $\boldsymbol{v}'$ such that $\boldsymbol{v} \cong \boldsymbol{v}'$ yield next-state vectors that are equivalent with respect to $\cong$.

Lemma A.20. *Let $\boldsymbol{v}$ and $\boldsymbol{v}'$ be state vectors such that $\boldsymbol{v} \cong \boldsymbol{v}'$, $i \in I$ a summand, $\boldsymbol{d}'_{\boldsymbol{i}} \in \boldsymbol{D}_{\boldsymbol{i}}$ a local variable vector for i and $\boldsymbol{e}'_{\boldsymbol{i}} \in \boldsymbol{E}_{\boldsymbol{i}}$ a probabilistic choice for i. Then, $c_i(\boldsymbol{v}, \boldsymbol{d}'_{\boldsymbol{i}})$ implies that*

$$n_i(\boldsymbol{v}, \boldsymbol{d}'_{\boldsymbol{i}}, \boldsymbol{e}'_{\boldsymbol{i}}) \cong n_i(\boldsymbol{v}', \boldsymbol{d}'_{\boldsymbol{i}}, \boldsymbol{e}'_{\boldsymbol{i}})$$

Proof. By definition of $\cong$, it has to be shown that for all parameters g_k such that $\mathit{Relevant}(g_k, n_i(\boldsymbol{v}, \boldsymbol{d}'_{\boldsymbol{i}}, \boldsymbol{e}'_{\boldsymbol{i}}))$, it holds that $n_{i,k}(\boldsymbol{v}, \boldsymbol{d}'_{\boldsymbol{i}}, \boldsymbol{e}'_{\boldsymbol{i}}) = n_{i,k}(\boldsymbol{v}', \boldsymbol{d}'_{\boldsymbol{i}}, \boldsymbol{e}'_{\boldsymbol{i}})$. For CFPs this is immediate, since by definition these are either unchanged or have a unique destination in each summand; hence, from now on we assume that g_k is a DP.

To show that $n_{i,k}(\boldsymbol{v}, \boldsymbol{d}'_{\boldsymbol{i}}, \boldsymbol{e}'_{\boldsymbol{i}}) = n_{i,k}(\boldsymbol{v}', \boldsymbol{d}'_{\boldsymbol{i}}, \boldsymbol{e}'_{\boldsymbol{i}})$, we show that $v_m = v'_m$ for all parameters $g_m \in \text{pars}(n_{i,k})$. Since CFPs are always relevant they cannot differ between $\boldsymbol{v}$ and $\boldsymbol{v}'$, therefore we also assume that g_m is a DP from now on.

So, let $g_k, g_m \in \mathcal{D}$ such that $\mathit{Relevant}(g_k, n_i(\boldsymbol{v}, \boldsymbol{d}'_{\boldsymbol{i}}, \boldsymbol{e}'_{\boldsymbol{i}}))$ and $g_m \in \text{pars}(n_{i,k})$. Furthermore, let g_l be a CFP that g_m belongs to. Now, we distinguish between whether g_k belongs to g_l or not.

- Suppose that g_k belongs to g_l. Because $\mathit{Relevant}(g_k, n_i(\boldsymbol{v}, \boldsymbol{d}'_{\boldsymbol{i}}, \boldsymbol{e}'_{\boldsymbol{i}}))$ and g_k belongs to g_l, by definition of *Relevant* we have $R(g_k, g_l, n_{i,l}(\boldsymbol{v}, \boldsymbol{d}'_{\boldsymbol{i}}, \boldsymbol{e}'_{\boldsymbol{i}}))$. Now, if g_l rules g_i, then by Lemma A.14 (and the initial assumption that $c_i(\boldsymbol{v}, \boldsymbol{d}'_{\boldsymbol{i}})$ holds), we have $(v_l, i, n_{i,l}(\boldsymbol{v}, \boldsymbol{d}'_{\boldsymbol{i}}, \boldsymbol{e}'_{\boldsymbol{i}})) \in E_{g_l}$. Now it immediately follows from the second clause of the definition of relevance that $R(g_m, g_l, v_l)$.

  On the other hand, if g_l does not rule i, then by definition of belongs-to g_k is unchanged in i, so $n_{i,k} = g_k$. This implies that $g_m = g_k$. Since g_l does not rule i, but it is a CFP, it follows that $n_{i,l} = g_l$. So, $R(g_k, g_l, n_{i,l}(\boldsymbol{v}, \boldsymbol{d}'_{\boldsymbol{i}}, \boldsymbol{e}'_{\boldsymbol{i}})) = R(g_m, g_l, v_l)$, and therefore $R(g_m, g_l, v_l)$ follows trivially.
- Suppose that g_k does not belong to g_l. Since g_m does belong to g_l, $g_m \neq g_k$. So, i uses g_m (because g_m occurs in $\text{pars}(n_{i,k})$ and $g_m \neq g_k$), hence g_l rules i. Using Lemma A.14, we obtain $(v_l, i, n_{i,l}(\boldsymbol{v}, \boldsymbol{d}'_{\boldsymbol{i}}, \boldsymbol{e}'_{\boldsymbol{i}})) \in E_{g_l}$, which implies

that $v_l = source(i, g_l)$. Next, let g_p be some CFP that g_k belongs to. Then, by definition of *Relevant* and the fact that $Relevant(g_k, n_i(\boldsymbol{v}, \boldsymbol{d_i'}, \boldsymbol{e_i'}))$, we know that $R(g_k, g_p, n_{i,p}(\boldsymbol{v}, \boldsymbol{d_i'}, \boldsymbol{e_i'}))$. Because $g_m \neq g_k$ and $g_m \in \text{pars}(n_{i,k})$ we know that g_k is changed by i, and because g_k belongs to g_p it follows that g_p rules i. Applying Lemma A.14 again, $(v_p, i, n_{i,p}(\boldsymbol{v}, \boldsymbol{d_i'}, \boldsymbol{e_i'})) \in E_{g_p}$. Then, by the third clause of the definition of relevance, $R(g_m, g_l, v_l)$.

Since in all cases we have shown that $R(g_m, g_l, v_l)$ for an arbitrary g_l that g_m belongs to, by definition $Relevant(g_m, \boldsymbol{v})$. Therefore, since $\boldsymbol{v} \cong \boldsymbol{v}'$, by definition of $\cong$ it follows that $v_m = v_m'$. □

Theorem 5.21. *The relation $\cong$ is a derivation-preserving bisimulation.*

Proof. As we assumed the LPPE X to be decodable, derivation-preserving bisimulation can indeed be defined for its state vectors. Also, we already saw in Lemma A.16 that $\cong$ indeed is an equivalence relation.

Let $\boldsymbol{v}$ and $\boldsymbol{v}'$ be state vectors such that $\boldsymbol{v} \cong \boldsymbol{v}'$. We show that

$$\begin{aligned} c_i(\boldsymbol{v}, \boldsymbol{d_i'}) \implies c_i(\boldsymbol{v}', \boldsymbol{d_i'}) \wedge \boldsymbol{b_i}(\boldsymbol{v}, \boldsymbol{d_i'}) = \boldsymbol{b_i}(\boldsymbol{v}', \boldsymbol{d_i'}) \wedge \\ target_i(\boldsymbol{v}, \boldsymbol{d_i'}) \equiv_{\cong} target_i(\boldsymbol{v}', \boldsymbol{d_i'}) \end{aligned}$$

for every $i \in I$ and $\boldsymbol{d_i'} \in \boldsymbol{D_i}$. Then, by Lemma A.13 it follows that $\cong$ is a derivation-preserving bisimulation.

So, take an arbitrary $i \in I$ and $\boldsymbol{d_i'} \in \boldsymbol{D_i}$ and assume that $c_i(\boldsymbol{v}, \boldsymbol{d_i'})$. Now, by Lemma A.17 we know that $c_i(\boldsymbol{v}', \boldsymbol{d_i'})$ holds, and by Lemma A.18 that $\boldsymbol{b_i}(\boldsymbol{v}, \boldsymbol{d_i'}) = \boldsymbol{b_i}(\boldsymbol{v}', \boldsymbol{d_i'})$. Finally, Lemma A.20 tells us that

$$n_i(\boldsymbol{v}, \boldsymbol{d_i'}, \boldsymbol{e_i'}) \cong n_i(\boldsymbol{v}', \boldsymbol{d_i'}, \boldsymbol{e_i'})$$

for every $\boldsymbol{e_i'}$, and hence $target_i(\boldsymbol{v}, \boldsymbol{d_i'}) \equiv_{\cong} target_i(\boldsymbol{v}', \boldsymbol{d_i'})$ since additionally $f_i(\boldsymbol{v}, \boldsymbol{d_i'}, \boldsymbol{e_i'}) = f_i(\boldsymbol{v}', \boldsymbol{d_i'}, \boldsymbol{e_i'})$ for every $\boldsymbol{e_i'}$ (by Lemma A.19). □

A.3.3 Proof of Theorem 5.23

In the following lemmas, we use n_i' to refer to the transformed next state as given in Definition 5.22.

Lemma A.21. *For every summand $i \in I$, state vector $\boldsymbol{v}$ and local variable vector $\boldsymbol{d_i'} \in \boldsymbol{D_i}$ such that $c_i(\boldsymbol{v}, \boldsymbol{d_i'})$ holds, we have $n_i(\boldsymbol{v}, \boldsymbol{d_i'}, \boldsymbol{e_i'}) \cong n_i'(\boldsymbol{v}, \boldsymbol{d_i'}, \boldsymbol{e_i'})$ for every probabilistic choice $\boldsymbol{e_i'} \in \boldsymbol{E_i}$.*

Proof. To show that $n_i(\boldsymbol{v}, \boldsymbol{d_i'}, \boldsymbol{e_i'}) \cong n_i'(\boldsymbol{v}, \boldsymbol{d_i'}, \boldsymbol{e_i'})$, by definition of $\cong$ we need to show that for all parameters g_k such that $Relevant(g_k, n_i(\boldsymbol{v}, \boldsymbol{d_i'}, \boldsymbol{e_i'}))$, it holds that $n_{i,k}(\boldsymbol{v}, \boldsymbol{d_i'}, \boldsymbol{e_i'}) = n_{i,k}'(\boldsymbol{v}', \boldsymbol{d_i'}, \boldsymbol{e_i'})$. Assume such a $g_k \in J$. Then, by definition of *Relevant* we have

$$\bigwedge_{\substack{g_j \in \mathcal{C} \\ g_k \text{ belongs to } g_j}} R(g_k, g_j, n_{i,j}(\boldsymbol{v}, \boldsymbol{d_i'}, \boldsymbol{e_i'}))$$

and thus also

$$\bigwedge_{\substack{g_j \in \mathcal{C} \\ g_j \text{ rules } i \\ g_k \text{ belongs to } g_j}} R(g_k, g_j, n_{i,j}(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e'_i}))$$

By definition of n' and the fact that $dest(i, g_j) = n_{i,j}(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e'_i})$ when g_j rules i and $c_i(\boldsymbol{v}, \boldsymbol{d'_i})$ holds (as is the case), we obtain $n'_{i,k}(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e'_i}) = n_{i,k}(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e'_i})$. □

Lemma A.22. *Let $\approxeq$ be the smallest equivalence relation such that*

$$\boldsymbol{v} \cong \boldsymbol{v'} \implies X(\boldsymbol{v}) \approxeq X'(\boldsymbol{v'})$$

where the relation $\cong$ is used as it was defined for X. Then, $\approxeq$ is a derivation-preserving bisimulation.

Proof. In this proof, we use primed notations c'_i, n'_i and so on to refer to the elements of X'. It is easy to see that the reasoning behind Lemma A.13 still applies, except for the symmetric argument. It now suffices to show for an arbitrary pair of state vectors $\boldsymbol{v}, \boldsymbol{v'}$ such that $X(\boldsymbol{v}) \approxeq X'(\boldsymbol{v'})$, that both

$$\begin{aligned} c_i(\boldsymbol{v}, \boldsymbol{d'_i}) \implies & c'_i(\boldsymbol{v'}, \boldsymbol{d'_i}) \wedge \boldsymbol{b_i}(\boldsymbol{v}, \boldsymbol{d'_i}) = \boldsymbol{b'_i}(\boldsymbol{v'}, \boldsymbol{d'_i}) \wedge \\ & X(\mathit{target}_i(\boldsymbol{v}, \boldsymbol{d'_i})) \equiv_{\approxeq} X'(\mathit{target}'_i(\boldsymbol{v'}, \boldsymbol{d'_i})) \end{aligned}$$

and

$$\begin{aligned} c'_i(\boldsymbol{v}, \boldsymbol{d'_i}) \implies & c_i(\boldsymbol{v'}, \boldsymbol{d'_i}) \wedge \boldsymbol{b_i}(\boldsymbol{v}, \boldsymbol{d'_i}) = \boldsymbol{b'_i}(\boldsymbol{v'}, \boldsymbol{d'_i}) \wedge \\ & X(\mathit{target}_i(\boldsymbol{v}, \boldsymbol{d'_i})) \equiv_{\approxeq} X'(\mathit{target}'_i(\boldsymbol{v'}, \boldsymbol{d'_i})) \end{aligned}$$

for every $i \in I$ and $\boldsymbol{d'_i} \in \boldsymbol{D_i}$, where we use $X(\mathit{target}_i(\boldsymbol{v}, \boldsymbol{d'_i}))$ to denote the probability distribution μ such that $\mu(X(\boldsymbol{v'})) = \mathit{target}_i(\boldsymbol{v}, \boldsymbol{d'_i})(\boldsymbol{v'})$. We prove the first statement; the second follows by an almost-identical reasoning. Note that $X(\boldsymbol{v}) \approxeq X'(\boldsymbol{v'})$ implies $\boldsymbol{v} \cong \boldsymbol{v'}$, due to the fact that $\cong$ is an equivalence relation.

So, take an arbitrary $i \in I$ and $\boldsymbol{d'_i} \in \boldsymbol{D_i}$ and assume that $c_i(\boldsymbol{v}, \boldsymbol{d'_i})$. Now, by Lemma A.17 we know that $c_i(\boldsymbol{v'}, \boldsymbol{d'_i})$ holds, and since $c'_i = c_i$ also $c'_i(\boldsymbol{v'}, \boldsymbol{d'_i})$. Similarly, Lemma A.18 yields $\boldsymbol{b_i}(\boldsymbol{v}, \boldsymbol{d'_i}) = \boldsymbol{b_i}(\boldsymbol{v'}, \boldsymbol{d'_i})$, and due to $\boldsymbol{b'_i} = \boldsymbol{b_i}$ we find $\boldsymbol{b_i}(\boldsymbol{v}, \boldsymbol{d'_i}) = \boldsymbol{b'_i}(\boldsymbol{v'}, \boldsymbol{d'_i})$. Finally, Lemma A.20 yields $n_i(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e'_i}) \cong n_i(\boldsymbol{v'}, \boldsymbol{d'_i}, \boldsymbol{e'_i})$ for every $\boldsymbol{e'_i}$, and combining this with Lemma A.21 and the fact that $\cong$ is an equivalence relation, shows that

$$n_i(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e'_i}) \cong n'_i(\boldsymbol{v'}, \boldsymbol{d'_i}, \boldsymbol{e'_i})$$

Hence, since $f'_i = f_i$ and $f_i(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e'_i}) = f_i(\boldsymbol{v'}, \boldsymbol{d'_i}, \boldsymbol{e'_i})$ for every $\boldsymbol{e'_i}$ (as shown by Lemma A.19), also $\mathit{target}_i(\boldsymbol{v}, \boldsymbol{d'_i}) \equiv_{\cong} \mathit{target}'_i(\boldsymbol{v'}, \boldsymbol{d'_i})$, and by definition of $\approxeq$ and $\equiv$ this directly implies that $X(\mathit{target}_i(\boldsymbol{v}, \boldsymbol{d'_i})) \equiv_{\approxeq} X'(\mathit{target}'_i(\boldsymbol{v'}, \boldsymbol{d'_i}))$, completing the proof. □

Theorem 5.23. *Let X be a decodable LPPE, X' its transform, and $\boldsymbol{v}$ a state vector for X. Then, $X(\boldsymbol{v}) \sim_{\text{dp}} X'(\boldsymbol{v})$.*

Proof. Immediate from Lemma A.22. □

A.3.4 Proof of Proposition 5.24

Proposition 5.24. *For any state vector $\boldsymbol{v}$ reachable by the process $X'(\boldsymbol{init})$, invariably $\neg\,\mathit{Relevant}(g_k, \boldsymbol{v})$ implies that $v_k = \mathit{init}_k$.*

Proof. For the initial state the invariant is trivially true. Now assume that the invariant holds for an arbitrary g_k for some reachable state vector $\boldsymbol{v}$. We prove by induction that it still holds for all states reachable by an arbitrary summand i given a local state vector $\boldsymbol{d'_i} \in \boldsymbol{D_i}$ such that $c_i(\boldsymbol{v}, \boldsymbol{d'_i})$ holds and a probabilistic choice $\boldsymbol{e'_i} \in \boldsymbol{E_i}$. If

$$\bigwedge_{\substack{g_j \in \mathcal{C} \\ g_j \text{ rules } i \\ g_k \text{ belongs to } g_j}} R(g_k, g_j, \mathit{dest}(i, g_j)) \tag{A.7}$$

is false, then by definition we have $n'_{i,k} = \mathit{init}_k$ and the invariant holds. From now on we therefore assume that this conjunction is true, so $n'_{i,k} = n_{i,k}$ by definition. We make a case distinction between $\mathit{Relevant}(g_k, \boldsymbol{v})$ and $\neg\,\mathit{Relevant}(g_k, \boldsymbol{v})$, i.e., whether or not g_k is relevant in the current state.

- Assume that $\mathit{Relevant}(g_k, \boldsymbol{v})$, so by definition

  $$\bigwedge_{\substack{g_j \in \mathcal{C} \\ g_k \text{ belongs to } g_j}} R(g_k, g_j, v_j)$$

  For $g_j \in \mathcal{C}$ such that g_k belongs to g_j and g_j does not rule i, by definition g_j is unchanged in i, so from $R(g_k, g_j, v_j)$ it immediately follows that $R(g_k, g_j, n_{i,j}(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e'_i}))$. Now we easily obtain

  $$\bigwedge_{\substack{g_j \in \mathcal{C} \\ g_k \text{ belongs to } g_j}} R(g_k, g_j, n_{i,j}(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e'_i}))$$

  since we already assumed $R(g_k, g_j, \mathit{dest}(i, g_j))$ for all $g_j \in \mathcal{C}$ such that g_k belongs to g_j and g_j does rule i (and for those g_j by definition $\mathit{dest}(i, g_j) = n_{i,j}(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e'_i})$). Since CFPs do not belong to any CFPs, $n'_{i,j} = n_{i,j}$ for all $g_j \in \mathcal{C}$, and therefore

  $$\bigwedge_{\substack{g_j \in \mathcal{C} \\ g_k \text{ belongs to } g_j}} R(g_k, g_j, n'_{i,j}(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e'_i}))$$

  Now, by definition we have $\mathit{Relevant}(g_k, n'_i(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e'_i}))$, so the invariant holds.
- Assume that $\neg\,\mathit{Relevant}(g_k, \boldsymbol{v})$. If there exists a g_j such that g_k belongs to g_j and g_j does not rule i, then by definition of belongs-to g_k is unchanged in i. By the induction hypothesis $v_k = \mathit{init}_k$, and since $n'_{i,k} = n_{i,k}$ we find that

  $$n'_{i,k}(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e'_i}) = n_{i,k}(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e'_i}) = v_k = \mathit{init}_k$$

and the invariant holds.

From now on assume that all g_j that g_k belongs to rule i. Therefore, by the assumed truth of Equation (A.7) and the fact that $\mathit{dest}(i, g_j) = n_{i,j}(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e'_i})$ for all g_j that rule i, it follows that

$$\bigwedge_{\substack{g_j \in \mathcal{C} \\ g_k \text{ belongs to } g_j}} R(g_k, g_j, n_{i,j}(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e'_i}))$$

and, since $n'_{i,j} = n_{i,j}$ for all $g_j \in \mathcal{C}$, we obtain

$$\bigwedge_{\substack{g_j \in \mathcal{C} \\ g_k \text{ belongs to } g_j}} R(g_k, g_j, n'_{i,j}(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e'_i}))$$

Hence, by definition $\mathit{Relevant}(g_k, n'_i(\boldsymbol{v}, \boldsymbol{d'_i}, \boldsymbol{e'_i}))$ and the invariant holds. □

A.3.5 Proof of Theorem 5.25

Lemma A.23. *Let h be a function over state vectors, given for any $\boldsymbol{v}$ by*

$$h_k(\boldsymbol{v}) = \begin{cases} v_k & \textit{if Relevant}(g_k, \boldsymbol{v}) \\ \mathit{init}_k & \textit{otherwise} \end{cases}$$

and let H be the smallest equivalence relation H such that

$$h(\boldsymbol{v}) = \boldsymbol{v'} \implies (X(\boldsymbol{v}), X'(\boldsymbol{v'})) \in H$$

Then, H is a derivation-preserving bisimulation relating the reachable states of $X(\boldsymbol{init})$ and $X'(\boldsymbol{init})$.

Proof. Note that indeed $(X(\boldsymbol{init}), X'(\boldsymbol{init})) \in H$, since each element of a state vector is either left alone by h or reset to its initial value. Since all values of $\boldsymbol{init}$ are already equal to their initial value, we find that $h(\boldsymbol{init}) = \boldsymbol{init}$.

To see that H is a derivation-preserving bisimulation, we first note that Lemma A.22 already provided a derivation-preserving bisimulation $\approxeq$, relating all states $X(\boldsymbol{v})$ and $X'(\boldsymbol{v'})$ such that $\boldsymbol{v} \cong \boldsymbol{v'}$. To see how H and $\approxeq$ relate, we first show that $H \subseteq \approxeq$:

- If $(X(\boldsymbol{v}), X'(\boldsymbol{v'})) \in H$, then this must be due to $h(\boldsymbol{v}) = \boldsymbol{v'}$. Hence, by definition $\boldsymbol{v} \cong \boldsymbol{v'}$ and thus $X(\boldsymbol{v}) \approxeq X'(\boldsymbol{v'})$.
- If $(X'(\boldsymbol{v'}), X(\boldsymbol{v})) \in H$, then by symmetry of H also $(X(\boldsymbol{v}), X'(\boldsymbol{v'})) \in H$ and hence $X(\boldsymbol{v}) \approxeq X'(\boldsymbol{v'})$ as in the previous case. Applying symmetry of $\approxeq$, this yields $X'(\boldsymbol{v'}) \approxeq X(\boldsymbol{v})$.
- If $(X'(\boldsymbol{v'_1}), X'(\boldsymbol{v'_2})) \in H$, then it can only be the case that $\boldsymbol{v'_1} = \boldsymbol{v'_2}$ (since h is a function). Hence, $\boldsymbol{v'_1} \cong \boldsymbol{v'_2}$ and thus $X(\boldsymbol{v'_1}) \approxeq X'(\boldsymbol{v'_2})$. Since $\cong$ is an equivalence relation, also $\boldsymbol{v'_1} \cong \boldsymbol{v'_1}$ and thus $X(\boldsymbol{v'_1}) \approxeq X'(\boldsymbol{v'_1})$. By symmetry and transitivity of $\approxeq$, we obtain $X'(\boldsymbol{v'_1}) \approxeq X'(\boldsymbol{v'_2})$.

- If $(X(\boldsymbol{v_1}), X(\boldsymbol{v_2})) \in H$, then this must be due to $h(\boldsymbol{v_1}) = h(\boldsymbol{v_2})$. We know that $\boldsymbol{v_1} \cong h(\boldsymbol{v_1})$ and $\boldsymbol{v_2} \cong h(\boldsymbol{v_2})$. Using $h(\boldsymbol{v_1}) = h(\boldsymbol{v_2})$ and the fact that $\cong$ is an equivalence relation, this yields $\boldsymbol{v_1} \cong \boldsymbol{v_2}$ and thus $X(\boldsymbol{v_1}) \approxeq X'(\boldsymbol{v_2})$. Since $X(\boldsymbol{v_2}) \approxeq X'(\boldsymbol{v_2})$ by reflexivity of $\cong$, this yields $X(\boldsymbol{v_1}) \approxeq X(\boldsymbol{v_2})$.

So, indeed $H \subseteq \approxeq$.

Now, consider an arbitrary equivalence class C of $\approxeq$, given by

$$C = \{X(\boldsymbol{v_1}), X(\boldsymbol{v_2}), \ldots, X(\boldsymbol{v_n}), X'(\boldsymbol{w_1}), X'(\boldsymbol{w_2}), \ldots, X'(\boldsymbol{w_m})\}$$

As mentioned in the proof of Lemma A.22, $X(\boldsymbol{v}) \approxeq X'(\boldsymbol{v'})$ implies $\boldsymbol{v} \cong \boldsymbol{v'}$. Hence, $\boldsymbol{v_i} \cong \boldsymbol{w_j}$ for all i, j. Due to the fact that $\cong$ is an equivalence relation, this implies that also $\boldsymbol{v_i} \cong \boldsymbol{v_j}$ and $\boldsymbol{w_i} \cong \boldsymbol{w_j}$ for all i, j.

Let $\boldsymbol{w} = h(\boldsymbol{v_1})$. By definition of h and $\cong$, this implies that $\boldsymbol{v_1} \cong \boldsymbol{w}$. By definition of $\approxeq$, this in turn implies that $X(\boldsymbol{v_1}) \approxeq X'(\boldsymbol{w})$. Hence, it must be the case that $\boldsymbol{w_j} = \boldsymbol{w} = h(\boldsymbol{v_1})$ for some j. Without loss of generality, we assume that $\boldsymbol{w_1} = h(\boldsymbol{v_1})$. It is easy to see that $\boldsymbol{v} \cong \boldsymbol{v'}$ implies $h(\boldsymbol{v}) = h(\boldsymbol{v'})$. Hence, actually $\boldsymbol{w_1} = h(\boldsymbol{v_i})$ for all i. For all other states $\boldsymbol{w_j}$ with $j > 1$, we find $\boldsymbol{w_j} \neq h(\boldsymbol{v_i})$ for all i since each $\boldsymbol{v_i}$ can only be associated with one $h(\boldsymbol{v_i})$.

Hence, in H the above equivalence class will be split into the following classes:

$$\begin{aligned}
C_1 &= \{X(\boldsymbol{v_1}), X(\boldsymbol{v_2}), \ldots, X(\boldsymbol{v_n}), X'(\boldsymbol{w_1})\} \\
C_2 &= \{X'(\boldsymbol{w_2})\} \\
&\vdots \\
C_m &= \{X'(\boldsymbol{w_m})\}
\end{aligned}$$

Note that for each state vector $\boldsymbol{w_j}$ with $j > 1$, there also cannot be a state vector $\boldsymbol{z}$ different from any of the vectors $\boldsymbol{v_i}$ such that $h(\boldsymbol{z}) = \boldsymbol{w_j}$. After all, this would imply that $(X(\boldsymbol{z}), X'(\boldsymbol{w_j})) \in H$, while $X(\boldsymbol{z}) \not\approxeq X'(\boldsymbol{w_j})$ and we already showed that $H \subseteq \approxeq$. As we took an arbitrary equivalence class, the same splitting as above happens for all equivalence classes of $\approxeq$.

We first show that H is a strong bisimulation. Let $\boldsymbol{v}$ and $\boldsymbol{v'}$ be state vectors such that $h(\boldsymbol{v}) = \boldsymbol{v'}$. Assuming $X(\boldsymbol{v}) \xrightarrow{a} \mu$, we show that there exists a transition $X'(\boldsymbol{v'}) \xrightarrow{a} \nu$ such that $\mu \equiv_H \nu$ (the other direction can be shown symmetrically). Since $\approxeq$ is a strong bisimulation relation and $h(\boldsymbol{v}) = \boldsymbol{v'}$ implies $\boldsymbol{v} \cong \boldsymbol{v'}$ and hence $X(\boldsymbol{v}) \approxeq X'(\boldsymbol{v'})$, we already know that there exists a transition $X'(\boldsymbol{v'}) \xrightarrow{a} \nu$ such that $\mu \equiv_{\approxeq} \nu$. It remains to show that also $\mu \equiv_H \nu$.

So, take an arbitrary equivalence class of H. It will be like one of the equivalence classes C_i discussed above. Without loss of generality, we assume that $m = 2$. We find $\mu(C_2) = 0$, since it only contains an X'-process. Now, we show by contradiction that also $\nu(C_2) = 0$, so assume that $\nu(C_2) \neq 0$, and hence $\nu(X'(\boldsymbol{w_2})) > 0$. Since $\mu \equiv_{\approxeq} \nu$, there must be at least one state $\boldsymbol{u}$ such that $\mu(X(\boldsymbol{u})) > 0$ and $X(\boldsymbol{u}) \approxeq X'(\boldsymbol{w_2})$, and hence $\boldsymbol{u} \cong \boldsymbol{w_2}$. Since Proposition 5.24 tells us that every reachable state of X' has all its irrelevant variables reset (and we do not care about unreachable states), this implies that $h(\boldsymbol{u}) = \boldsymbol{w_2}$. This is a contradiction, since we already concluded that there is no state $\boldsymbol{z}$

such that $h(\boldsymbol{z}) = \boldsymbol{w_2}$. So, $\nu(C_2) = 0$. Since $\mu(C) = \nu(C)$ due to $\mu \equiv_{\approxeq} \nu$, and $\mu(C_2) = \nu(C_2)$, the fact that $C = C_1 \cup C_2$ yield $\mu(C_1) = \nu(C_1)$.

As the argument above can be repeated for all equivalence classes of H, we can indeed conclude that $\mu \equiv_H \nu$.

Finally, to see why H is derivation preserving, note that $\approxeq$ was already shown to be derivation preserving. Also, recall that we showed above that H and $\approxeq$ are very similar; the only difference is that some equivalence classes of H are a bit smaller due to some unreachable states having been separated. Since these new equivalence classes are all singletons, nothing about derivation preservation has to be proven for the states that they contain. For the equivalence classes such as C_1 above, all states have the same number of derivations to for instance an equivalence class D_1 of H as to the equivalence class D of $\approxeq$ (using the same naming convention as above). Hence, since $\approxeq$ was derivation preserving, so is H. □

Theorem 5.25. *The number of reachable states of $X'(\boldsymbol{init})$ is at most the number of reachable states of $X(\boldsymbol{init})$.*

Proof. The bisimulation relation H provided by Lemma A.23 relates each state of X to precisely one state of X' (due to its functional aspect). Hence, the number of reachable states of X' cannot be larger than the number of reachable states of X, otherwise some reachable states of X' would be unrelated to X and hence H could not have been a bisimulation relation. □

A.4 Proofs for Chapter 6

In the proofs for this chapter, whenever a confluent set $\mathcal{T}$ is given, we abuse notation by writing *confluent transition* to denote a transition in this set $\mathcal{T}$. Note that, in general, there may also be confluent transitions that are not in $\mathcal{T}$.

A.4.1 Proof of Proposition 6.10

Proposition 6.10. *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L\rangle$ be an MA, $P \subseteq \mathscr{P}(\hookrightarrow)$ a confluence classification for $\mathcal{M}$ and $\mathcal{T}$ a Markovian confluent set for P. Then,*

$$s \twoheadrightarrow\twoheadleftarrow_{\mathcal{T}} t \qquad \textit{if and only if} \qquad s \twoheadleftarrow\!\!\!\twoheadrightarrow_{\mathcal{T}} t$$

Proof. We separately prove both directions of the equivalence.

($\Longrightarrow$) Let $s \twoheadrightarrow\twoheadleftarrow_{\mathcal{T}} t$. Then, by definition there is a state u such that $s \twoheadrightarrow_{\mathcal{T}} u$ and $t \twoheadrightarrow_{\mathcal{T}} u$. This immediately implies that $s \twoheadleftarrow\!\!\!\twoheadrightarrow_{\mathcal{T}} t$.

($\Longleftarrow$) Let $s \twoheadleftarrow\!\!\!\twoheadrightarrow_{\mathcal{T}} t$. This means that there is a path from s to t such as

$$s_0 \leftarrow s_1 \rightarrow s_2 \rightarrow s_3 \leftarrow s_4 \leftarrow s_5 \rightarrow s_6,$$

where $s_0 = s$, $s_6 = t$ and each of the transitions is in $\mathcal{T}$. Note that $s_i \twoheadrightarrow\twoheadleftarrow_{\mathcal{T}} s_{i+1}$ for all s_i, s_{i+1} on this path. After all, if $s_i \rightarrow s_{i+i}$ then they can join at s_{i+1}, otherwise they can join at s_i. Hence, to show that $s \twoheadrightarrow\twoheadleftarrow_{\mathcal{T}} t$, it suffices to show that $\twoheadrightarrow\twoheadleftarrow_{\mathcal{T}}$ is transitive.

Let $s' \twoheadrightarrow\twoheadleftarrow_{\mathcal{T}} s$ and $s \twoheadrightarrow\twoheadleftarrow_{\mathcal{T}} s''$. We show that $s' \twoheadrightarrow\twoheadleftarrow_{\mathcal{T}} s''$. Let t' be a state such that $s \twoheadrightarrow_{\mathcal{T}} t'$ and $s' \twoheadrightarrow_{\mathcal{T}} t'$, and likewise, let t'' be a similar state for s and s''. If we can show that there is some state t such that $t' \twoheadrightarrow_{\mathcal{T}} t$ and $t'' \twoheadrightarrow_{\mathcal{T}} t$, we have the result. Let a minimal confluent path from s to t' be given by $s_0 \rightarrow_{\mathcal{T}} s_1 \rightarrow_{\mathcal{T}} \cdots \rightarrow_{\mathcal{T}} s_n$, with $s_0 = s$ and $s_n = t'$. By induction on the length of this path, we show that for each state s_i on it, there is some state t such that $s_i \twoheadrightarrow_{\mathcal{T}} t$ and $t'' \twoheadrightarrow_{\mathcal{T}} t$. Since t' is also on the path, this completes the argument.

Base case. There clearly is a state t such that $s_0 \twoheadrightarrow_{\mathcal{T}} t$ and $t'' \twoheadrightarrow_{\mathcal{T}} t$, namely t'' itself. After all, $s_0 = s$ and $s \twoheadrightarrow_{\mathcal{T}} t''$, and $\twoheadrightarrow_{\mathcal{T}}$ is reflexive.

Inductive case. Let there be a state t_k such that $s_k \twoheadrightarrow_{\mathcal{T}} t_k$ and $t'' \twoheadrightarrow_{\mathcal{T}} t_k$. We show that there exists a state t_{k+1} such that $s_{k+1} \twoheadrightarrow_{\mathcal{T}} t_{k+1}$ and $t'' \twoheadrightarrow_{\mathcal{T}} t_{k+1}$. Let $s_k \xrightarrow{\tau} u$ be the first transition on the $\mathcal{T}$-path from s_k to t_k. Let $s_k \xrightarrow{\tau} s_{k+1}$ be the $\mathcal{T}$-transition between s_k and s_{k+1}. Since it is in $\mathcal{T}$, there must be at least one group $C \in P \cap \mathcal{T}$ such that $s_k \xrightarrow{\tau}_C s_{k+1}$.

By definition of confluence, since $(s_k \xrightarrow{\tau} u) \in \mathcal{T}$ and $s_k \xrightarrow{\tau}_C s_{k+1}$ for some $C \in P$, either (1) $s_{k+1} = u$ (the transitions coincide), or (2) there is a transition $u \xrightarrow{\tau}_C u'$ such that $\mathbb{1}_{s_{k+1}} \equiv_R \mathbb{1}_{u'}$, with R the equivalence relation given in Definition 6.7.

In case (1), we directly find $s_{k+1} \twoheadrightarrow_{\mathcal{T}} t_k$. Hence, we can just take $t_{k+1} = t_k$. In case (2), either $s_{k+1} = u'$ or $s_{k+1} \xrightarrow{\tau}_{\mathcal{T}} u'$. In both cases, if $u = t_k$, we can take $t_{k+1} = u'$ and indeed $s_{k+1} \twoheadrightarrow_{\mathcal{T}} t_{k+1}$ and $t'' \twoheadrightarrow_{\mathcal{T}} t_{k+1}$. Otherwise, we can use the same reasoning to show that there is a state t_{k+1} such that $u' \twoheadrightarrow_{\mathcal{T}} t_{k+1}$ and $t'' \twoheadrightarrow_{\mathcal{T}} t_{k+1}$, based on $u \twoheadrightarrow_{\mathcal{T}} t_k$, $t'' \twoheadrightarrow_{\mathcal{T}} t_k$ and $u \xrightarrow{\tau}_{\mathcal{T}} u'$. Since the path from u to t_k is one transition shorter than the path from s_k to t_k, this argument terminates. □

A.4.2 Proof of Theorem 6.11

Theorem 6.11. *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L\rangle$ be an MA, $P \subseteq \mathscr{P}(\hookrightarrow)$ a confluence classification for $\mathcal{M}$ and $\mathcal{T}_1, \mathcal{T}_2$ two Markovian confluent sets for P. Then, $\mathcal{T}_1 \cup \mathcal{T}_2$ is also a Markovian confluent set for P.*

Proof. Let $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$. Clearly, $\mathcal{T}$ still only contain invisible transitions with Dirac distributions, since $\mathcal{T}_1$ and $\mathcal{T}_2$ do. Consider a transition $(s \xrightarrow{\tau}_{\mathcal{T}} t)$, and another transition $s \xrightarrow{a} \mu$. We need to show that

$$\begin{cases} \forall C \in P \,.\, s \xrightarrow{a}_C \mu \implies \exists \nu \in \mathrm{Distr}(S) \,.\, t \xrightarrow{a}_C \nu \wedge \mu \equiv_R \nu & \text{,if } (s \xrightarrow{a} \mu) \in P \\ \qquad\qquad\qquad\qquad\quad \exists \nu \in \mathrm{Distr}(S) \,.\, t \xrightarrow{a} \nu \ \ \wedge \mu \equiv_R \nu & \text{,if } (s \xrightarrow{a} \mu) \notin P \end{cases}$$

where R is the smallest equivalence relation such that

$$R \supseteq \{(s,t) \in \mathit{supp}(\mu) \times \mathit{supp}(\nu) \mid (s \xrightarrow{\tau} t) \in \mathcal{T}\}.$$

Without loss of generality, assume that $s \xrightarrow{\tau}_{\mathcal{T}_1} t$. Hence, by definition of

Markovian confluence, we find that

$$\begin{cases} \forall C \in P \,.\, s \xrightarrow{a}_C \mu \implies \exists \nu \in \text{Distr}(S) \,.\, t \xrightarrow{a}_C \nu \wedge \mu \equiv_{R_1} \nu & \text{, if } (s \xrightarrow{a} \mu) \in P \\ \exists \nu \in \text{Distr}(S) \,.\, t \xrightarrow{a} \nu \;\; \wedge \mu \equiv_{R_1} \nu & \text{, if } (s \xrightarrow{a} \mu) \notin P \end{cases}$$

where R_1 is the smallest equivalence relation such that

$$R_1 \supseteq \{(s,t) \in \mathit{supp}(\mu) \times \mathit{supp}(\nu) \mid (s \xrightarrow{\tau} t) \in \mathcal{T}_1\}$$

Note that $R \supseteq R_1$ since $\mathcal{T} \supseteq \mathcal{T}_1$. Therefore, $\mu \equiv_{R_1} \nu$ implies $\mu \equiv_R \nu$ (using Proposition 5.2.1.5 from [Sto02a]). The result now immediately follows. □

A.4.3 Proof of Theorem 6.13

Lemma A.24. *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \text{AP}, L\rangle$ be an MA, $s, s' \in S$ two of its states, $a \in A$, $\mu \in \text{Distr}(S)$, $P \subseteq \mathscr{P}(\hookrightarrow)$ a confluence classification for $\mathcal{M}$ and $\mathcal{T}$ a Markovian confluent set for P. Then,*

$$s \twoheadrightarrow_{\mathcal{T}} s' \wedge s \xrightarrow{a} \mu \implies (a = \tau \wedge \mu \equiv_R \mathbb{1}_{s'}) \vee \big(\exists \nu \in \text{Distr}(S) \,.\, s' \xrightarrow{a} \nu \wedge \mu \equiv_R \nu\big)$$

where $R = \{(u,v) \mid u \twoheadrightarrow\twoheadleftarrow_{\mathcal{T}} v\}$.

Proof. First of all, we note that R is indeed an equivalence relation, as shown in Proposition 6.10.

Let $s, s' \in S$ be such that $s \twoheadrightarrow_{\mathcal{T}} s'$, and assume a transition $s \xrightarrow{a} \mu$. Let $R = \{(u,v) \mid u \twoheadrightarrow\twoheadleftarrow_{\mathcal{T}} v\}$. We show that either $a = \tau \wedge \mu \equiv_R \mathbb{1}_{s'}$ or that there exists a transition $s' \xrightarrow{a} \nu$ such that $\mu \equiv_R \nu$, by induction on the length of the confluent path from s to s'. Let $s_0 \xrightarrow{\tau}_{\mathcal{T}} s_1 \xrightarrow{\tau}_{\mathcal{T}} \dots \xrightarrow{\tau}_{\mathcal{T}} s_{n-1} \xrightarrow{\tau}_{\mathcal{T}} s_n$, with $s_0 = s$ and $s_n = s'$, denote this path. Then, we show that

$$(a = \tau \wedge \mu \equiv_R \mathbb{1}_{s'}) \vee \big(\exists \nu \in \text{Distr}(S) \,.\, s_i \xrightarrow{a} \nu \wedge \mu \equiv_R \nu\big)$$

holds for every state s_i on this path. For the base case s this is immediate, since $s \xrightarrow{a} \mu$ and the relation $\equiv_R$ is reflexive.

As induction hypothesis, assume that the formula holds for some state s_i $(0 \leq i < n)$. We show that it still holds for state s_{i+1}. If the above formula was true for s_i due to the clause $a = \tau \wedge \mu \equiv_R \mathbb{1}_{s'}$, then this still holds for s_{i+1}. So, assume that $s_i \xrightarrow{a} \nu$ such that $\mu \equiv_R \nu$.

Since $s_i \xrightarrow{\tau}_{\mathcal{T}} s_{i+1}$ and $s_i \xrightarrow{a} \nu$, by definition of confluence either (1) $a = \tau$ and $\nu = \mathbb{1}_{s_{i+1}}$, or (2) there is a transition $s_{i+1} \xrightarrow{a} \nu'$ such that $\nu \equiv_{R'} \nu'$, where R' is the smallest equivalence relation such that

$$R' \supseteq \{(s,t) \in \mathit{supp}(\nu) \times \mathit{supp}(\nu') \mid (s \xrightarrow{\tau} t) \in \mathcal{T}\}$$

(1) In the first case, $\nu = \mathbb{1}_{s_{i+1}}$ implies that $\nu \equiv_R \mathbb{1}_{s'}$ as there is a $\mathcal{T}$-path from s_{i+1} to s' and hence $(s_{i+1}, s') \in R$. Since we assumed that $\mu \equiv_R \nu$, and the relation $\equiv_R$ is transitive, this yields $\mu \equiv_R \mathbb{1}_{s'}$. Together with $a = \tau$, this completes the proof.

(2) In the second case, note that $R \supseteq R'$. After all, $R = \twoheadrightarrow\twoheadleftarrow_{\mathcal{T}} = \twoheadleftarrow\!\!\twoheadrightarrow_{\mathcal{T}}$ (by Proposition 6.10), and obviously $(s,t) \in R'$ implies that $s \twoheadleftarrow\!\!\twoheadrightarrow_{\mathcal{T}} t$. Hence, $\nu \equiv_{R'} \nu'$ implies $\nu \equiv_R \nu'$ (using Proposition 5.2.1.5 from [Sto02a]). Since we assumed that $\mu \equiv_R \nu$, by transitivity of $\equiv_R$ we obtain $\mu \equiv_R \nu'$. Hence, there is a transition $s_{i+1} \xrightarrow{a} \nu'$ such that $\mu \equiv_R \nu'$, which completes the proof. □

Theorem 6.13. *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L\rangle$ be an MA, $s, s' \in S$ two of its states, $P \subseteq \mathscr{P}(\hookrightarrow)$ a confluence classification for $\mathcal{M}$ and $\mathcal{T}$ a Markovian confluent set for P. Then,*

$$s \twoheadleftarrow\!\!\twoheadrightarrow_{\mathcal{T}} s' \textit{ implies } s \approx_{\mathrm{b}}^{\mathrm{div}} s'$$

Proof. We show that $s \twoheadrightarrow\twoheadleftarrow_{\mathcal{T}} s'$ implies $s \approx_{\mathrm{b}}^{\mathrm{div}} s'$. By Proposition 6.10, this is equivalent to the theorem. So, assume that $s \twoheadrightarrow\twoheadleftarrow_{\mathcal{T}} s'$. To show that $s \approx_{\mathrm{b}}^{\mathrm{div}} s'$, consider the relation

$$R = \{(u,v) \mid u \twoheadrightarrow\twoheadleftarrow_{\mathcal{T}} v\}$$

Clearly $(s,s') \in R$, and from Proposition 6.10 and the obvious fact that $\twoheadleftarrow\!\!\twoheadrightarrow_{\mathcal{T}}$ is an equivalence relation, it follows that R is an equivalence relation as well.

It remains to show that R is a divergence-sensitive branching bisimulation. First, $(p,q) \in R$ indeed implies $L(p) = L(q)$ by the requirements that confluent transitions are invisible (Definition 6.7). Second, let $(p,q) \in R$, i.e., $p \twoheadrightarrow\twoheadleftarrow_{\mathcal{T}} q$. We need to show that for every extended transition $p \xrightarrow{a} \mu$ there is a transition $q \overset{a}{\Longrightarrow}_R \mu'$ such that $\mu \equiv_R \mu'$.

So, assume such a transition $p \xrightarrow{a} \mu$. Let r be a state such that $p \twoheadrightarrow_{\mathcal{T}} r$ and $q \twoheadrightarrow_{\mathcal{T}} r$. By Lemma A.24, either (1) $a = \tau \wedge \mu \equiv_R \mathbb{1}_r$ or (2) there is a distribution $\nu \in \mathrm{Distr}(S)$ such that $r \xrightarrow{a} \nu \wedge \mu \equiv_R \nu$.

(1) In the first case, note that $q \twoheadrightarrow_{\mathcal{T}} r$ immediately implies that $q \overset{\tau}{\Longrightarrow}_R \mathbb{1}_r$. After all, we can schedule the (invisible) confluent transitions from q to r and then terminate. Indeed, all intermediate states are clearly related by R. Together with the assumption that $\mu \equiv_R \mathbb{1}_r$, this completes the argument.

(2) In the second case, note that $q \twoheadrightarrow_{\mathcal{T}} r$ and $r \xrightarrow{a} \nu$ together immediately imply that $q \overset{a}{\Longrightarrow}_R \nu$. After all, we can schedule the (invisible) confluent transitions from q to r, perform the transition $r \xrightarrow{a} \nu$ and then terminate. Indeed, all intermediate states before the a-transition are clearly related by R. Together with the assumption that $\mu \equiv_R \nu$, this completes the argument.

It remains to show that R is divergence sensitive. So, let $(s,s') \in R$ (and hence $s \twoheadrightarrow\twoheadleftarrow_{\mathcal{T}} s'$) and assume that there is a scheduler $\mathcal{S}$ such that

$$\forall \pi \in \mathit{finpaths}_{\mathcal{M}}^{\mathcal{S}}(s) \;.\; A_{\mathrm{red}}(\pi) \wedge \mathcal{S}(\pi)(\bot) = 0$$

It is well known that we can assume that such diverging schedulers are memoryless and deterministic.

We show that there also is a diverging scheduler from s'. First, note that since $s \twoheadrightarrow\twoheadleftarrow_{\mathcal{T}} s'$, there is a state t such that $s \twoheadrightarrow_{\mathcal{T}} t$ and $s' \twoheadrightarrow_{\mathcal{T}} t$. We show that there is a diverging scheduler from t; then, the result follows as from s' we can schedule to first follow the confluent (and hence invisible) transitions to t and then continue with the diverging scheduler from t.

Let $s_0 \xrightarrow{\tau}_{\mathcal{T}} s_1 \xrightarrow{\tau}_{\mathcal{T}} s_2 \xrightarrow{\tau}_{\mathcal{T}} \dots \xrightarrow{\tau}_{\mathcal{T}} s_n$ be the confluent path from s to t; hence, $s_0 = s$ and $s_n = t$. It might be the case that some states on this path also occur on the tree associated with $\mathcal{S}$; hence, for those states a diverging scheduler already exists. Let s_i be the last state on the path from s_0 to s_n that occurs on the tree of $\mathcal{S}$. We show that s_n also has a diverging scheduler by induction on the length of the path from s_i to s_n; note that the base case is immediate.

Assume that s_j (with $i \leq j < n$) has a diverging scheduler $\mathcal{S}'$. We show that s_{j+1} has one too. If s_{j+1} occurs on the tree associated with $\mathcal{S}'$ this is immediate, so from now on assume that it does not. From s_j there now is a confluent transition $s_j \xrightarrow{\tau}_{\mathcal{T}} s_{j+1}$ and an invisible (not necessarily confluent) transition $s_j \xrightarrow{\tau} \mu$ (chosen by $\mathcal{S}'$ as first step of the diverging path form s_j). By definition of confluence, either these transitions coincide or there is a transition $s_{j+1} \xrightarrow{\tau} \nu$ such that $\mu \equiv_{R'} \nu$, with R' the smallest equivalence relation such that $R \supseteq \{(s,t) \in supp(\mu) \times supp(\nu) \mid (s \xrightarrow{\tau} t) \in \mathcal{T}\}$. The first option is impossible, since we assumed that s_{j+1} is not on the tree associated with $\mathcal{S}'$. Therefore, there is a transition $s_{j+1} \xrightarrow{\tau} \nu$ such that $\mu \equiv_{R'} \nu$.

Since $s_j \xrightarrow{\tau} \mu$ is invisible, $L(u) = L(s_j)$ for all states $u \in supp(\mu)$. Additionally, by definition of R', $\mu \equiv_{R'} \nu$ implies that each state $v \in supp(\nu)$ is either (1) in $supp(\mu)$ as well or (2) has an incoming confluent transition $u \xrightarrow{\tau}_{\mathcal{T}} v$ with $u \in supp(\mu)$. Hence, since confluent transitions are invisible, also $L(v) = L(s_j)$ for all states $v \in supp(\nu)$. Finally, since $s_j \xrightarrow{\tau} s_{j+1}$ is confluent, also $L(s_j) = L(s_{j+1})$. Together, these facts imply that $L(v) = L(s_{j+1})$ for all states $v \in supp(\nu)$, and hence that $s_{j+1} \xrightarrow{\tau} \nu$ is invisible. We schedule this transition from s_{j+1} in order to diverge. So, we still need to show that it is possible to diverge from all states $q \in supp(\nu)$.

By definition of R', $\mu \equiv_{R'} \nu$ implies that each state $v \in supp(\nu)$ is either (1) in $supp(\mu)$ as well or (2) has an incoming confluent transition $u \xrightarrow{\tau}_{\mathcal{T}} v$ with $u \in supp(\mu)$. In the first case, we can diverge from v using $\mathcal{S}'$. In the second case, we have reached the situation of a state v with an incoming confluent transition from a state u that has a diverging scheduler. Now, the above reasoning can be applied again, taking $s_j = u$ and $s_{j+1} = v$. Either at some point overlap with the scheduler of u occurs, or this argument is repeated indefinitely; in both cases, divergence is obtained. □

A.4.4 Proof of Proposition 6.18

Theorem 6.18. *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow, \mathrm{AP}, L\rangle$ be an MA, $\mathcal{T}$ a Markovian confluent set for $\mathcal{M}$, and $\varphi\colon S \to S$ a representation map for $\mathcal{M}$ under $\mathcal{T}$. Then,*

$$\mathcal{M}/\varphi \approx_{\mathrm{b}}^{\mathrm{div}} \mathcal{M}$$

Proof. We denote the extended transition relation of $\mathcal{M}$ by $\to$, and the one of

$\mathcal{M}/\varphi$ by $\rightarrow_\varphi$. We take the disjoint union $\mathcal{M}'$ of $\mathcal{M}$ and $\mathcal{M}/\varphi$, to provide a bisimulation relation over this state space that contains their initial states. We denote the transition relation of $\mathcal{M}'$ by $\rightarrow'$. Note that, based on whether $s \in \mathcal{M}$ or $s \in \mathcal{M}/\varphi$, a transition $s \xrightarrow{a}' \mu$ corresponds to either $s \xrightarrow{a} \mu$ or $s \xrightarrow{a}_\varphi \mu$.

To distinguish between for instance a state $\varphi(s) \in \mathcal{M}$ and the corresponding state $\varphi(s) \in \mathcal{M}/\varphi$, we denote all states $s, \varphi(s)$ from $\mathcal{M}$ just by $s, \varphi(s)$, and all states $s, \varphi(s)$ from $\mathcal{M}/\varphi$ by $\hat{s}, \hat{\varphi}(s)$.

Let R be the smallest equivalence relation containing the set

$$\{(s, \hat{\varphi}(s)) \mid s \in S\},$$

i.e., R relates all states from $\mathcal{M}$ that have the same representative to each other and to this mutual representative from $\mathcal{M}/\varphi$. Clearly, $(s^0, \hat{\varphi}(s^0)) \in R$. Also, $(p, q) \in R$ implies $L(p) = L(q)$ by definition of the representation map (prescribing all states that have the same representative to be connected by confluent transitions), the fact that confluent transitions are invisible and the construction of the quotient (leaving the state labelling invariant).

Note that given this equivalence relation R, for every probability distribution μ we have $\mu \equiv_R \mu_\varphi$ (no matter whether μ_φ is in $\mathcal{M}$ or in $\mathcal{M}/\varphi$). After all, the lifting over φ just changes the states in the support of μ to their representatives; as R relates precisely such states, clearly $\mu \equiv_R \mu_\varphi$. This observation is used several times in the proof below.

Now, let $(s, s') \in R$ and assume that there is an extended transition $s \xrightarrow{a}' \mu$. We show that also $s' \overset{a}{\Longrightarrow}'_R \mu'$ such that $\mu \equiv_R \mu'$. Note that there are four possible cases to consider with respect to the origin of s and s', indicated by the presence or absence of hats:

- Case 1: $(\hat{s}, \hat{s}')$. Since every equivalence class of R contains precisely one representative from $\mathcal{M}/\varphi$, we find that $\hat{s} = \hat{s}'$. Hence, the result follows directly by the scheduler that takes the transition $s \xrightarrow{a}' \mu$ and then terminates.

- Case 2: (s, s'). If both states are in $\mathcal{M}$, then the quotient is not involved and $\varphi(s) = \varphi(s')$. By definition of the representation map, we find $s \twoheadrightarrow\twoheadleftarrow_{\mathcal{T}} s'$. Using Theorem 6.13, this immediately implies that $s' \overset{a}{\Longrightarrow}_{R'} \mu'$ such that $\mu \equiv_{R'} \mu'$ for $R' = \{(u, v) \mid u \twoheadrightarrow\twoheadleftarrow_{\mathcal{T}} v\}$. Since all states connected by $\mathcal{T}$-transitions are required to have the same representative, we have $R \supseteq R'$. Hence, also $s' \overset{a}{\Longrightarrow}_R \mu'$, as this is then less restrictive. Moreover, $\mu \equiv_R \mu'$ by Proposition 5.2.1.5 from [Sto02a]. Finally, note that $s' \overset{a}{\Longrightarrow}_R \mu'$ implies $s' \overset{a}{\Longrightarrow}'_R \mu'$.

- Case 3: $(\hat{s}, s')$. Since $\hat{s}$ is in $\mathcal{M}/\varphi$ and s' is not, by definition of R we find that $\hat{s} = \hat{\varphi}(s')$. Hence, by assumption $\hat{\varphi}(s') \xrightarrow{a}_\varphi \mu$, and thus by definition of the extended arrow either (1) $a \in A$ and $\hat{\varphi}(s') \overset{a}{\hookrightarrow}_\varphi \mu$, or (2) $a = \chi(\lambda)$ for $\lambda = \mathit{rate}(\hat{\varphi}(s'))$, $\lambda > 0$, $\mu = \mathbb{P}_{\hat{\varphi}(s')}$ and there is no μ' such that $\hat{\varphi}(s') \overset{\tau}{\hookrightarrow}_\varphi \mu'$. We make a case distinction based on this.

  (1) Let $a \in A$ and $\hat{\varphi}(s') \overset{a}{\hookrightarrow}_\varphi \mu$. By definition of the quotient, this implies that there is a transition $\varphi(s') \overset{a}{\hookrightarrow} \mu'$ in $\mathcal{M}$ such that $\mu = \mu'_\varphi$. By

definition of the representation map, there is a $\mathcal{T}$-path (which is invisible and deterministic) from s' to $\varphi(s')$ in $\mathcal{M}$. Hence, $s' \xRightarrow{a}_R \mu'$ (and therefore also $s' \xRightarrow{a}{}'_R \mu'$) by the scheduler from s' that first goes to $\varphi(s')$ and then executes the $\varphi(s') \xhookrightarrow{a} \mu'$ transition. Note that the transition is indeed branching, as all steps in between have the same representative and thus are related by R.

It remains to show that $\mu \equiv_R \mu'$. We already saw that $\mu = \mu'_\varphi$; hence, the result follows from the observation that $\mu \equiv_R \mu_\varphi$ for every μ.

(2) Let $a = \chi(\lambda)$ for $\lambda = \mathit{rate}(\hat{\varphi}(s'))$, $\lambda > 0$, $\mu = \mathbb{P}_{\hat{\varphi}(s')}$ and there is no μ' such that $\hat{\varphi}(s') \xhookrightarrow{\tau}_\varphi \mu'$. Note that this means that from $\hat{\varphi}(s')$ there is a total outgoing rate of λ, spreading out according to μ. Hence, given an arbitrary state $\hat{u}$ in $\mathcal{M}/\varphi$, we have

$$\mu(\hat{u}) = \frac{\mathit{rate}(\hat{\varphi}(s'), \hat{u})}{\lambda}$$

By definition of the quotient there is at most one Markovian transition between any pair of states in $\mathcal{M}/\varphi$, so for every $\hat{u} \in \mathit{supp}(\mu)$, there is precisely one Markovian transition $\hat{\varphi}(s') \overset{\lambda'}{\rightsquigarrow}_\varphi \hat{u}$ with $\lambda' = \mu(\hat{u}) \cdot \lambda$. By definition of the quotient we then also find that λ' is the sum of all outgoing Markovian transitions in $\mathcal{M}$ from $\varphi(s')$ to states t such that $\varphi(t) = u$. Since each state in $\mathcal{M}$ has precisely one representative and $\hat{\varphi}(s')$ has a Markovian transition to all representatives of states reached from $\varphi(s')$ by Markovian transitions, it follows that the total outgoing rate of $\varphi(s')$ is also λ.

Additionally, there is no outgoing τ-transition from $\varphi(s')$, since by definition of the quotient this would have resulted in a τ-transition from $\hat{\varphi}(s')$, which we assumed is not present. Hence, there is an extended transition $\varphi(s') \xrightarrow{\chi(\lambda)} \mu'$ in $\mathcal{M}$. As the total outgoing rates of $\varphi(s')$ and $\hat{\varphi}(s')$ are equal, and the sum of all outgoing Markovian transitions from $\varphi(s')$ to states t such that $\varphi(t) = u$ equals the rate from $\hat{\varphi}(s')$ to $\hat{u}$, we find that $\mu \equiv_R \mu'$ since R equates states to their representative and to other states with the same representative.

By definition of the representation map, there is a $\mathcal{T}$-path (which is invisible and deterministic) from s' to $\varphi(s')$ in $\mathcal{M}$. Hence, $\varphi(s') \xrightarrow{\chi(\lambda)} \mu'$ implies that $s' \xRightarrow{\chi(\lambda)}_R \mu'$ and therefore also $s' \xRightarrow{\chi(\lambda)}{}'_R \mu'$. As $\chi(\lambda) = a$ and we already saw that $\mu \equiv_R \mu'$, this completes this part of the proof.

- Case 4: $(s, \hat{s}')$. Since $\hat{s}'$ is in $\mathcal{M}/\varphi$ and s is not, by definition of R we find that $\hat{s}' = \hat{\varphi}(s)$. By definition of the representation map, there is a $\mathcal{T}$-path from s to $\varphi(s)$ in $\mathcal{M}$. Hence, since $s \xrightarrow{a} \mu$, by Lemma A.24 we have either (1) $a = \tau \wedge \mu \equiv_{R'} \mathbb{1}_{\varphi(s)}$, or (2) there exists a transition $\varphi(s) \xrightarrow{a} \nu$ such that $\mu \equiv_{R'} \nu$, for $R' = \{(u, v) \mid u \twoheadrightarrow\twoheadleftarrow_{\mathcal{T}} v\}$. Again, as in case 2 we can safely substitute R' by R.

  (1) We need to show that $\hat{\varphi}(s) \xRightarrow{\tau}{}'_R \mu'$ such that $\mathbb{1}_{\varphi(s)} \equiv_R \mu'$. By definition of branching steps, we trivially have $\hat{\varphi}(s) \xRightarrow{\tau}{}'_R \mathbb{1}_{\hat{\varphi}(s)}$. Note that indeed $\mathbb{1}_{\varphi(s)} \equiv_R \mathbb{1}_{\hat{\varphi}(s)}$, since $(\varphi(s), \hat{\varphi}(s)) \in R$.

(2) If $\varphi(s) \xrightarrow{a} \nu$, by definition of the extended arrow either (2.a) $a \in A$ and $\varphi(s) \xhookrightarrow{a} \mu$, or (2.b) $a = \chi(\lambda)$ for $\lambda = \mathit{rate}(\varphi(s))$, $\lambda > 0$, $\mu = \mathbb{P}_{\varphi(s)}$ and there is no μ' such that $\varphi(s) \xhookrightarrow{\tau} \mu'$.

In case of (2.a), by definition of the quotient we find that $\hat{\varphi}(s) \xhookrightarrow{a}_{\varphi} \nu_{\varphi}$. Hence, also $\hat{\varphi}(s) \overset{a}{\Longrightarrow}'_R \nu_{\varphi}$. As observed above, $\nu_{\varphi} \equiv_R \nu$. Also, since $\mu \equiv_R \nu$ by assumption, transitivity of $\equiv_R$ yields $\mu \equiv_R \nu_{\varphi}$.

In case of (2.b), $\varphi(s)$ has a total outgoing rate of λ and this is spread out according to μ. That is, for each state t, there is a rate of $\lambda \cdot \mu(t)$ from $\varphi(s)$ to t. Let $C = [t]_R$ for some state t, and let λ' be the total rate from $\varphi(s)$ to C. By definition of the quotient, this implies that there is a rate of λ' from $\hat{\varphi}(s)$ to $\hat{\varphi}(t)$ in $\mathcal{M}/\varphi$ as well. Since the only element of C reachable from $\hat{\varphi}(s)$ is $\hat{\varphi}(t)$, this implies that there is a rate from $\hat{\varphi}(s)$ to C of λ'. Hence, for an arbitrary equivalence class C we find identical rates from $\varphi(s)$ to C and from $\hat{\varphi}(s)$ to C. This immediately implies that the outgoing rates of $\varphi(s)$ and $\hat{\varphi}(s)$ coincide, and that $\mathbb{P}_{\varphi(s)} \equiv_R \mathbb{P}_{\hat{\varphi}(s)}$. By definition of extended transitions now $\hat{\varphi}(s) \xrightarrow{\chi(\lambda)}_{\varphi} \mathbb{P}_{\hat{\varphi}(s)}$, and hence $\hat{\varphi}(s) \overset{a}{\Longrightarrow}'_R \mathbb{P}_{\hat{\varphi}(s)}$. Since $\mu = \mathbb{P}_{\varphi(s)}$ and $\mathbb{P}_{\varphi(s)} \equiv_R \mathbb{P}_{\hat{\varphi}(s)}$, this completes this part of the proof.

It remains to show that R is divergence sensitive. So, let $(s, s') \in R$. Again, we make a case distinction based on the origin of s and s'. Like before, if both states are in $\mathcal{M}/\varphi$ then they coincide, and hence the result immediately follows. Also, if both states are in $\mathcal{M}$, then divergence of s' is implied by divergence of s. After all, having the same representative they must be connected by confluent transitions, and hence Theorem 6.13 and the fact that the quotient is not involved give the result.

So, we only need to show (1) whether divergence in a state s in $\mathcal{M}$ implies divergence in its representative $\hat{\varphi}(s)$ in $\mathcal{M}/\varphi$, and (2) whether divergence in a state $\hat{t} \in \mathcal{M}/\varphi$ implies divergence in all states s in $\mathcal{M}$ such that $\varphi(s) = t$.

(1) Assume that there is a diverging scheduler for some state s in $\mathcal{M}$. We need to show that there also is a diverging scheduler for $\hat{\varphi}(s)$ in $\mathcal{M}/\varphi$. First of all note that, by Theorem 6.13, divergence of s implies divergence of $\varphi(s)$ in $\mathcal{M}$. Hence, we can assume that there is a scheduler $\mathcal{S}$ such that

$$\forall \pi \in \mathit{finpaths}^{\mathcal{S}}_{\mathcal{M}}(\varphi(s)) \; . \; A_{\text{red}}(\pi) \wedge \mathcal{S}(\pi)(\bot) = 0$$

It is well known that we can assume that this diverging scheduler is memoryless and deterministic.

By the existence of $\mathcal{S}$, there must be some transition $\varphi(s) \xhookrightarrow{\tau} \mu$ such that every state $t \in \mathit{supp}(\mu)$ is also diverging and has the same labelling as $\varphi(s)$. By the definition of the quotient, then there also is a transition $\hat{\varphi}(s) \xhookrightarrow{\tau}_{\varphi} \mu_{\varphi}$ in $\mathcal{M}/\varphi$. Since μ_{φ} is obtained from μ by taking the representatives of all states in its support, $\hat{\varphi}(s) \xhookrightarrow{\tau}_{\varphi} \mu_{\varphi}$ is also invisible by definition of the representation map, the fact that confluent transitions are invisible and the fact that the quotient leaves the state labelling invariant.

We can now construct a diverging scheduler for $\hat{\varphi}(s)$ that starts with this transition. Then, it invisibly ends up in either one of a set of states that are all representatives of diverging states. From all those states, the above argument can be repeated to take the next invisible transition. As this process can be extended indefinitely, indeed $\hat{\varphi}(s)$ is diverging too.

(2) Assume that there is a scheduler $\mathcal{S}$ such that

$$\forall \pi \in \mathit{finpaths}^{\mathcal{S}}_{\mathcal{M}/\varphi}(\hat{s}) \ . \ A_{\mathrm{red}}(\pi) \wedge \mathcal{S}(\pi)(\bot) = 0$$

for some state $\hat{s}$ in $\mathcal{M}/\varphi$. It is well known that we can assume that this diverging scheduler is memoryless and deterministic.

We need to show that there also is a diverging scheduler for every state s' in $\mathcal{M}$ such that $\varphi(s') = s$. First of all note that, by Theorem 6.13, divergence of $\varphi(s')$ implies divergence of s' in $\mathcal{M}$. Hence, it suffices to show divergence of $\varphi(s')$ based on divergence of $\hat{\varphi}(s')$ $(= \hat{s})$.

By the existence of $\mathcal{S}$, there must be an invisible transition $\hat{\varphi}(s') \xhookrightarrow{\tau}_{\varphi} \mu$ such that every state $\hat{t} \in \mathit{supp}(\mu)$ is also diverging. By the definition of the quotient, then there also is a transition $\varphi(s') \xhookrightarrow{\tau} \nu$ in $\mathcal{M}$ such that $\nu_{\varphi} = \mu$. Again, it can easily be seen that $\varphi(s') \xhookrightarrow{\tau} \nu$ is also invisible. Hence, we can now construct a diverging scheduler for $\varphi(s')$ that starts with this transition. Then, it invisibly ends up in either one of a set of states that all have a diverging representative. From all those states, the above argument can be repeated to take the next invisible transition. As this process can be extended indefinitely, indeed $\varphi(s')$ (and hence s') is diverging too. □

A.5 Proofs for Chapter 7

A.5.1 Proof of Theorem 7.21

Lemma A.25. *Let $M = (S, s^0, A, P, \mathrm{AP}, L)$ be an MDP, and $a, b \in A$ two independent actions such that $a \in A_{\mathrm{det}}$. Let $s \in S$ such that $\{a, b\} \subseteq \mathit{en}(s)$, and assume that $s \xrightarrow{a} s'$ and $s \xrightarrow{b} \mu$.*

If $\mathcal{T}$ contains all outgoing a-transitions from states in the support of μ, i.e.,

$$\mathcal{T} \supseteq \{(t, a, \mu) \in \Delta_M \mid t \in \mathit{supp}(\mu)\}$$

then there is a distribution $\nu \in \mathrm{Distr}(S)$ such that $s' \xrightarrow{b} \nu$ and $\mu \equiv_R \nu$, with R the smallest equivalence relation such that

$$R \supseteq \{(s, t) \in \mathit{supp}(\mu) \times \mathit{supp}(\nu) \mid (s \xrightarrow{a} t) \in \mathcal{T}\}$$

Proof. Since a and b are independent, by definition $\{a, b\} \subseteq \mathit{en}(s)$ and $s \xrightarrow{a} s'$ imply that there is a distribution $\nu \in \mathrm{Distr}(S)$ such that $s' \xrightarrow{b} \nu$.

For any $t \in \mathit{supp}(\nu)$, let $R_t = \{r \in \mathit{supp}(\mu) \mid r \xrightarrow{a} t\}$ be the set of states from the support of μ that can reach t by an a-action. As a and b are independent, R_t is not empty, and when taking into account the assumption that a is deterministic it follows that $\{R_t \mid t \in supp(\nu)\}$ is a partitioning of $supp(\mu)$. Note that

$$\begin{aligned}\nu(t) &= \sum_{s'' \in S} P(s,a)(s'') \cdot P(s'',b)(t) = \sum_{s'' \in S} P(s,b)(s'') \cdot P(s'',a)(t) \\ &= \sum_{s'' \in R_t} P(s,b)(s'') \cdot P(s'',a)(t) = P(s,b)(R_t) = \mu(R_t).\end{aligned}$$

The first equality follows from the fact that a is deterministic, the second from the independence of a and b, the third from the definition of R_t and the fourth from the fact that a is deterministic.

Note that R is the smallest equivalence relation relating each state $t \in supp(\nu)$ to all states in R_t. From the above it follows that $\mu \equiv_R \nu$. □

Recall that $\overline{A}(s)$ contains the actions that are enabled from s by a reduction function A in case s is not fully explored (the *nontrivial transitions*); otherwise, $\overline{A}(s)$ is the empty set (Definition 7.6).

Theorem 7.21. *Let A be an ample set reduction function for an MDP $M = (S, s^0, A, P, \mathrm{AP}, L)$. Then, the set $\mathcal{T}_A = \{(s,a,\mu) \in \Delta_M \mid a \in \overline{A}(s)\}$ is acyclic, and consists of probabilistically confluent transitions.*

Proof. Firstly, the fact that $\mathcal{T}_A$ is acyclic follows from the ample set condition A3: a cycle of nontrivial transitions would violate the condition. Secondly, to show that all the transitions in $\mathcal{T}_A$ are confluent, we need to find a confluent set of transitions $\mathcal{T}_A^* \supseteq \mathcal{T}_A$ in which they are contained. Let $\mathcal{T}_A^*$ be defined as the minimal set that satisfies the following:

- $\mathcal{T_A}^* \supseteq \mathcal{T}_A$;
- If $(s, a, \mathbb{1}_t) \in \mathcal{T}_A^*$ and $b \in en(s)$ $(b \neq a)$, then

$$\{(s_0, a, \mu) \in \Delta_M \mid s_0 \in supp(P(s,b))\} \subseteq \mathcal{T}_A^*$$

To prove that $\mathcal{T}_A^*$ is confluent, first note that by conditions A1 and A4 of the definition of ample sets and by construction of $\mathcal{T}_A^*$, only transitions with invisible actions are ever added to the set. Second, let $(s \xrightarrow{a} t) \in \mathcal{T}_A^*$ and let $s \xrightarrow{b} \mu$ be a transition of M. If b equals a, then the condition for confluence is trivially fulfilled, so assume that $b \neq a$. If we can prove that a and b are independent, confluence follows from Lemma A.25. Note that this lemma is indeed applicable, since by construction $\mathcal{T}_A^*$ contains all a-transitions from the support of $P(s,b)$.

By definition of $\mathcal{T}_A^*$, there must be some state s^* and a (possibly trivial) path $s^* \xrightarrow{b_1 \ldots b_n} s$ such that $b_i \neq a$ for each i, and $a \in \overline{A}(s^*)$. Then, $\overline{A}(s^*) = \{a\}$, by condition A4 of ample sets. Condition A2 guarantees that if b depends on a, we would have at least one $b_i \in \overline{A}(s^*)$, contradicting A4. Thus, a and b are independent.

Also note that, if $(s \xrightarrow{b} \mu) \in \mathcal{T}_A^*$ too, then for confluence it has to be mimicked by a confluent transition. Indeed, since $(s \xrightarrow{b} \mu) \in \mathcal{T}_{\mathcal{A}}^*$ and $a \in en(s)$, by construction also the b-transition from t is in $\mathcal{T}_{\mathcal{A}}^*$. □

A.5.2 Proof of Theorem 7.31

Lemma A.26. *Let $M = (S, s^0, A, P, \mathrm{AP}, L)$ be an MDP, $a \in A_{\mathrm{det}}$ a deterministic action, $s \in S$ a state, and $\mathcal{T} \supseteq \{(t, a, \mu) \in \Delta_M \mid t \in R_a(s)\}$ a set containing all a-labelled transitions enabled from some state that is reachable from s without doing any a-transitions. For any action $b \in A$ such that $b \neq a$, the implication*

$$\{a, b\} \subseteq en(s') \quad \Longrightarrow \quad P(s', b) \rightsquigarrow_{\mathcal{T}} P(target(s', a), b)$$

holds for every $s' \in R_a(s)$ if and only if a is independent of b at s.

Proof. ($\Rightarrow$) To prove the "only if" part of this lemma, take an arbitrary action $b \neq a$ and consider any state $s' \in R_a(s)$ such that $\{a, b\} \subseteq en(s')$. According to the assumptions of the lemma the a-transition from s' has to be in $\mathcal{T}$, so let $(s', a, \mathbb{1}_t) \in \mathcal{T}$.

Due to the implication assumed by this lemma, $P(s', b) \rightsquigarrow_{\mathcal{T}} P(t, b)$. Now, from this and the fact that $\mathcal{T}$ only contains a-transitions, using the part $\forall s^* \in S_i \,.\, \exists a \in A \,.\, (s^*, a, \mathbb{1}_{s_i}) \in \mathcal{T}$ of the conjunction in the definition of $\rightsquigarrow_{\mathcal{T}}$, the first condition for independence is satisfied. For the second condition, observe that

$$\begin{aligned}\sum_{s^* \in S} P(s', a)(s^*) \cdot P(s^*, b)(u) = P(t, b)(u) &= \sum_{s^* \in S_u} P(s', b)(s^*) \\ &= \sum_{\substack{s^* \in S \\ s^* \xrightarrow{a} u}} P(s', b)(s^*) = \sum_{s^* \in S} P(s', b)(s^*) \cdot P(s^*, a)(u)\end{aligned}$$

where the first and last step follow from the fact that a is deterministic, the second and third from the definition of $\rightsquigarrow_{\mathcal{T}}$. We used S_u to denote the class in the partitioning according to $\rightsquigarrow_{\mathcal{T}}$, corresponding to state u.

($\Leftarrow$) For the "if" part of this lemma, assume that a is independent of b at s, and let $s' \in R_a(s)$ be an arbitrary state such that $\{a, b\} \subseteq en(s')$. Carrying out exactly the same calculations as in Lemma A.25 for s' (note that $\mathcal{T}$ indeed contains all a-transitions from the support of $P(s', b)$ since all these states are also in $R_a(s)$), we see that $P(s', b) \rightsquigarrow_{\mathcal{T}} P(target(s', a), b)$. $\square$

Theorem 7.31. *Let $M = (S, s^0, A, P, \mathrm{AP}, L)$ be an MDP. Then, $T \colon S \to \mathscr{P}(A)$ is an acyclic action-separable restricted confluence reduction function if and only if T is a relaxed ample set reduction function.*

Proof. ($\Rightarrow$) To prove the "only if" part of the theorem, let $\mathcal{T}$ be the action-separable restricted confluent set underlying T, and let $s \in S$ be an arbitrary state. In this proof, when we write that a transition is confluent we mean that it is confluent *and* that it is in $\mathcal{T}$. If $T(s) = en(s)$, then all ample set conditions hold vacuously, so assume that $T(s) \neq en(s)$. Thus, by definition of confluence reduction functions, $T(s) = \{a\}$ for some action $a \in A_{\mathrm{red}}$ such that $(s, a, \mathbb{1}_t) \in \mathcal{T}$ for some state $t \in S$.

Condition A0 is clearly satisfied. Moreover, A1 follows from fact that only transitions with invisible (and thus stuttering) actions can be confluent, A3 from the acyclicity of T and A4 by construction and from the fact that all confluent transitions are deterministic.

For condition A2*, we prove the contrapositive: given an arbitrary path $s \xrightarrow{a_1 a_2 \dots a_n b} t$ in M such that $b \notin T(s)$ and $a_i \notin T(s)$ for every i, we show that $T(s)$ is independent of b at s. For each action a_i on this path, let s_i be the state reached immediate after taking this action. Due to Lemma A.26, it is enough to prove that $(s', a, \mathbb{1}_{target(s',a)}) \in \mathcal{T}$ for every $s' \in R_a(s)$ and additionally $P(s', b) \rightsquigarrow_{\mathcal{T}} P(target(s', a), b)$ if $\{a, b\} \subseteq en(s')$.

Let $s' \in R_a(s)$, so there is a path $s \xrightarrow{a_1 a_2 \dots a_m} s_m$ such that $a_i \neq a$ for every i and $s_m = s'$. Again, for each action a_i on this path, let s_i be the state reached immediate after taking this action. Since there is a confluent a-transition from s and also $a_1 \in en(s)$ and $a_1 \neq a$, by definition of restricted confluence $P(s, a_1) \rightsquigarrow_{\mathcal{T}} P(target(s, a), a_1)$. Now, by definition of $\rightsquigarrow$ and using action-separability, there has to be a confluent a-labelled transition from s_1. Repeating this argument from s_1 we find that $P(s_1, a_2) \rightsquigarrow_{\mathcal{T}} P(target(s_1, a), a_2)$ and that there is a confluent a-labelled transition from s_2, and continuing this way that $P(s_{m-1}, a_m) \rightsquigarrow_{\mathcal{T}} P(target(s_{m-1}, a), a_m)$ and that there is a confluent a-labelled transition from s_m. So, since $s_m = s'$, indeed $(s', a, \mathbb{1}_{target(s',a)}) \in \mathcal{T}$. Now, if $\{a, b\} \subseteq en(s')$, then the same argument can be applied once more from s', obtaining $P(s', b) \rightsquigarrow_{\mathcal{T}} P(target(s', a), b)$. ($b \neq a$ since it was assumed that $b \notin T(s)$.)

($\Leftarrow$) To prove the "if" part of the theorem, let $\mathcal{T}_a$ be the set of nontrivial actions of the relaxed ample set reduction function that are labelled by a. Now, the construction and proof of confluence of a set $\mathcal{T}_a^* \supseteq \mathcal{T}_a$ works almost exactly as in Theorem 7.21: the construction never adds actions that have a label that is different from a to the set (so action-separability is guaranteed), and the proof of confluence does not rely in any way on the liberal parts that we removed from the definitions.

The only difference is that now, due to the relaxed condition A2*, a and b are not necessarily globally independent anymore. However, confluence can still be proven. To see this, let $(s, a, \mathbb{1}_t) \in \mathcal{T}_a^*$ and let $s \xrightarrow{b} \mu$ be a transition of M. If b equals a, then again the condition for confluence is trivially fulfilled, so assume that $b \neq a$. Now, by definition of $\mathcal{T}_a^*$, there must be some state s^* and a (possibly empty) path $s^* \xrightarrow{b_1 \dots b_n} s$ such that $b_i \neq a$ for each i, and $a \in \overline{T}(s^*)$. Then, $\overline{T}(s^*) = \{a\}$, by condition A4 of ample sets. Condition A2* guarantees that if a depends on b at s^*, we would have at least one $b_i \in \overline{T}(s^*)$, contradicting A4. Thus, a is independent of b at s^*. As $s \in R_a(s^*)$, the conditions of local independence hold at s. Now, confluence follows from Lemma A.25. (Note that, technically, this lemma is not applicable: although by construction $\mathcal{T}_a^*$ contains all a-transitions from the support of $P(s, b)$, a and b are not globally independent. However, the fact that the independence equations hold at s is the only thing that is used in the proof of Lemma A.25, so the result is still valid.)

Note that the union of these confluent sets $\mathcal{T}_a$ is an action-separable confluent set, as the action-specific subsets are exactly the sets $\mathcal{T}_a$ constructed above. Thus, we get the result by taking the union of every $\mathcal{T}_a$, as a ranges over all (invisible) actions: the resulting action-separable confluent set $\mathcal{T}$ contains all nontrivial transitions of T and therefore proves that T is an acyclic action-separable restricted confluence reduction function.

The fact that we indeed can take the union of these sets $\mathcal{T}_a$ follows from the same argument as discussed in the end of Theorem 7.21. $\square$

A.6 Proofs for Chapter 8

A.6.1 Proof of Theorem 8.11

Theorem 8.11. *Let M be an MDP, $\mathcal{T}$ a confluent set of its transitions and T an acyclic confluence reduction function under $\mathcal{T}$. Let M_T be the reduced MDP. Then, M and M_T satisfy the same PCTL$^*_{\setminus X}$ formulae.*

Proof. This theorem precisely corresponds to Corollary 7.20. That corollary is based on Theorem 7.19, which states that $M \equiv_{\text{pvb}} M_T$. Although those results were for MDPs where each state can have only one outgoing transition for each action label, this property is not used in any of the proofs. Hence, the results apply just as well for our type of MDPs. Additionally, we now allow countable state spaces, while in Chapter 7 finiteness was assumed. However, as we only consider finite subparts of an MDP during simulation, this also does not matter.

More importantly, the results in Chapter 7 were based on a slightly more restrictive notion of confluence. Although technically probabilistic visible bisimulation is not preserved anymore under the new definition of Chapter 8, the bisimulation notion could be altered to also just require invisible transitions instead of invisible actions, and also allow transitions to be mimicked by transitions with a different action. As discussed in detail in Section 8.3, this would not change anything to the fact that PCTL$^*_{\setminus X}$ properties are preserved. $\square$

A.6.2 Proof of Theorem 8.12

Lemma A.27. *Given two distributions μ, ν,*

$$\textit{checkEquivalence}(\mu, \nu) \quad \Longrightarrow \quad \mu \equiv_{R^{\mathcal{T}}_{\mu,\nu}} \nu$$

where $\mathcal{T}$ is the set of confluent transitions at termination of checkEquivalence.

Proof. First of all, note that $\mathcal{T}$ only grows during *checkEquivalence*. After all, each call to *checkConfluence* may only add transitions to it, or leave it unchanged.

Assume that *checkEquivalence*(μ, ν) yields *true*. Hence, $\mu(q) = \nu(q)$ for every $q \in Q$, using the set Q after the loop. Note that Q is a partitioning of the set $\mathit{supp}(\mu) \cup \mathit{supp}(\nu)$, since initially it contains all singletons, and the loop only merges some of its elements. Now let $Q' = Q \cup \{\{q\} \mid q \notin \mathit{supp}(\mu) \cup \mathit{supp}(\nu)\}$ be a partitioning of the complete set of states S. We also have $\mu(q) = \nu(q)$ for every $q \in Q'$, as both μ and ν assign probability 0 to all newly added classes. Let Q'' be the equivalence relation associated with Q', i.e., $(s,t) \in Q''$ if and only if there is a set $q' \in Q'$ such that $s, t \in q'$. Since the function returns *true*, by definition we have $\mu \equiv_{Q''} \nu$.

It remains to show that $Q'' \subseteq R^{\mathcal{T}}_{\mu,\nu}$; by Proposition 5.2.1.5 of [Sto02a], then indeed $\mu \equiv_{R^{\mathcal{T}}_{\mu,\nu}} \nu$. Recall that $R^{\mathcal{T}}_{\mu,\nu}$ is the smallest equivalence relation containing the set

$$R' = \{(s,t) \mid s \in \mathit{supp}(\mu), t \in \mathit{supp}(\nu), \exists\, a \,.\, s \xrightarrow{a} t \in \mathcal{T}\}$$

where we chose $\mathcal{T}$ to be the set at termination of *checkEquivalence*. Hence, $(s,t) \in R^{\mathcal{T}}_{\mu,\nu}$ if and only if $s = t$ or there are states $s_0, s_1, \dots, s_n$ such that $s_0 = s$, $s_n = t$ and either $(s_i, s_{i+1}) \in R'$ or $(s_{i+1}, s_i) \in R'$ for every $0 \leq i < n$.

So, let $(s,t) \in Q''$. We show that also $(s,t) \in R^{\mathcal{T}}_{\mu,\nu}$. If $s = t$, this is immediate, so assume that $s \neq t$. By construction, there is a set $q' \in Q$ such that $s, t \in q'$. For s and t to be in the same set, some merges must have taken place in the loop.

If $s \in \mathit{supp}(\mu)$, $s_1 \in \mathit{supp}(\nu)$ and $s \xrightarrow{a} s_1 \in \mathcal{T}$ (at some point, so since $\mathcal{T}$ only grows also at the end), then $\{s\}$ and $\{s_1\}$ are merged. Hence, this corresponds to $(s, s_1) \in R'$. Alternatively, the same merge also happens if $s \in \mathit{supp}(\nu)$, $s_1 \in \mathit{supp}(\mu)$ and $s_1 \xrightarrow{a} s \in \mathcal{T}$, hence, $(s_1, s) \in R'$. The set $\{s, s_1\}$ can grow further in the same way, until it at some point contains t. This procedure corresponds exactly to the requirement that $(s,t) \in R^{\mathcal{T}}_{\mu,\nu}$.

(In this proof we used $s \xrightarrow{a} \mu \in \mathcal{T}$ and *checkConfluence*$(s \xrightarrow{a} \mu)$ interchangeably; after all, if *checkConfluence*$(s \xrightarrow{a} \mu)$ returns *true* then indeed also $s \xrightarrow{a} \mu \in \mathcal{T}$, and if $s \xrightarrow{a} \mu \in \mathcal{T}$ then *checkConfluence*$(s \xrightarrow{a} \mu)$ returns *true*.) □

Theorem 8.12. *Given a transition $p \xrightarrow{l} \mathbb{1}_q$, checkConfluence$(p \xrightarrow{l} \mathbb{1}_q)$ and checkConfluentMimicking together imply that $p \xrightarrow{l} \mathbb{1}_q$ is confluent.*

Proof. We need to show that there exists a confluent set of transitions containing $p \xrightarrow{l} \mathbb{1}_q$. We show that, upon termination of the algorithm and returning *true*, the set $\mathcal{T}$ fulfills this condition. Clearly, $p \xrightarrow{l} \mathbb{1}_q \in \mathcal{T}$, since it is always added immediately at the beginning of *checkConfluence* (except in case that *false* is returned due to it being visible), and only removed before returning *false*. Since we assumed that *true* is returned, indeed $p \xrightarrow{l} \mathbb{1}_q \in \mathcal{T}$.

To show that $\mathcal{T}$ is a confluent set, let $s \xrightarrow{a} \mathbb{1}_t \in \mathcal{T}$ be an arbitrary element. Note that indeed any element of $\mathcal{T}$ is deterministic, since the inner *foreach* loop of *checkConfluence* ascertains that only for such transitions the function *checkConfluence* is called (and hence only they are potentially added to $\mathcal{T}$). We have to prove that $s \xrightarrow{a} \mathbb{1}_t$ is invisible and that, for every $s \xrightarrow{b} \mu$ we have either $\mu = \mathbb{1}_t$ or there exists a transition $t \xrightarrow{c} \nu$ such that $\mu \equiv_{R^{\mathcal{T}}_{\mu,\nu}} \nu$. Also, we need to show that $t \xrightarrow{c} \nu$ is in $\mathcal{T}$ if $s \xrightarrow{b} \mu$ is. We postpone this last part to the end.

Since $s \xrightarrow{a} \mathbb{1}_t \in \mathcal{T}$, at some point *checkConfluence*$(s \xrightarrow{a} \mathbb{1}_t)$ must have been called, $s \xrightarrow{a} \mathbb{1}_t$ was added to $\mathcal{T}$ and subsequently not removed. This implies that $L(s) = L(t)$ (and hence indeed the transition is invisible) and that the algorithm terminated with the final *return true* statement. Hence, the outermost *foreach* loop never reached the end of its body, but was always cut short before by a *continue* statement. So, for each $s \xrightarrow{b} \mu$ it holds that either $\mu = \mathbb{1}_t$ or there exists a transition $t \xrightarrow{c} \nu$ for which the second *foreach* loop reached its *continue* statement. In the second case, *checkEquivalence*(μ, ν) yielded *true*, and by Lemma A.27, this implies that $\mu \equiv_{R^{\mathcal{T}}_{\mu,\nu}} \nu$ was true at the end of each iteration of the loop. Since $\mathcal{T}$ can only grow during the loop, and also afterwards no transitions are removed from $\mathcal{T}$ anymore (because otherwise $p \xrightarrow{l} \mathbb{1}_q$ would have been removed too), the set $\mathcal{T}$ at the end of the algorithm is a superset of the set $\mathcal{T}$ at the moment that $\mu \equiv_{R^{\mathcal{T}}_{\mu,\nu}} \nu$ was established. Hence, we also have $\mu \equiv_{R^{\mathcal{T}}_{\mu,\nu}} \nu$ for the final $\mathcal{T}$ (based on Proposition 5.2.1.5 of [Sto02a]), as required.

Finally, we show that if $s \xrightarrow{b} \mu$ is mimicked by $t \xrightarrow{c} \nu$ and $s \xrightarrow{b} \mu \in \mathcal{T}$, then so is $t \xrightarrow{c} \nu$. This follows from *checkConfluentMimicking*. After all, each transition and its mimicking transition that are found, are added to C in the body of *checkConfluence*. Only when $\mathcal{T}$ is reset also C is, since the mimickings that were found in that call are then clearly not relevant anymore. At the end, *checkConfluentMimicking* checks all of the mimicking pairs. If one fails the test, the function checks to see if it can still add $t \xrightarrow{c} \nu$ to $\mathcal{T}$ to make things right. Since we assumed that it returns *true*, apparently no irreparable violation was found, and indeed all confluent transitions are mimicked by confluent transitions. $\square$

APPENDIX B

List of papers by the author

Formal Verification

[1] D. Guck, H. Hatefi, H. Hermanns, J.-P. Katoen, and M. Timmer. Modelling, reduction and analysis of Markov automata. In *Proceedings of the 10th International Conference on Quantitative Evaluation of Systems (QEST)*, volume 8054 of *Lecture Notes in Computer Science*, pages 55–71. Springer, 2013.

[2] H. Hansen and M. Timmer. A comparison of confluence and ample sets in probabilistic and non-probabilistic branching time. *Theoretical Computer Science*, 2013. In press.

[3] A. Hartmanns and M. Timmer. On-the-fly confluence detection for statistical model checking. In *Proceedings of the 5th International NASA Formal Methods Symposium (NFM)*, volume 7871 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 2013.

[4] J.-P. Katoen, J. C. van de Pol, M. I. A. Stoelinga, and M. Timmer. A linear process-algebraic format for probabilistic systems with data. In *Proceedings of the 10th International Conference on Application of Concurrency to System Design (ACSD)*, pages 213–222. IEEE, 2010.

[5] J.-P. Katoen, J. C. van de Pol, M. I. A. Stoelinga, and M. Timmer. A linear process-algebraic format with data for probabilistic automata. *Theoretical Computer Science*, 413(1):36–57, 2012.

[6] M. Timmer. SCOOP: A tool for symbolic optimisations of probabilistic processes. In *Proceedings of the 8th International Conference on Quantitative Evaluation of Systems (QEST)*, pages 149–150. IEEE, 2011.

[7] M. Timmer, J.-P. Katoen, J. C. van de Pol, and M. I. A. Stoelinga. Efficient modelling and generation of Markov automata. In *Proceedings of the 23rd International Conference on Concurrency Theory (CONCUR)*, volume 7454 of *Lecture Notes in Computer Science*, pages 364–379. Springer, 2012.

[8] M. Timmer, M. I. A. Stoelinga, and J. C. van de Pol. Confluence reduction for probabilistic systems. In *Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 6605 of *Lecture Notes in Computer Science*, pages 311–325. Springer, 2011.

[9] M. Timmer, M. I. A. Stoelinga, and J. C. van de Pol. Confluence reduction for Markov automata. In *Proceedings of the 11th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, volume 8053 of *Lecture Notes in Computer Science*, pages 243–257. Springer, 2013.

[10] J. C. van de Pol and M. Timmer. State space reduction of linear processes using control flow reconstruction. In *Proceedings of the 7th International Symposium*

on Automated Technology for Verification and Analysis (ATVA), volume 5799 of *Lecture Notes in Computer Science*, pages 54–68. Springer, 2009.

Model-Based Testing

[11] M. I. A. Stoelinga and M. Timmer. Interpreting a successful testing process: Risk and actual coverage. In *Proceedings the of 3rd IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pages 251–258. IEEE, 2009.

[12] W. G. J. Stokkink, M. Timmer, and M. I. A. Stoelinga. Talking quiescence: a rigorous theory that supports parallel composition, action hiding and determinisation. In *Proceedings of the 7th Workshop on Model-Based Testing (MBT)*, volume 80 of *Electronic Proceedings in Theoretical Computer Science*, pages 73–87. Open Publishing Association, 2012.

[13] W. G. J. Stokkink, M. Timmer, and M. I. A. Stoelinga. Divergent quiescent transition systems. In *Proceedings the of 7th International Conference on Tests & Proofs (TAP)*, volume 7942 of *Lecture Notes in Computer Science*, pages 214–231. Springer, 2013.

[14] M. Timmer, H. Brinksma, and M. I. A. Stoelinga. Model-based testing. In *Software and Systems Safety - Specification and Verification*, volume 30 of *NATO Science for Peace and Security Series - D: Information and Communication Security*, pages 1–32. IOS Press, 2011.

Mathematics Teaching

[15] M. Alberink, H. Muijlwijk, and M. Timmer. De sinus: van meetkundige definitie naar analytisch begrip (The sine: from geometrical definition to analytical concept). *Euclides*, 86(6):250–252, 2011.

[16] M. I. A. Stoelinga and M. Timmer. Efficiënt zoeken in grote tekstbestanden (Searching efficiently in large text files). *Nieuwe Wiskrant*, 30(4):35–38, 2011.

[17] M. Timmer and N. C. Verhoef. Analytische meetkunde door een synthetische bril (Analytical geometry through a synthetic eye). *Nieuwe Wiskrant*, 31(4):13–18, 2012.

[18] M. Timmer and N. C. Verhoef. Increasing insightful thinking in analytic geometry. *Nieuw Archief voor Wiskunde*, 5/13(3):217–219, 2012.

[19] N. C. Verhoef and M. Timmer. Lesson study, deel 3: ervaringen bij de introductie van periodieke bewegingen (Lesson study, part 3: experiences with the introduction of periodic movements). *Euclides*, 88(4):173–176, 2013.

Telematics

[20] M. Timmer, P.-T. de Boer, and A. Pras. How to identify the speed limiting factor of a TCP flow. In *Proceedings of the 4th IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services (E2EMON)*, pages 17–24. IEEE, 2006.

Technical reports

[21] D. Guck, H. Hatefi, H. Hermanns, J.-P. Katoen, and M. Timmer. Modelling, reduction and analysis of Markov automata (extended version). Technical Report 1305.7050, ArXiv e-prints, 2013.

[22] A. Hartmanns and M. Timmer. On-the-fly confluence detection for statistical model checking (extended version). Technical Report TR-CTIT-13-04, Centre for Telematics and Information Technology, University of Twente, 2013.

[23] J.-P. Katoen, J. C. van de Pol, M. I. A. Stoelinga, and M. Timmer. A linear process algebraic format for probabilistic systems with data (extended version). Technical Report TR-CTIT-10-11, Centre for Telematics and Information Technology, University of Twente, 2010.

[24] M. I. A. Stoelinga and M. Timmer. Interpreting a successful testing process: Risk and actual coverage. Technical Report TR-CTIT-09-17, Centre for Telematics and Information Technology, University of Twente, 2009.

[25] W. G. J. Stokkink, M. Timmer, and M. I. A. Stoelinga. Talking quiescence: a rigorous theory that supports parallel composition, action hiding and determinisation (extended version). Technical Report TR-CTIT-12-05, Centre for Telematics and Information Technology, University of Twente, 2012.

[26] W. G. J. Stokkink, M. Timmer, and M. I. A. Stoelinga. Divergent quiescent transition systems. Technical Report TR-CTIT-13-08, Centre for Telematics and Information Technology, University of Twente, 2013.

[27] M. Timmer, J.-P. Katoen, J. C. van de Pol, and M. I. A. Stoelinga. Efficient modelling and generation of Markov automata (extended version). Technical Report TR-CTIT-12-16, Centre for Telematics and Information Technology, University of Twente, 2012.

[28] M. Timmer, M. I. A. Stoelinga, and J. C. van de Pol. Confluence reduction for probabilistic systems (extended version). Technical Report 1011.2314, ArXiv e-prints, 2010.

[29] M. Timmer, M. I. A. Stoelinga, and J. C. van de Pol. Confluence reduction for Markov automata (extended version). Technical Report TR-CTIT-13-14, Centre for Telematics and Information Technology, University of Twente, 2013.

[30] J. C. van de Pol and M. Timmer. State space reduction of linear processes using control flow reconstruction. Technical Report TR-CTIT-09-24, Centre for Telematics and Information Technology, University of Twente, 2009.

References

[ABC$^+$94] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons, 1994.

[ABW10] Y. Aït Ameur, F. Boniol, and V. Wiels. Toward a wider use of formal methods for aerospace systems design and verification. *International Journal on Software Tools for Technology Transfer*, 12(1):1–7, 2010.

[ACB84] M. Ajmone Marsan, G. Conte, and G. Balbo. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems*, 2(2):93–122, 1984.

[ASM80] J.-R. Abrial, S. A. Schuman, and B. Meyer. Specification language. In *On the Construction of Programs*, pages 343–410. Cambridge University Press, 1980.

[ASSB00] A. Aziz, K. Sanwal, V. Singhal, and R. K. Brayton. Model-checking continous-time Markov chains. *ACM Transactions on Computational Logic (TOCL)*, 1(1):162–170, 2000.

[ASU86] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 1986.

[Bae05] J. C. M. Baeten. A brief history of process algebra. *Theoretical Computer Science*, 335(2-3):131–146, 2005.

[Bam12] R. Bamberg. Non-deterministic generalised stochastic Petri nets: Modelling and analysis. Master's thesis, University of Twente, 2012.

[BB96] N. Balakrishnan and A. P. Basu, editors. *The Exponential Distribution: Theory, Methods and Applications*. Gorden and Breach Science Publishers, 1996.

[BBC$^+$09] S. Baarir, M. Beccuti, D. Cerotti, M. De Pierro, S. Donatelli, and G. Franceschinis. The GreatSPN tool: recent enhancements. *SIGMETRICS Performance Evaluation Review*, 36(4):4–9, 2009.

[BBG$^+$63] J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden, M. Woodger, and P. Naur. Revised report on the algorithm language ALGOL 60. *Communications of the ACM*, 6(1):1–17, 1963.

[BCD$^+$03] H. C. Bohnenkamp, T. Courtney, D. Daly, S. Derisavi, H. Hermanns, J.-P. Katoen, R. Klaren, V. Vi Lam, and W. H. Sanders. On integrating the Möbius and Modest modeling tools. In *Proceedings of the 33rd International Conference on Dependable Systems and Networks (DSN)*, page 671. IEEE, 2003.

[BCG04] C. Baier, F. Ciesinski, and M. Größer. PROBMELA: a modeling language for communicating probabilistic processes. In *Proceedings of* *the 2nd ACM/IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE)*, pages 57–66. IEEE, 2004.

[BCH+08] H. Boudali, P. Crouzen, B. R. Haverkort, M. Kuntz, and M. I. A. Stoelinga. Architectural dependability evaluation with Arcade. In *Proceedings of the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 512–521. IEEE, 2008.

[BCK+11] M. Bozzano, A. Cimatti, J.-P. Katoen, V. Y. Nguyen, T. Noll, and M. Roveri. Safety, dependability and performance analysis of extended AADL models. *The Computer Journal*, 54(5):754–775, 2011.

[BCP12] M. Bernardo, V. Cortellessa, and A. Pierantonio, editors. *Proceedings of the 12th International School on Formal Methods for the Design of Computer, Communication, and Software Systems: Formal Methods for Model-Driven Engineering (SFM)*, volume 7320 of *Lecture Notes in Computer Science*. Springer, 2012.

[BCS10] H. Boudali, P. Crouzen, and M. I. A. Stoelinga. A rigorous, compositional, and extensible framework for dynamic fault tree analysis. *IEEE Transactions on Dependable and Secure Computing*, 7(2):128–143, 2010.

[BdA95] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *Proceedings of the 15th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 1026 of *Lecture Notes in Computer Science*, pages 499–513. Springer, 1995.

[BDG06] C. Baier, P. R. D'Argenio, and M. Größer. Partial order reduction for probabilistic branching time. In *Proceedings of the 3rd Workshop on Quantitative Aspects of Programming Languages (QAPL)*, volume 153(2) of *Electronic Notes in Theoretical Computer Science*, pages 97–116. Elsevier, 2006.

[BDHK06] H. C. Bohnenkamp, P. R. D'Argenio, H. Hermanns, and J.-P. Katoen. MODEST: A compositional modeling formalism for hard and softly timed systems. *IEEE Transactions on Software Engineering*, 32(10):812–830, 2006.

[BDL+06] G. Behrmann, A. David, K. G. Larsen, J. Håkansson, P. Pettersson, W. Yi, and M. Hendriks. Uppaal 4.0. In *Proceedings of the 3rd International Conference on Quantitative Evaluation of Systems (QEST)*, pages 125–126. IEEE, 2006.

[Bel10] A. F. E. Belinfante. JTorX: A tool for on-line model-driven test derivation and execution. In *Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 6015 of *Lecture Notes in Computer Science*, pages 266–270. Springer, 2010.

[Ber08] Y. Bertot. A short presentation of Coq. In *Proceedings of the 21st International Conference on Theorem Proving in Higher Order Logics (TPHOLs)*, volume 5170 of *Lecture Notes in Computer Science*, pages 12–16. Springer, 2008.

[BFG99] M. Bozga, J.-C. Fernandez, and L. Ghirvu. State space reduction based on live variables analysis. In *Proceedings of the 6th International Symposium on Static Analysis (SAS)*, volume 1694 of *Lecture Notes in Computer Science*, pages 164–178. Springer, 1999.

[BFHH11] J. Bogdoll, L. M. F. Fioriti, A. Hartmanns, and H. Hermanns. Partial order methods for statistical model checking and simulation. In *Proceedings of the Joint 13th IFIP International Conference on Formal Methods for Open Object-based Distributed Systems (FMOODS) and 31st IFIP International Conference on FORmal TEchniques for Networked and Distributed Systems (FORTE)*, volume 6722 of *Lecture Notes in Computer Science*, pages 59–74. Springer, 2011.

[BG94a] M. Bezem and J. F. Groote. A correctness proof of a one-bit sliding window protocol in μCRL. *The Computer Journal*, 37(4):289–307, 1994.

[BG94b] M. Bezem and J. F. Groote. Invariants in process algebra with data. In *Proceedings of the 5th International Conference on Concurrency Theory (CONCUR)*, volume 836 of *Lecture Notes in Computer Science*, pages 401–416. Springer, 1994.

[BGC04] C. Baier, M. Größer, and F. Ciesinski. Partial order reduction for probabilistic systems. In *Proceedings of the 1st International Conference on Quantitative Evaluation of Systems (QEST)*, pages 230–239. IEEE, 2004.

[BGC09] C. Baier, M. Größer, and F. Ciesinski. Quantitative analysis under fairness constraints. In *Proceedings of the 7th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, volume 5799 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 2009.

[BGdMT06] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queueing Networks and Markov Chains - Modeling and Performance Evaluation with Computer Science Applications; 2nd Edition.* John Wiley & Sons, 2006.

[BH97] C. Baier and H. Hermanns. Weak bisimulation for fully probabilistic processes. In *Proceedings of the 9th International Conference on Computer Aided Verification (CAV)*, volume 1254 of *Lecture Notes in Computer Science*, pages 119–130. Springer, 1997.

[BH06] J. P. Bowen and M. G. Hinchey. Ten commandments of formal methods... ten years later. *IEEE Computer*, 39(1):40–48, 2006.

[BH$^+$09] E. Böde, M. Herbstritt, H. Hermanns, S. Johr, T. Peikenkamp, R. Pulungan, J. Rakow, R. Wimmer, and B. Becker. Compositional dependability evaluation for STATEMATE. *IEEE Transactions on Software Engineering*, 35(2):274–292, 2009.

[BHH12] J. Bogdoll, A. Hartmanns, and H. Hermanns. Simulation and statistical model checking for Modestly nondeterministic models. In *Proceedings of the 16th International GI/ITG Conference on Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance (MMB/DFT)*, volume 7201 of *Lecture Notes in Computer Science*, pages 249–252. Springer, 2012.

[BHHK03] C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time markov chains. *IEEE Transactions on Software Engineering*, 29(6):524–541, 2003.

[BHHZ11] P. Buchholz, E. M. Hahn, H. Hermanns, and L. Zhang. Model checking algorithms for CTMDPs. In *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV)*, volume 6806 of *Lecture Notes in Computer Science*, pages 225–242. Springer, 2011.

[BHK$^+$12] T. Brázdil, H. Hermanns, J. Krcál, J. Kretínský, and V. Rehák. Verification of open interactive Markov chains. In *Proceedings of the IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 18 of *Leibniz International Proceedings in Informatics*, pages 474–485. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012.

[BK89] J. A. Bergstra and J. W. Klop. ACP$_\tau$: A universal axiom system for process specification. In *Proceedings of the 1st Workshop on Algebraic Methods: Theory, Tools and Applications*, volume 394 of *Lecture Notes in Computer Science*, pages 447–463. Springer, 1989.

[BK08] C. Baier and J.-P. Katoen. *Principles of Model Checking.* MIT Press, 2008.

[BKHW05] C. Baier, J.-P. Katoen, H. Hermanns, and V. Wolf. Comparative branching-time semantics for Markov chains. *Information and Computation*, 200(2):149–214, 2005.

[Bla05] M. Bladt. A review on phase-type distributions and their use in risk theory. *ASTIN bulletin*, 35(1):145–161, 2005.

[Blo01] S. C. C. Blom. Partial τ-confluence for efficient state space generation. Technical Report SEN-R0123, Centrum voor Wiskunde en Informatica, 2001.

[Blo12] V. Bloemen. Analyzing old games with modern techniques: Probabilistic model checking using SCOOP and PRISM. Bachelor's thesis, University of Twente, 2012.

[BLvdPW11] S. C. C. Blom, B. Lisser, J. C. van de Pol, and M. Weber. A database approach to distributed state-space generation. *Journal of Logic and Computation*, 21:45–62, 2011.

[BN98] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[BP95] D. Bosscher and A. Ponse. Translating a process algebra with symbolic data values to linear format. In *Proceedings of the 1st Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume NS-95-2 of *BRICS Notes Series*, pages 119–130. University of Aarhus, 1995.

[Bro08] T. Bromwich. *An Introduction to the Theory of Infinite Series.* Macmillan, 1908.

[BS00] C. Baier and M. I. A. Stoelinga. Norm functions for probabilistic bisimulations with delays. In *Proceedings of the 3rd International Conference on Foundations of Software Science and Computation Structures (FOSSACS)*, volume 1784 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2000.

[BvdP02] S. C. C. Blom and J. C. van de Pol. State space reduction by proving confluence. In *Proceedings of the 14th International Conference on Computer Aided Verification (CAV)*, volume 2404 of *Lecture Notes in Computer Science*, pages 596–609. Springer, 2002.

[BvdP08] S. C. C. Blom and J. C. van de Pol. Symbolic reachability for process algebras with recursive data types. In *Proceedings of the 5th International Colloquium on Theoretical Aspects of Computing (ICTAC)*, volume 5160 of *Lecture Notes in Computer Science*, pages 81–95. Springer, 2008.

[BvdPW10] S. C. C. Blom, J. C. van de Pol, and M. Weber. LTSmin: Distributed and symbolic reachability. In *Proceedings of the 22nd International Conference on Computer Aided Verification (CAV)*, volume 6174 of *Lecture Notes in Computer Science*, pages 354–359. Springer, 2010.

[CB06] F. Ciesinski and C. Baier. LiQuor: A tool for qualitative and quantitative linear time analysis of reactive systems. In *Proceedings of the 3rd International Conference on Quantitative Evaluation of Systems (QEST)*, pages 131–132. IEEE, 2006.

[CE81] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Proceedings of the 3rd Workshop on Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.

[CGH$^+$10] N. Coste, H. Garavel, H. Hermanns, F. Lang, R. Mateescu, and W. Serwe. Ten years of performance evaluation for concurrent systems using CADP. In *Proceedings of the 4th International Symposium on Leveraging Applications of Formal Methods, Verification, and Validation (ISoLA), Part II*, volume 6416 of *Lecture Notes in Computer Science*, pages 128–142. Springer, 2010.

[CGK$^+$13] S. Cranen, J. F. Groote, J. J. A. Keiren, F. P. M. Stappers, E. P. de Vink, W. Wesselink, and T. A. C. Willemse. An overview of the mCRL2 toolset and its recent advances. In *Proceedings of the 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 7795 of *Lecture Notes in Computer Science*, pages 199–213. Springer, 2013.

[CGP01] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking.* MIT Press, 2001.

[Cha88] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

[CJEF96] E. M. Clarke, S. Jha, R. Enders, and T. Filkorn. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design*, 9(1/2):77–104, 1996.

[CJMS06] G. Ciardo, R. L. Jones, A. S. Miner, and R. Siminiceanu. Logic and stochastic modeling with Smart. *Performance Evaluation*, 63(6):578–608, 2006.

[CK04] M. Capinski and P. E. Kopp. *Measure, Integral and Probability.* Springer, 2004.

[CM88] K. M. Chandy and J. Misra. *Parallel Program Design: a Foundation.* Addison-Wesley, 1988.

[CY95] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.

[dAR07] L. de Alfaro and P. Roy. Magnifying-lens abstraction for Markov decision processes. In *Proceedings of the 19th International Conference on Computer Aided Verification (CAV)*, volume 4590 of *Lecture Notes in Computer Science*, pages 325–338. Springer, 2007.

[DH11] Y. Deng and M. Hennessy. On the semantics of Markov automata. In *Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 6756 of *Lecture Notes in Computer Science*, pages 307–318. Springer, 2011.

[DH13] Y. Deng and M. Hennessy. On the semantics of Markov automata. *Information and Computation*, 222:139–168, 2013.

[Dij70] E. W. Dijkstra. Structured programming. In *Software Engineering Techniques*. NATO Science Committee, 1970.

[DJJL01] P. R. D'Argenio, B. Jeannet, H. E. Jensen, and K. G. Larsen. Reachability analysis of probabilistic systems by successive refinements. In *Proceedings of the 1st Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM-PROBMIV)*, volume 2165 of *Lecture Notes in Computer Science*, pages 39–56. Springer, 2001.

[DJJL02] P. R. D'Argenio, B. Jeannet, H. E. Jensen, and K. G. Larsen. Reduction and refinement strategies for probabilistic analysis. In *Proceedings of the 2nd Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM-PROBMIV)*, volume 2399 of *Lecture Notes in Computer Science*, pages 57–76. Springer, 2002.

[DKP13] C. Dehnert, J.-P. Katoen, and D. Parker. SMT-based bisimulation minimisation of Markov models. In *Proceedings of the 14th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, volume 7737 of *Lecture Notes in Computer Science*, pages 28–47. Springer, 2013.

[DLLM09] R. De Nicola, D. Latella, M. Loreti, and M. Massink. Rate-based transition systems for stochastic process calculi. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP), part II*, volume 5556 of *Lecture Notes in Computer Science*, pages 435–446. Springer, 2009.

[DN04] P. R. D'Argenio and P. Niebert. Partial order reduction on concurrent probabilistic programs. In *Proceedings of the 1st International Conference on Quantitative Evaluation of Systems (QEST)*, pages 240–249. IEEE, 2004.

[Duf91] D. A. Duffy. *Principles of Automated Theorem Proving.* John Wiley & Sons, 1991.

[EHKZ13] C. Eisentraut, H. Hermanns, J.-P. Katoen, and L. Zhang. A semantics for every GSPN. In *Proceedings of the 34th International Conference on Application and Theory of Petri Nets and Other Models of Concurrency (ICATPN)*, volume 7927 of *Lecture Notes in Computer Science*, pages 90–109. Springer, 2013.

[EHZ10a] C. Eisentraut, H. Hermanns, and L. Zhang. Concurrency and composition in a stochastic world. In *Proceedings of the 21st International Conference on Concurrency Theory (CONCUR)*, volume 6269 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2010.

[EHZ10b] C. Eisentraut, H. Hermanns, and L. Zhang. On probabilistic automata in continuous time. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 342–351. IEEE, 2010.

[EP10] S. Evangelista and C. Pajault. Solving the ignoring problem for partial order reduction. *International Journal on Software Tools for Technology Transfer*, 12(2):155–170, 2010.

[ES96] E. A. Emerson and A. P. Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9(1/2):105–131, 1996.

[FBG03] J.-C. Fernandez, M. Bozga, and L. Ghirvu. State space reduction based on live variables analysis. *Science of Computer Programming*, 47(2-3):203–220, 2003.

[FG06] A. Fehnker and P. Gao. Formal verification and simulation for performance analysis for probabilistic broadcast protocols. In *Proceedings of the 5th International Conference on Ad-Hoc, Mobile, and Wireless Networks (ADHOC-NOW)*, volume 4104 of *Lecture Notes in Computer Science*, pages 128–141. Springer, 2006.

[FGK97] L. Fredlund, J. F. Groote, and H. Korver. Formal verification of a leader election protocol in process algebra. *Theoretical Computer Science*, 177(2):459–486, 1997.

[FKN$^+$11] V. Forejt, M. Z. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Quantitative multi-objective verification for probabilistic systems. In *Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 6605 of *Lecture Notes in Computer Science*, pages 112–127. Springer, 2011.

[FM91] J.-C. Fernandez and L. Mounier. “On the Fly” Verification of Behavioural Equivalences and Preorders. In *Proceedings of the 3rd International Workshop on Computer Aided Verification (CAV)*, volume 575 of *Lecture Notes in Computer Science*, pages 181–191. Springer, 1991.

[FO94] O. Færgemand and A. Olsen. Introduction to SDL-92. *Computer Networks and ISDN Systems*, 26(9):1143–1167, 1994.

[Fok07] W. Fokkink. *Introduction to Process Algebra*. Springer, 2007.

[FP05] W. Fokkink and J. Pang. Simplifying Itai-Rodeh leader election for anonymous rings. In *Proceedings of the 4th International Workshop on Automated Verification of Critical Systems (AVOCS)*, volume 128(6) of *Electronic Notes in Theoretical Computer Science*, pages 53–68. Elsevier, 2005.

[FPvdP06] W. Fokkink, J. Pang, and J. C. van de Pol. Cones and foci: A mechanical framework for protocol verification. *Formal Methods in System Design*, 29(1):1–31, 2006.

[FTW10] W. Fokkink, M. Torabi Dashti, and A. Wijs. Partial order reduction for branching security protocols. In *Proceedings of the 10th International Conference on Application of Concurrency to System Design (ACSD)*, pages 191–200. IEEE, 2010.

[GDF09] S. Giro, P. R. D’Argenio, and L. María Ferrer Fioriti. Partial order reduction for probabilistic systems: A revision for distributed schedulers. In *Proceedings of the 20th International Conference on Concurrency Theory (CONCUR)*, volume 5710 of *Lecture Notes in Computer Science*, pages 338–353. Springer, 2009.

[GH94] S. Gilmore and J. Hillston. The PEPA workbench: A tool to support a process algebra-based approach to performance modelling. In *Proceedings of the 7th International Conference on Computer Performance Evaluation, Modeling Techniques and Tools (TOOLS)*, volume 794 of *Lecture Notes in Computer Science*, pages 353–368. Springer, 1994.

[GHH$^+$13a] D. Guck, H. Hatefi, H. Hermanns, J.-P. Katoen, and M. Timmer. Modelling, reduction and analysis of Markov automata. In *Proceedings of the 10th International Conference on Quantitative Evaluation of Systems (QEST)*, volume 8054 of *Lecture Notes in Computer Science*, pages 55–71. Springer, 2013.

[GHH$^+$13b] D. Guck, H. Hatefi, H. Hermanns, J.-P. Katoen, and M. Timmer. Modelling, reduction and analysis of Markov automata (extended version). Technical Report 1305.7050, ArXiv e-prints, 2013.

[GHKN12] D. Guck, T. Han, J.-P. Katoen, and M. R. Neuhäußer. Quantitative timed analysis of interactive Markov chains. In *Proceedings of the 4th International NASA Formal Methods Symposium (NFM)*, volume 7226 of *Lecture Notes in Computer Science*, pages 8–23. Springer, 2012.

[GKO12] J. F. Groote, T. W. D. M. Kouters, and A. Osaiweran. Specification guidelines to avoid the state space explosion problem. In *Proceedings of the 4th IPM International Conference on Fundamentals of Software Engineering (FSEN)*, volume 7141 of *Lecture Notes in Computer Science*, pages 112–127. Springer, 2012.

[GKPP95] R. Gerth, R. Kuiper, D. Peled, and W. Penczek. A partial order approach to branching time logic model checking. In *Proceedings of the 3rd Israel Symposium on Theory of Computing and Systems (ISTCS)*, pages 130–139. IEEE, 1995.

[GKPP99] R. Gerth, R. Kuiper, D. Peled, and W. Penczek. A partial order approach to branching time logic model checking. *Information and Computation*, 150(2):132–152, 1999.

[GL01] J. F. Groote and B. Lisser. Computer assisted manipulation of algebraic process specifications. Technical Report SEN-R0117, Centrum voor Wiskunde en Informatica, 2001.

[GLMS13] H. Garavel, F. Lang, R. Mateescu, and W. Serwe. CADP 2011: a toolbox for the construction and analysis of distributed processes. *International Journal on Software Tools for Technology Transfer*, 15(2):89–107, 2013.

[GM98] J. F. Groote and R. Mateescu. Verification of temporal properties of processes in a setting with data. In *Proceedings of the 7th International Conference on Algebraic Methodology and Software Technology (AMAST)*, volume 1548 of *Lecture Notes in Computer Science*, pages 74–90. Springer, 1998.

[God96] P. Godefroid. *Partial-order Methods for the Verification of Concurrent Systems: an Approach to the State-explosion Problem*, volume 1032 of *Lecture Notes in Computer Science*. Springer, 1996.

[GP93] P. Godefroid and D. Pirottin. Refining dependencies improves partial-order verification methods. In *Proceedings of the 5th International Conference on Computer Aided Verification (CAV)*, volume 697 of *Lecture Notes in Computer Science*, pages 438–449. Springer, 1993.

[GP95] J. F. Groote and A. Ponse. The syntax and semantics of μCRL. In *Proceedings of the 1st Workshop on the Algebra of Communicating Processes (ACP)*, Workshops in Computing, pages 26–62. Springer, 1995.

[GPU01] J. F. Groote, A. Ponse, and Y. S. Usenko. Linearization in parallel pCRL. *Journal of Logic and Algebraic Programming*, 48(1-2):39–70, 2001.

[GPW03] J. F. Groote, J. Pang, and A. G. Wouters. Analysis of a distributed system for lifting trucks. *Journal of Logic and Algebraic Programming*, 55(1-2):21–56, 2003.

[Gri03] R. P. Grimaldi. *Discrete and Combinatorial Mathematics — An Applied Introduction, 5th Edition*. Pearson, 2003.

[Grö08] M. Größer. *Reduction Methods for Probabilistic Model Checking*. PhD thesis, Technische Universität Dresden, 2008.

[GS98] H. Garavel and M. Sighireanu. Towards a second generation of formal description techniques - rationale for the design of E-LOTOS. In *Proceedings of the 3rd International Workshop on Formal Methods for Industrial Critical Systems (FMICS)*, pages 198–230. Centrum voor Wiskunde en Informatica, 1998.

[GS01] J. F. Groote and J. Springintveld. Focus points and convergent process operators: a proof strategy for protocol verification. *Journal of Logic and Algebraic Programming*, 49(1-2):31–60, 2001.

[GS06] H. Garavel and W. Serwe. State space reduction for process algebra specifications. *Theoretical Computer Science*, 351(2):131–145, 2006.

[Guc12] D. Guck. Quantitative analysis of Markov automata. Master's thesis, RWTH Aachen University, 2012.

[GvdP00] J. F. Groote and J. C. van de Pol. State space reduction using partial τ-confluence. In *Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 1893 of *Lecture Notes in Computer Science*, pages 383 – 393. Springer, 2000.

[GW05] J. F. Groote and T. A. C. Willemse. Model-checking processes with data. *Science of Computer Programming*, 56(3):251–273, 2005.

[Har10] A. Hartmanns. Model-checking and simulation for stochastic timed systems. In *Proceedings of the 9th International Symposium on Formal Methods for Components and Objects (FMCO)*, volume 6957 of *Lecture Notes in Computer Science*, pages 372–391. Springer, 2010.

[Hav98] B. R. Haverkort. *Performance of Computer Communication Systems - a Model-Based Approach*. John Wiley & Sons, 1998.

[Her02] H. Hermanns. *Interactive Markov Chains: The Quest for Quantified Quality*, volume 2428 of *Lecture Notes in Computer Science*. Springer, 2002.

[Hes98] W. H. Hesselink. Invariants for the construction of a handshake register. *Information Processing Letters*, 68(4):173–177, 1998.

[HH09] A. Hartmanns and H. Hermanns. A Modest approach to checking probabilistic timed automata. In *Proceedings of the 6th International Conference on Quantitative Evaluation of Systems (QEST)*, pages 187–196. IEEE, 2009.

[HH12] H. Hatefi and H. Hermanns. Model checking algorithms for Markov automata. In *Proceedings of the 12th International Workshop on Automated Verification of Critical Systems (AVOCS)*, volume 53 of *Electronic Communications of the EASST*, 2012.

[HHK00] B. R. Haverkort, H. Hermanns, and J.-P. Katoen. On the use of model checking techniques for dependability evaluation. In *Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems (SRDS)*, pages 228–237. IEEE, 2000.

[HHK02] H. Hermanns, U. Herzog, and J.-P. Katoen. Process algebra for performance evaluation. *Theoretical Computer Science*, 274(1-2):43–87, 2002.

[Hil05] J. Hillston. Process algebras for quantitative analysis. In *Proceedings of the 20th IEEE Symposium on Logic in Computer Science (LICS)*, pages 239–248. IEEE, 2005.

[HJ90] H. Hansson and B. Jonsson. A calculus for communicating systems with time and probabilities. In *Proceedings of the 11th IEEE Real-Time Systems Symposium (RTSS)*, pages 278–287. IEEE, 1990.

[HJ94] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[HK97] M. Huth and M. Z. Kwiatkowska. Quantitative analysis and model checking. In *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 111–122. IEEE, 1997.

[HK09] H. Hermanns and J.-P. Katoen. The how and why of interactive Markov chains. In *Proceedings of the 8th International Symposium on Formal Methods for Components and Objects (FMCO)*, volume 6286 of *Lecture Notes in Computer Science*, pages 311–337. Springer, 2009.

[HKN$^+$08] J. Heath, M. Z. Kwiatkowska, G. Norman, D. Parker, and O. Tymchyshyn. Probabilistic model checking of complex biological pathways. *Theoretical Computer Science*, 391(3):239–257, 2008.

[HKQ11] H. Hansen, M. Z. Kwiatkowska, and H. Qu. Partial order reduction for model checking Markov decision processes under unconditional fairness. In *Proceedings of the 8th International Conference on Quantitative Evaluation of Systems (QEST)*, pages 203–212. IEEE, 2011.

[HLMP04] T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet. Approximate probabilistic model checking. In *Proceedings of the 5th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, volume 2937 of *Lecture Notes in Computer Science*, pages 73–84. Springer, 2004.

[HLP06] T. Hérault, R. Lassaigne, and S. Peyronnet. APMC 3.0: Approximate verification of discrete and continuous time Markov chains. In *Proceedings of the 3rd International Conference on Quantitative Evaluation of Systems (QEST)*, pages 129–130. IEEE, 2006.

[HMW09] T. A. Henzinger, M. Mateescu, and V. Wolf. Sliding window abstraction for infinite Markov chains. In *Proceedings of the 21st International Conference on Computer Aided Verification (CAV)*, volume 5643 of *Lecture Notes in Computer Science*, pages 337–352. Springer, 2009.

[HMZ$^+$12] D. Henriques, J. Martins, P. Zuliani, A. Platzer, and E. M. Clarke. Statistical model checking for Markov decision processes. In *Proceedings of the 9th International Conference on Quantitative Evaluation of Systems (QEST)*, pages 84–93. IEEE, 2012.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[Hol97] G. J. Holzmann. State compression in SPIN: Recursive indexing and compression training runs. In *Proceedings of the 3rd International SPIN workshop*, 1997.

[Hol04] G. J. Holzmann. *The SPIN Model Checker - Primer and Reference Manual.* Addison-Wesley, 2004.

[HP95] G. J. Holzmann and D. Peled. An improvement in formal verification. In *Proceedings of the 7th IFIP WG6.1 International Conference on Formal Description Techniques (FORTE)*, volume 6 of *IFIP Conference Proceedings*, pages 197–211. Chapman & Hall, 1995.

[HT13a] H. Hansen and M. Timmer. A comparison of confluence and ample sets in probabilistic and non-probabilistic branching time. *Theoretical Computer Science*, 2013. In press.

[HT13b] A. Hartmanns and M. Timmer. On-the-fly confluence detection for statistical model checking. In *Proceedings of the 5th International NASA Formal Methods Symposium (NFM)*, volume 7871 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 2013.

[HT13c] A. Hartmanns and M. Timmer. On-the-fly confluence detection for statistical model checking (extended version). Technical Report TR-CTIT-13-04, Centre for Telematics and Information Technology, University of Twente, 2013.

[HWZ08] H. Hermanns, B. Wachter, and L. Zhang. Probabilistic CEGAR. In *Proceedings of the 20th International Conference on Computer Aided Verification (CAV)*, volume 5123 of *Lecture Notes in Computer Science*, pages 162–175. Springer, 2008.

[ID96] C. N. Ip and D. L. Dill. Better verification through symmetry. *Formal Methods in System Design*, 9(1/2):41–75, 1996.

[IEE97] IEEE Standards Department. *802.11: IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification*, 1997.

[IR81] A. Itai and M. Rodeh. Symmetry breaking in distributive networks. In *Proceedings of the 22nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 150–158. IEEE, 1981.

[IR90] A. Itai and M. Rodeh. Symmetry breaking in distributed networks. *Information and Computation*, 88(1):60–87, 1990.

[JJ05] C. Jard and T. Jéron. TGV: theory, principles and algorithms. *International Journal on Software Tools for Technology Transfer*, 7(4):297–315, 2005.

[Kat12] J.-P. Katoen. GSPNs revisited: Simple semantics and new analysis algorithms. In *Proceedings of the 12th International Conference on Application of Concurrency to System Design (ACSD)*, pages 6–11. IEEE, 2012.

[KKLW07] J.-P. Katoen, D. Klink, M. Leucker, and V. Wolf. Three-valued abstraction for continuous-time Markov chains. In *Proceedings of the 19th International Conference on Computer Aided Verification (CAV)*, volume 4590 of *Lecture Notes in Computer Science*, pages 311–324. Springer, 2007.

[KKNP09] M. Kattenbelt, M. Z. Kwiatkowska, G. Norman, and D. Parker. Abstraction refinement for probabilistic software. In *Proceedings of the 19th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, volume 5403 of *Lecture Notes in Computer Science*, pages 182–197. Springer, 2009.

[KKNP10] M. Kattenbelt, M. Z. Kwiatkowska, G. Norman, and D. Parker. A game-based abstraction-refinement framework for Markov decision processes. *Formal Methods in System Design*, 36(3):246–280, 2010.

[KKZ05] J.-P. Katoen, M. Khattri, and I. S. Zapreev. A Markov reward model checker. In *Proceedings of the 2nd International Conference on Quantitative Evaluation of Systems (QEST)*, pages 243–244. IEEE, 2005.

[KKZJ07] J.-P. Katoen, T. Kemna, I. S. Zapreev, and D. N. Jansen. Bisimulation minimisation mostly speeds up probabilistic model checking. In *Proceedings of the 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 4424 of *Lecture Notes in Computer Science*, pages 87–101. Springer, 2007.

[KM04] M. Kaufmann and J. S. Moore. Some key research problems in automated theorem proving for hardware and software verification. *Ciencias de la Computación*, 98(1):181–195, 2004.

[KNP06] M. Z. Kwiatkowska, G. Norman, and D. Parker. Symmetry reduction for probabilistic model checking. In *Proceedings of the 18th International Conference on Computer Aided Verification (CAV)*, volume 4144 of *Lecture Notes in Computer Science*, pages 234–248. Springer, 2006.

[KNP08] M. Z. Kwiatkowska, G. Norman, and D. Parker. Using probabilistic model checking in systems biology. *SIGMETRICS Performance Evaluation Review*, 35(4):14–21, 2008.

[KNP11] M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV)*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591. Springer, 2011.

[KNP12] M. Z. Kwiatkowska, G. Norman, and D. Parker. Probabilistic verification of Herman's self-stabilisation algorithm. *Formal Aspects of Computing*, 24(4-6):661–670, 2012.

[KNS03] M. Z. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic model checking of deadline properties in the IEEE 1394 FireWire root contention protocol. *Formal Aspects of Computing*, 14(3):295–318, 2003.

[KP92] S. Katz and D. Peled. Defining conditional independence using collapses. *Theoretical Computer Science*, 101(2):337–359, 1992.

[KR01] J. F. Kurose and K. W. Ross. *Computer Networking - a Top-down Approach Featuring the Internet*. Addison-Wesley-Longman, 2001.

[KS94] H. Korver and J. Springintveld. A computer-checked verification of Milner's scheduler. In *Proceedings of the 2nd International Conference on Theoretical Aspects of Computer Software (TACS)*, volume 789 of *Lecture Notes in Computer Science*, pages 161–178. Springer, 1994.

[KvdPST10a] J.-P. Katoen, J. C. van de Pol, M. I. A. Stoelinga, and M. Timmer. A linear process-algebraic format for probabilistic systems with data. In *Proceedings of the 10th International Conference on Application of Concurrency to System Design (ACSD)*, pages 213–222. IEEE, 2010.

[KvdPST10b] J.-P. Katoen, J. C. van de Pol, M. I. A. Stoelinga, and M. Timmer. A linear process algebraic format for probabilistic systems with data (extended version). Technical Report TR-CTIT-10-11, Centre for Telematics and Information Technology, University of Twente, 2010.

[KvdPST12] J.-P. Katoen, J. C. van de Pol, M. I. A. Stoelinga, and M. Timmer. A linear process-algebraic format with data for probabilistic automata. *Theoretical Computer Science*, 413(1):36–57, 2012.

[KWW13] J. J. A. Keiren, W. Wesselink, and T. A. C. Willemse. Improved static analysis of parameterised boolean equation systems using control flow reconstruction. Technical Report 1304.6482, ArXiv e-prints, 2013.

[KZH$^+$11] J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, and D. N. Jansen. The ins and outs of the probabilistic model checker MRMC. *Performance Evaluation*, 68(2):90–104, 2011.

[Lam83] L. Lamport. What good is temporal logic? In *Proceedings of the IFIP 9th World Computer Congress on Information Processing*, pages 657–668. Elsevier, 1983.

[Lam06] L. Lamport. Checking a multithreaded algorithm with $^{+}$CAL. In *Proceedings of the 20th International Symposium on Distributed Computing (DISC)*, volume 4167 of *Lecture Notes in Computer Science*, pages 151–163. Springer, 2006.

[LDB10] A. Legay, B. Delahaye, and S. Bensalem. Statistical model checking: An overview. In *Proceedings of the 1st International Conference on Runtime Verification (RV)*, volume 6418 of *Lecture Notes in Computer Science*, pages 122–135. Springer, 2010.

[LLPY97] K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Efficient verification of real-time systems: compact data structure and state-space reduction. In *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS)*, pages 14–24. IEEE, 1997.

[LM09] F. Lang and R. Mateescu. Partial order reductions using compositional confluence detection. In *Proceedings of the 16th International Symposium on Formal Methods (FM)*, volume 5850 of *Lecture Notes in Computer Science*, pages 157–172. Springer, 2009.

[LMdV12] D. Latella, M. Massink, and E. P. de Vink. Bisimulation of labeled state-to-function transition systems of stochastic process languages. In *Proceedings of the 7th Workshop on Applied and Computational Category Theory (ACCAT)*, volume 93 of *Electronic Proceedings in Theoretical Computer Science*, pages 23–43. Open Publishing Association, 2012.

[LP12] R. Lassaigne and S. Peyronnet. Approximate planning and verification for large Markov decision processes. In *Proceedings of the 27th ACM Symposium on Applied Computing (SAC)*, pages 1314–1319. ACM, 2012.

[LS91] K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.

[LT89] N. A. Lynch and M. R. Tuttle. An introduction to Input/Output automata. *CWI Quarterly*, 2(3):219–246, 1989.

[LvdPW11] A. Laarman, J. C. van de Pol, and M. Weber. Parallel recursive state compression for free. In *Proceedings of the 18th International SPIN Workshop on Model Checking Software (SPIN)*, volume 6823 of *Lecture Notes in Computer Science*, pages 38–56. Springer, 2011.

[McM93] K. L. McMillan. *Symbolic Model Checking.* Kluwer, 1993.

[Mil80] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science.* Springer, 1980.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[MM99] C. Morgan and A. McIver. pGCL: formal reasoning for random algorithms. *South African Computer Journal*, 22:14–27, 1999.

[MW12] R. Mateescu and A. Wijs. Sequential and distributed on-the-fly computation of weak tau-confluence. *Science of Computer Programming*, 77(10-11):1075–1094, 2012.

[Nel95] R. D. Nelson. *Probability, Stochastic Processes, and Queueing Theory - the Mathematics of Computer Performance Modeling.* Springer, 1995.

[NK07] M. R. Neuhäußer and J.-P. Katoen. Bisimulation and logical preservation for continuous-time Markov decision processes. In *Proceedings of the 18th International Conference on Concurrency Theory (CONCUR)*, volume 4703 of *Lecture Notes in Computer Science*, pages 412–427. Springer, 2007.

[NM10] U. Ndukwu and A. McIver. An expectation transformer approach to predicate abstraction and data independence for probabilistic programs. In *Proceedings of the 8th Workshop on Quantitative Aspects of Programming Languages (QAPL)*, volume 28 of *Electronic Proceedings in Theoretical Computer Science*, pages 129–143. Open Publishing Association, 2010.

[NS06] G. Norman and V. Shmatikov. Analysis of probabilistic contract signing. *Journal of Computer Security*, 14(6):561–589, 2006.

[NSK09] M. R. Neuhäußer, M. I. A. Stoelinga, and J.-P. Katoen. Delayed nondeterminism in continuous-time Markov decision processes. In *Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures (FOSSACS)*, volume 5504 of *Lecture Notes in Computer Science*, pages 364–379. Springer, 2009.

[ORR$^+$96] S. Owre, S. Rajan, J. M. Rushby, N. Shankar, and M. K. Srivas. PVS: Combining specification, proof checking, and model checking. In *Proceedings of the 8th International Conference on Computer Aided Verification (CAV)*, volume 1102 of *Lecture Notes in Computer Science*, pages 411–414. Springer, 1996.

[Osa12] A. Osaiweran. *Formal Development of Control Software in the Medical Systems Domain.* PhD thesis, Eindhoven University of Technology, 2012.

[Pel93] D. Peled. All from one, one for all: on model checking using representatives. In *Proceedings of the 5th International Conference on Computer Aided Verification (CAV)*, volume 697 of *Lecture Notes in Computer Science*, pages 409–423. Springer, 1993.

[Pel98] D. Peled. Ten years of partial order reduction. In *Proceedings of the 10th International Conference on Computer Aided Verification (CAV)*, volume 1427 of *Lecture Notes in Computer Science*, pages 17–28. Springer, 1998.

[Pel08] R. Pelánek. Fighting state space explosion: Review and evaluation. In *Proceedings of the 13th International Workshop on Formal Methods for Industrial Critical Systems (FMICS)*, volume 5596 of *Lecture Notes in Computer Science*, pages 37–52. Springer, 2008.

[PIM$^+$06] G. Della Penna, B. Intrigila, I. Melatti, E. Tronci, and M. Venturini Zilli. Finite horizon analysis of Markov chains with the Murphi verifier. *International Journal on Software Tools for Technology Transfer*, 8(4-5):397–409, 2006.

[PIP13] Platform independent Petri net editor 2, 2013. `http://pipe2.sourceforge.net/`.

[PLM03] G. J. Pace, F. Lang, and R. Mateescu. Calculating τ-confluence compositionally. In *Proceedings of the 15th International Conference on Computer Aided Verification (CAV)*, volume 2725 of *Lecture Notes in Computer Science*, pages 446–459. Springer, 2003.

[Plo81] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

[PLS00] A. Philippou, I. Lee, and O. Sokolsky. Weak bisimulation for probabilistic systems. In *Proceedings of the 11th International Conference on Concurrency Theory (CONCUR)*, volume 1877 of *Lecture Notes in Computer Science*, pages 334–349. Springer, 2000.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 46–57. IEEE, 1977.

[Pri95] C. Priami. Stochastic pi-calculus. *The Computer Journal*, 38(7):578–589, 1995.

[PRI13] PRISM manual: The APMC method, 2013. `http://www.prismmodelchecker.org/manual/RunningPRISM/ApproximateModelChecking`.

[PS07] B. Ploeger and L. J. Somers. Analysis and verification of an automatic document feeder. In *Proceedings of the 2007 ACM Symposium on Applied Computing (SAC)*, pages 1499–1505. ACM, 2007.

[Put05] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, 2005.

[QS82] J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proceedings of the 5th International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 1982.

[QWP99] Q. Qiu, Q. Wu, and M. Pedram. Stochastic modeling of a power-managed system: construction and optimization. In *Proceedings of the 1999 International Symposium on Low Power Electronics and Design (ISLPED)*, pages 194–199. ACM, 1999.

[Ros11] S. M. Ross. *Introduction to Probability Models, 10th Revised Edition.* Academic Press, 2011.

[Seg95] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems.* PhD thesis, Massachusetts Institute of Technology, 1995.

[SL95] R. Segala and N. A. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computation*, 2(2):250–273, 1995.

[Sri91] M. M. Srinivasan. Nondeterministic polling systems. *Management Science*, 37(6):667–681, 1991.

[SSBM11] M. Sijtema, M. I. A. Stoelinga, A. F. E. Belinfante, and L. Marinelli. Experiences with formal engineering: Model-based specification, implementation and testing of a software bus at Neopost. In *Proceedings of the 16th International Workshop on Formal Methods for Industrial Critical Systems (FMICS)*, volume 6959 of *Lecture Notes in Computer Science*, pages 117–133. Springer, 2011.

[Sto02a] M. I. A. Stoelinga. *Alea jacta est: Verification of Probabilistic, Real-time and Parametric Systems.* PhD thesis, University of Nijmegen, 2002.

[Sto02b] M. I. A. Stoelinga. An introduction to probabilistic automata. *Bulletin of the EATCS*, 78:176–198, 2002.

[SvdZ98] C. Shankland and M. van der Zwaag. The tree identify protocol of IEEE 1394 in μCRL. *Formal Aspects of Computing*, 10(5-6):509–531, 1998.

[SZG11] L. Song, L. Zhang, and J. C. Godskesen. Bisimulations meet PCTL equivalences for probabilistic automata. In *Proceedings of the 22nd International Conference on Concurrency Theory (CONCUR)*, volume 6901 of *Lecture Notes in Computer Science*, pages 108–123. Springer, 2011.

[SZG12] L. Song, L. Zhang, and J. C. Godskesen. Late weak bisimulation for Markov automata. Technical report, ArXiv e-prints, 2012.

[Tar72] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[TBS11] M. Timmer, H. Brinksma, and M. I. A. Stoelinga. Model-based testing. In *Software and Systems Safety - Specification and Verification*, volume 30 of *NATO Science for Peace and Security Series - D: Information and Communication Security*, pages 1–32. IOS Press, 2011.

[Tim11] M. Timmer. SCOOP: A tool for symbolic optimisations of probabilistic processes. In *Proceedings of the 8th International Conference on Quantitative Evaluation of Systems (QEST)*, pages 149–150. IEEE, 2011.

[TKvdPS12a] M. Timmer, J.-P. Katoen, J. C. van de Pol, and M. I. A. Stoelinga. Efficient modelling and generation of Markov automata. In *Proceedings of the 23rd International Conference on Concurrency Theory (CONCUR)*, volume 7454 of *Lecture Notes in Computer Science*, pages 364–379. Springer, 2012.

[TKvdPS12b] M. Timmer, J.-P. Katoen, J. C. van de Pol, and M. I. A. Stoelinga. Efficient modelling and generation of Markov automata (extended version). Technical Report TR-CTIT-12-16, Centre for Telematics and Information Technology, University of Twente, 2012.

[TSvdP10] M. Timmer, M. I. A. Stoelinga, and J. C. van de Pol. Confluence reduction for probabilistic systems (extended version). Technical Report 1011.2314, ArXiv e-prints, 2010.

[TSvdP11] M. Timmer, M. I. A. Stoelinga, and J. C. van de Pol. Confluence reduction for probabilistic systems. In *Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 6605 of *Lecture Notes in Computer Science*, pages 311–325. Springer, 2011.

[TSvdP13a] M. Timmer, M. I. A. Stoelinga, and J. C. van de Pol. Confluence reduction for Markov automata. In *Proceedings of the 11th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, volume 8053 of *Lecture Notes in Computer Science*, pages 243–257. Springer, 2013.

[TSvdP13b] M. Timmer, M. I. A. Stoelinga, and J. C. van de Pol. Confluence reduction for Markov automata (extended version). Technical Report TR-CTIT-13-14, Centre for Telematics and Information Technology, University of Twente, 2013.

[Use02] Y. S. Usenko. *Linearization in μCRL*. PhD thesis, Eindhoven University of Technology, 2002.

[Val90] A. Valmari. Stubborn sets for reduced state space generation. In *Proceedings of the 10th International Conference on Applications and Theory of Petri Nets (ICATPN)*, volume 483 of *Lecture Notes in Computer Science*, pages 491–515. Springer, 1990.

[Val93] A. Valmari. On-the-fly verification with stubborn sets. In *Proceedings of the 5th International Conference on Computer Aided Verification (CAV)*, volume 697 of *Lecture Notes in Computer Science*, pages 397–408. Springer, 1993.

[Val96] A. Valmari. Stubborn set methods for process algebras. In *Proceedings of the DIMACS workshop on Partial order methods in verification (POMIV)*, pages 213–231. AMS Press, 1996.

[Var01] M. Y. Vardi. Branching vs. linear time: Final showdown. In *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2031 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2001.

[vdBRC06] P. C. W. van den Brand, M. A. Reniers, and P. J. L. Cuijpers. Linearization of hybrid processes. *Journal of Logic and Algebraic Programming*, 68(1-2):54–104, 2006.

[vdPT09a] J. C. van de Pol and M. Timmer. State space reduction of linear processes using control flow reconstruction. In *Proceedings of the 7th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, volume 5799 of *Lecture Notes in Computer Science*, pages 54–68. Springer, 2009.

[vdPT09b] J. C. van de Pol and M. Timmer. State space reduction of linear processes using control flow reconstruction. Technical Report TR-CTIT-09-24, Centre for Telematics and Information Technology, University of Twente, 2009.

[vGW96] R. J. van Glabbeek and W. P. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43(3):555–600, 1996.

[VvdP07] M. Valero Espada and J. C. van de Pol. An abstract interpretation toolkit for μCRL. *Formal Methods in System Design*, 30(3):249–273, 2007.

[Web06] M. Weber. *Parallel Algorithms for Verification of Large Systems*. PhD thesis, RWTH Aachen University, 2006.

[Wei84] M. Weiser. Program slicing. *IEEE Transactions on Software Engineering*, 10(4):352–357, 1984.

[WH00] B. D. Winters and A. J. Hu. Source-level transformations for improved formal verification. In *Proceedings of the 18th IEEE International Conference On Computer Design (ICCD)*, pages 599–602. IEEE, 2000.

[Wil11] D. J. Wilkinson. *Stochastic Modelling for Systems Biology*, volume 44 of *Chapman & Hall/CRC Mathematical & Computational Biology Series*. CRC Press, 2011.

[WLBF09] J. Woodcock, P. G. Larsen, J. Bicarregui, and J. S. Fitzgerald. Formal methods: Practice and experience. *ACM Computing Surveys*, 41(4), 2009.

[WW96] B. Willems and P. Wolper. Partial-order methods for model checking: From linear time to branching time. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 294–303. IEEE, 1996.

[YG04] K. Yorav and O. Grumberg. Static analysis for state-space reductions preserving temporal logics. *Formal Methods in System Design*, 25(1):67–96, 2004.

[YKNP06] H. L. S. Younes, M. Z. Kwiatkowska, G. Norman, and D. Parker. Numerical vs. statistical probabilistic model checking. *International Journal on Software Tools for Technology Transfer*, 8(3):216–228, 2006.

[YL92] W. Yi and K. G. Larsen. Testing probabilistic and nondeterministic processes. In *Proceedings of the 12th International Symposium on Protocol Specification, Testing and Verification (PSTV)*, IFIP Transactions, pages 47–61. North-Holland, 1992.

[YS02] H. L. S. Younes and R. G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *Proceedings of the 14th International Conference on Computer Aided Verification (CAV)*, volume 2404 of *Lecture Notes in Computer Science*, pages 223–235. Springer, 2002.

[ZN10] L. Zhang and M. R. Neuhäußer. Model checking interactive Markov chains. In *Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 6015 of *Lecture Notes in Computer Science*, pages 53–68. Springer, 2010.

Index

Samenvatting

Kwantitatief model checking houdt zich bezig met de verificatie van zowel kwantitatieve als kwalitatieve eigenschappen van modellen die kwantitatieve informatie bevatten. Een toename in expressiviteit van de modellen waar aan gerekend wordt zorgt ervoor dat meer soorten systemen geanalyseerd kunnen worden, maar vergroot ook de moeilijkheid van de berekeningen.

Drie jaar geleden werd de Markov-automaat (MA) geïntroduceerd. Dit model is een generalisatie van de probabilistische automaat (PA) en de interactieve Markov-keten (IMC). Het bevat nondeterminisme, discrete probabilistische keuze en stochastische tijd. Later werd bovendien de tool IMCA ontwikkeld, waarmee tijdsgebonden bereikbaarheidseigenschappen, tijdsverwachtingswaarden en langetermijngemiddelden bepaald kunnen worden voor een verzameling van doeltoestanden in een MA. Echter, tot nu toe was er nog geen efficiënt formalisme voor het modelleren en genereren van MAs. Bovendien loert de toestandsruimte-explosie altijd om de hoek, waardoor modellen al snel te groot worden om te analyseren. Dit proefschrift lost het eerste probleem op, en levert een significante bijdrage aan het verminderen van het tweede.

Ten eerste introduceren we de procesalgebraïsche taal MAPA voor het modelleren van MAs. Deze taal kan gebruikmaken van zowel statische als dynamische data (zoals lijsten), zodat systemen efficiënt gemodelleerd kunnen worden. Een transformatie van MAPA-specificaties naar een beperking van de taal—mogelijk gemaakt door een codering van Markoviaanse snelheden in acties—vereenvoudigt parallelle compositie, toestandsruimtegeneratie en syntactische optimalisaties (reductietechnieken).

Ten tweede introduceren we vijf reductietechnieken voor MAPA-specificaties: constanteneliminatie, expressievereenvoudiging, sommatie-eliminatie, dodevariabelenreductie en confluentiereductie. De eerste drie hebben als doel om de toestandsruimtegeneratie te versnellen door het vereenvoudigen van de specificatie, terwijl de laatste twee zich richten op efficiëntere analyse door een vermindering van het aantal toestanden. Dodevariabelenreductie reset datavariabelen op het moment dat hun waarde niet meer relevant is, en confluentiereductie detecteert en vermindert oneigenlijk nondeterminisme (dat vaak ontstaat door de parallelle compositie van beperkt samenwerkende componenten). Aangezien MAs zowel gelabelde transitiesystemen, Markov-ketens, Markov-processen, probabilistische automaten als interactive Markov-ketens generaliseren, zijn onze technieken en resultaten ook van toepassen op al deze deelmodellen.

Ten derde vergelijken we confluentiereductie met de ‘ample set’-variant van partial order reduction (POR). Aangezien POR nog niet gedefinieerd was voor MAs, vindt onze vergelijking plaats in de context van probabilistische automaten.

We stellen exact vast hoe de twee methoden theoretisch verschillen, en lossen daarmee een langdurige onzekerheid op over hoe deze concepten verband houden. Als we ons richten op 'branching-time' eigenschappen, blijkt confluentiereductie strikt krachtiger te zijn dan POR. Aansluitend hierop vergelijken we de twee technieken ook in praktische zin, in de context van statistische model checking. We demonstreren dat de aanvullende kracht van confluentie inderdaad tot extra reducties kan leiden (zelfs vergeleken met de POR-methode die slechts 'linear-time' eigenschappen bewaart).

We hebben een tool genaamd SCOOP ontwikkeld, waarin alle zojuist beschreven reductietechnieken zijn verwerkt. SCOOP kan exporteren naar de IMCA-tool, en vormt samen daarmee de eerste softwareketen voor de analyse van MAs. Case studies omtrent een handshake-register, een verkiezingsprotocol, een polling-systeem en een processorarchitectuur demonstreren de variëteit aan systemen die gemodelleerd kunnen worden met MAPA. Experimentele resultaten laten bovendien zien dat significante reducties behaald kunnen worden met onze technieken, waarmee de toestandsruimten soms krimpen tot minder dan een procent van hun oorspronkelijke grootte. Onze resultaten stellen ons bovendien in staat om richtlijnen te geven die voor iedere techniek aangeven wat voor eigenschappen in een case study indicatoren zijn voor grote reducties.

Uiteindelijk biedt MAPA ons de mogelijkheid om efficiënt systemen te modelleren met nondeterminisme, discrete probabilistische keuze en stochastische tijd. De taal staat ons bovendien toe om geavanceerde reductietechnieken eenvoudig te definiëren, wat inderdaad tot een aantal nieuwe technieken heeft geleid. Onze vergelijkingen tussen confluentiereductie en POR verschaffen nieuwe inzichten in hun relatie. Experimenten laten bovendien zien dat onze technieken van grote waarde kunnen zijn ter voorkoming van de toestandsruimte-explosie: een belangrijke stap voorwaards in efficiënte kwantitatieve verificatie.

Kan je dit ook uitleggen aan niet-informatici?

Het laatste gedeelte van dit proefschrift is bedoeld voor mijn oma—en voor alle andere lezers die weinig weten van informatica, maar wel enigszins benieuwd zijn wat ik de afgelopen jaren heb gedaan[1]. Ik zal beginnen met een versimpelde inleiding in mijn vakgebied, en vervolgens kort proberen uit te leggen wat mijn bijdrage hieraan is geweest. In de praktijk worden de technieken waar wij aan werken vaak gebruikt voor de kwaliteitsverbetering van ingewikkelde en essentiële computersystemen. Ik heb de voorbeelden echter bewust eenvoudig en binnen de belevingswereld van velen gehouden, om het één en ander begrijpelijker uit te kunnen leggen dan als ik realistische voorbeelden zou hebben gebruikt.

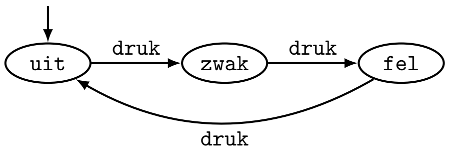
[1]Met dank aan 'wiskundemeisje' Ionica Smeets voor de tip om hier op het eind toch nog even wat aandacht aan te besteden.

Model checking

Mijn werk speelt zich af in het werkveld *model checking*, een onderdeel van de informatica. Om te begrijpen waar model checking goed voor is, gebruik ik het voorbeeld van de kerncentrale versus het boekhoudprogramma. Bij een boekhoudprogramma werkt het prima om een eerste versie te produceren, vervolgens te testen in hoeverre deze versie correct werkt, en op basis hiervan verbeteringen aan te brengen. Eventuele resterende fouten die tijdens het gebruik aan het licht komen kunnen wellicht door middel van updates verholpen worden. Bij een kerncentrale (of bijvoorbeeld een raket) is deze methode minder wenselijk; een klein foutje kan dan al snel leiden tot desastreuze gevolgen. Het is daarom essentieel om correctheid van de belangrijkste componenten van een dergelijk systeem zo goed mogelijk vooraf te garanderen.

In de informatica probeert men onder andere bij te dragen aan zulke garanties door middel van model checking. Deze aanpak bestaat uit het wiskundig modelleren van een systeem en het wiskundig formuleren van een aantal correctheidseigenschappen. Vervolgens kan een *model checker* (een speciaal computerprogramma) gebruikt worden om automatisch te controleren of het model aan alle eigenschappen voldoet. Vaak maken we hiervoor gebruik van gedragsmodellen die precies alle mogelijke toestanden van een systeem beschrijven, evenals alle overgangen tussen deze toestanden. Het model wordt daarom ook vaak een *toestandsruimte* genoemd.

Een eenvoudig voorbeeld van een toestandsruimte wordt weergegeven in het volgende plaatje. Het betreft een lamp met drie mogelijke standen: uit, zwak en fel. Door op een knop te drukken springt de lamp naar de volgende stand. De bolletjes representeren de toestanden van het systeem, terwijl de pijlen de acties van de gebruiker modelleren. De pijl 'uit het niets' linksboven wijst naar de initiële toestand van het systeem.



Een mogelijke correctheidseigenschap zou kunnen zijn: "het is altijd mogelijk om de lamp uit te schakelen". In dit model komt dat neer op de eis dat het vanuit iedere toestand mogelijk is om de toestand `uit` te bereiken; er moet altijd een pad (van mogelijk meerdere stappen) zijn naar deze toestand. Een model checker zou dit automatisch kunnen verifiëren. Uiteraard kunnen we in dit eenvoudige voorbeeld zelf ook direct zien dat inderdaad aan deze eigenschap wordt voldaan.

In mijn werk houd ik me bezig met ingewikkeldere modellen, waarin ook kwantitatieve aspecten (kansen en tijd) verwerkt zijn. Hier kom ik later op terug.

Toestandsruimte-explosie

In de praktijk zijn modellen van systemen vaak ontzettend groot; een paar miljoen toestanden is zeker niet ongebruikelijk, en realistische systemen komen al snel uit op miljarden of zelfs nog meer mogelijke toestanden. Dit is voornamelijk te wijten aan het feit dat systemen vaak uit verschillende componenten bestaan, die zich allemaal in meerdere toestanden kunnen bevinden. De combinatie van deze componenten leidt tot de zogeheten toestandsruimte-explosie (*state space explosion*).

Om de aard van de toestandsruimte-explosie te begrijpen, beschouwen we het aantal manieren om een diner samen te stellen op basis van een menukaart. Er zijn drie gangen: we hebben vier keuzes voor het voorgerecht, zes keuzes voor het hoofdgerecht en drie keuzes voor het toetje. Hoeveel mogelijkheden zijn er nu in totaal? Bij ieder van de vier voorgerechten kan je uit zes hoofdgerechten kiezen. Dit maakt daarom $4 \times 6 = 24$ combinaties van voorgerecht en hoofdgerecht. Voor ieder van deze combinaties kan je bovendien uit drie toetjes kiezen; dat resulteert in $24 \times 3 = 72$ mogelijkheden in totaal. Dit is al een behoorlijke hoeveelheid. Als we nu nog een extra keuze uit twee soorten amuses toevoegen, verdubbelt het aantal mogelijke menu's naar $72 \times 2 = 144$. Voor iedere extra gang vindt een soortgelijke verdubbeling of wellicht verdrievoudiging of verviervoudiging plaats in het aantal mogelijkheden.

Precies hetzelfde is er aan de hand bij het modelleren van een systeem dat uit meerdere componenten bestaat: het totale aantal toestanden (de bolletjes in het model) is grofweg de *vermenigvuldiging* van het aantal toestanden van de individuele componenten. Even voor het idee: een combinatie van 10 systemen die ieder uit 100 toestanden bestaan, levert

$$100^{10} = \underbrace{100 \cdot 100 \cdot \ldots \cdot 100}_{\text{10 keer}} = 100.000.000.000.000.000.000$$

toestanden op: het astronomische aantal van een 1 met 20 nullen. Dit is zo'n tien keer zoveel als het aantal zandkorrels op alle stranden van de wereld bij elkaar (volgens een grove schatting). Als een computer een miljoen toestanden per seconde zou doorrekenen, zou deze hier ruim drie miljoen jaar mee aan het werk zijn. Het feit dat het aantal toestanden zo snel uit de hand loopt is precies wat wordt bedoeld met de toestandsruimte-explosie. Aangezien een combinatie van 10 systemen die ieder uit 100 toestanden bestaan helemaal niet onrealistisch is, vormt deze eigenaardigheid een van de belangrijkste problemen in ons vakgebied—systemen in de praktijk hebben vaak een dusdanig grote toestandsruimte dat analyse niet meer mogelijk is binnen afzienbare tijd.

Procesalgebra

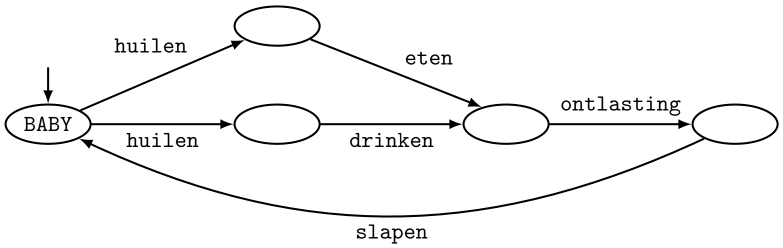
We hebben zojuist gezien dat modellen al snel onhandelbaar groot kunnen worden. Toch zijn er gelukkig ook genoeg nuttige (deel)systemen of protocollen die efficiënt gemodelleerd kunnen worden in bijvoorbeeld een miljoen toestanden of zelfs nog minder. Modellen van zo'n formaat kunnen met moderne computers nog prima doorgerekend worden. Echter, het is natuurlijk niet zo prettig om handmatig een model van bijvoorbeeld 500.000 toestanden op te schrijven.

Onze aanpak is daarom om bij een systeem dat uit meerdere componenten bestaat ook daadwerkelijk de individuele componenten van een systeem los van elkaar te modelleren. Hiervoor wordt een bepaald wiskundig formalisme (een *procesalgebra*) gebruikt, waarbij we aan kunnen geven hoe deze componenten met elkaar communiceren. Vergelijk het maar met de eerdergenoemde menukaart: hier willen we ook niet alle 144 mogelijke menu's onder elkaar noteren. In plaats daarvan kunnen alle keuzemogelijkheden toch beknopt worden opgeschreven door gewoon de componenten (in dit geval de gangen) los van elkaar te beschrijven en dus de 15 mogelijke gerechten onder elkaar op te schrijven. Zo wordt de toestandsruimte-explosie (de vermenigvuldiging) nog even uitgesteld. Op deze manier kunnen reusachtige systemen vaak op zeer beknopte wijze gemodelleerd worden; het komt regelmatig voor dat systemen met miljoenen toestanden gerepresenteerd worden door minder dan een A4'tje procesalgebra. Een computerprogramma kan dan automatisch het volledige model genereren op basis van zo'n beschrijving.

In een procesalgebra wordt het gedrag van processen gerepresenteerd. Je zou het kunnen zien als een wiskundige taal. In principe kan bijna alles als proces worden opgevat: een computerprogramma, een protocol, een spelletje of zelf een persoon. Simpel gezegd geeft de beschrijving de volgorde van de mogelijke acties van het proces aan. Een (voor de leesbaarheid zeer vereenvoudigd) voorbeeld van het gedrag van het proces `BABY` zou als volgt kunnen zijn:

$$\texttt{BABY} = (\texttt{huilen} \cdot \texttt{eten} + \texttt{huilen} \cdot \texttt{drinken}) \cdot \texttt{ontlasting} \cdot \texttt{slapen} \cdot \texttt{BABY}$$

In deze procesalgebraïsche beschrijving is `BABY` de naam van het proces dat beschreven wordt. De verschillende woorden geven de acties aan. De punt wordt gebruikt voor *sequentiële compositie* (volgorde): acties vinden na elkaar plaats van links naar rechts. Slapen komt dus bij deze baby altijd pas na ontlasting. De plus wordt gebruikt voor *alternatieve compositie* (keuze): ofwel het gedrag links van de plus wordt uitgevoerd, ofwel het gedrag rechts ervan. In dit geval wordt dus aangenomen dat een baby eerst huilt en eet, of eerst huilt en drinkt, en dan zijn behoefte doet en gaat slapen. Het woord `BABY` op het eind geeft aan dat het gedrag zich vervolgens weer herhaalt, wat leidt tot de volgende toestandsruimte:

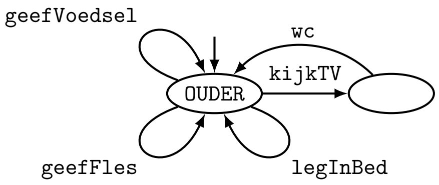


Hoewel de procesalgebraïsche beschrijving geen namen geeft aan de tussenliggende toestanden, zijn deze in dit geval prima te bedenken (honger, dorst, vol, moe). We zien in dit plaatje duidelijk dat de actie `huilen` tot twee verschillende toestanden kan leiden: één waarin de baby honger heeft en één waarin de baby dorst heeft. Dit fenomeen wordt *non-determinisme* genoemd: als ouder moet je maar gokken in welke toestand je baby zich bevindt, dat is niet af te leiden uit de observeerbare acties (in ieder geval niet volgens deze vereenvoudigde representatie).

Procesalgebra staat naast sequentiële en alternatieve compositie ook *parallelle compositie* (gelijktijdige uitvoering) toe. Dit is de grote kracht van het paradigma, aangezien hiermee op eenvoudige wijze deelsystemen samengesteld kunnen worden tot een totaalsysteem waarin allerlei gedrag van de deelsystemen onafhankelijk van elkaar kan gebeuren. Als illustratie modelleren we naast het gedrag van de baby ook het (wederom extreem vereenvoudigde) gedrag van een van zijn ouders:

$$\mathtt{OUDER} = (\mathtt{geefVoedsel} + \mathtt{geefFles} + \mathtt{legInBed} + \mathtt{kijkTV} \cdot \mathtt{wc}) \cdot \mathtt{OUDER}$$

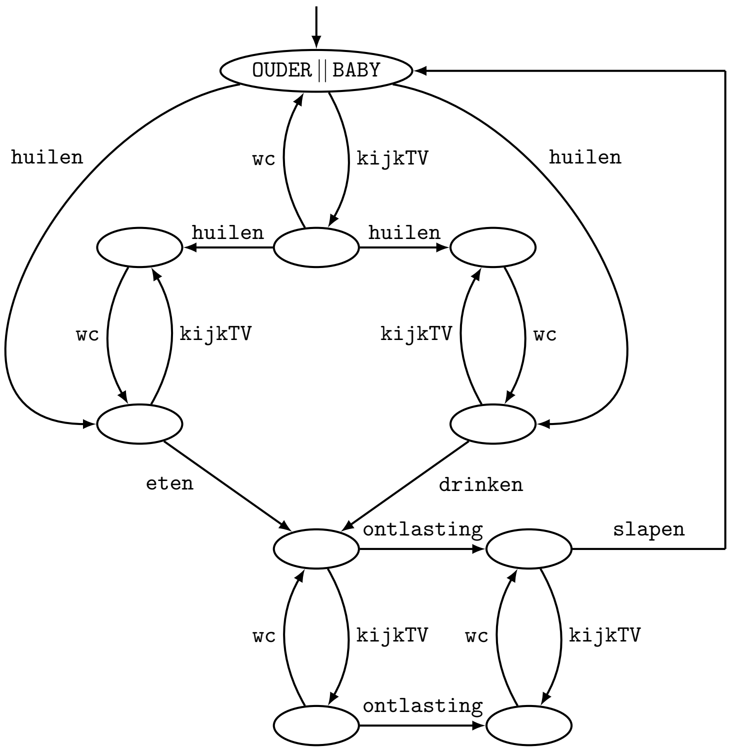
Deze ouder kan de baby voeden, de fles geven en in bed leggen. De resterende tijd besteedt hij aan televisie kijken, wat altijd direct gevolgd wordt door een bezoek aan het toilet. (Uiteraard is dit geen realistische weergave van de werkelijkheid, aangezien ouders onder andere ook luiers moeten verschonen, met kinderen spelen en zelf moeten eten en slapen, maar dat laten we even buiten beschouwing om de modellen niet te ingewikkeld te maken). De eenvoudige beschrijving van de ouder leidt tot de volgende toestandsruimte:



Merk op dat er slechts twee toestanden zijn: normaal gesproken kan deze ouder de baby voeden, de fles geven of in bed stoppen, tenzij hij al begonnen is met televisie kijken. In dat geval moet hij eerst naar de wc voordat hij zich weer om de baby kan bekommeren.

We kunnen deze twee processen nu in parallel zetten; we verkrijgen hiermee het totale gedrag van de ouder en de baby als eenheid. In overeenkomst met de werkelijkheid, eisen we *synchronisatie* van de acties `eten` (van `BABY`) en `geefFles` (van `OUDER`); deze twee acties moeten dan altijd tegelijk gebeuren. Een baby kan immers niet eten als de ouder niet aan het voeden is. Hetzelfde geldt voor drinken en de fles geven, en voor slapen en in bed stoppen. De parallelle compositie van beide processen ziet er dan uit zoals weergegeven in figuur 2.1 op de volgende pagina.

Het moge duidelijk zijn dat we blij zijn dat de computer dergelijke parallelle composities voor ons kan construeren op basis van de veel eenvoudigere procesalgebraïsche beschrijvingen. In mijn werk heb ik een nieuwe procesalgebra ontwikkeld waarmee eenvoudig grote modellen van geavanceerde systemen gegenereerd kunnen worden.

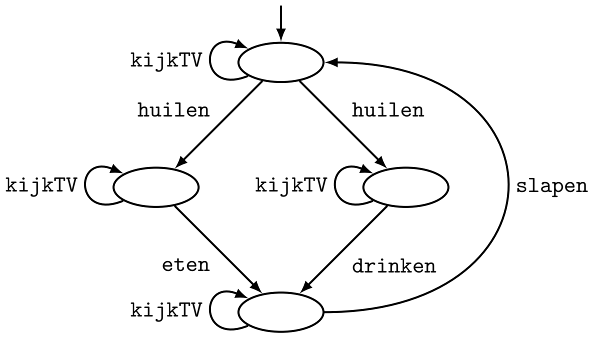


Figuur 2.1: Toestandsruimte van de parallelle compositie van de (zeer abstracte) representaties van een baby en zijn ouder.

Reductietechnieken

Zoals gezegd worden toestandsruimten al snel ontzettend groot. Hoewel procesalgebra's ons helpen om dergelijke systemen toch nog beknopt te kunnen representeren, is het boven bepaalde grenzen niet meer mogelijk om analyses uit te voeren op de verkregen modellen—het resultaat van de toestandsruimte-explosie. De afgelopen decennia is er veel onderzoek gedaan naar methodes om de toestandsruimte-explosie tegen te gaan. Er is gewerkt aan efficiëntere opslag van de modellen, aan het combineren van meerdere computers tot een grote rekenmachine die meer aan kan dan iedere computer individueel, en er is gewerkt aan methodes (*reductietechnieken*) die ervoor zorgen dat we slechts een *gedeelte* van de toestandsruimte hoeven te genereren. Deze laatstgenoemde aanpak is degene die in dit proefschrift veelvuldig wordt toegepast.

Om een reductietechniek te illustreren komen we terug op het voorbeeld van de baby en de ouder. Het model hierboven geeft de volgordes van alle acties precies weer. In de praktijk komt het echter regelmatig voor dat we uiteindelijk slechts geïnteresseerd zijn in een kleine selectie van de acties van het uiteindelijke model; de rest was slechts benodigd ter constructie van het model. In dat geval



Figuur 2.2: Gereduceerde toestandsruimte van de parallelle compositie van de (zeer abstracte) representaties van een baby en zijn ouder.

kunnen we een aantal acties *verbergen*. Het is vervolgens mogelijk om het model automatisch te verkleinen tot een vergelijkbaar model, waarin de acties die niet verborgen zijn nog steeds in de juiste volgorde te vinden zijn, maar de verborgen acties zoveel mogelijk weggewerkt zijn.

Stel nu dat we alleen geïnteresseerd zijn in het verband tussen de acties `kijkTV`, `huilen`, `drinken`, `eten` en `slapen`. Het blijkt dan mogelijk om het model van figuur 2.1 automatisch te reduceren tot het model in figuur 2.2. Nu is het model een stuk kleiner en overzichtelijker geworden, terwijl alle informatie betreffende de acties `kijkTV`, `huilen`, `drinken`, `eten` en `slapen` nog steeds aanwezig is. Zo is snel te zien dat de baby eerst huilt, dan eet of drinkt en als laatste slaapt. Ook is duidelijk dat de ouder volgens de versimpelde beschrijving op ieder moment kan besluiten om tv te kijken en de baby te negeren—uiteraard is dat in de praktijk echter niet de beste keuze!

In mijn werk heb ik me veel beziggehouden met de ontwikkeling van nieuwe reductietechnieken, waarbij gecompliceerde modellen automatisch gereduceerd kunnen worden tot kleinere modellen met nog wel dezelfde eigenschappen.

Kwantitatieve modellen

In het bovenstaande ben ik telkens uitgegaan van 'ouderwetse' modellen (de zogeheten *gelabelde transitiesystemen* vanwege hun transities die gelabeld zijn met actienamen). De enige informatie die dit type modellen bevat is de onderlinge volgorde van de acties. In de laatste jaren is er echter steeds meer aandacht gekomen voor *kwantitatieve modellen*. Dergelijke modellen bevatten naast informatie over de volgorde van acties eventueel ook gegevens over tijdsduur en/of waarschijnlijkheid. Zo zou ons babyvoorbeeld uitgebreid kunnen worden door informatie toe te voegen over de lengte van een gemiddelde voeding. Bovendien zou de keuze tussen eten en drinken verder ingevuld kunnen worden; waar we eerder aannamen dat een baby nou eenmaal wil eten of drinken (non-determinisme), zouden we ook specifieker kunnen zeggen dat beide gevallen een kans van 50% hebben als een baby huilt.

De toevoeging van kansen maakt het mogelijk om processen te modelleren die zich niet altijd op dezelfde manier gedragen, zoals een geboorte (ongeveer 50% kans op een jongen) of het versturen van een ansichtkaart vanuit een ver land naar Nederland (enige kans dat de kaart kwijtraakt). Bovendien kunnen we voor dit soort modellen ook *kwantitatieve eigenschappen* formuleren en deze door een model checker op waarheid laten controleren. Te denken aan eigenschappen zoals “de kans dat het programma een correct resultaat produceert is tenminste 99%”.

De toevoeging van tijdsinformatie maakt het mogelijk om eigenschappen te controleren betreffende prestaties en betrouwbaarheid: “de kans dat de cassières gezamenlijk minder dan 100 klanten in een uur kunnen helpen is kleiner dan 5%”, of “de kans dat het systeem pas kapot gaat meer dan 1,000 uur gebruik is tenminste 95%”.

Mijn bijdrage

Door het lezen van het bovenstaande heeft u een klein kijkje gekregen in de keuken van een gedeelte van mijn vakgebied. Tot zover is alles nog redelijk standaard, in de zin dat de meeste mensen in mijn vakgebied dit al weten en er ook niks genoemd is wat door mij bedacht is. Hoofdstuk 2 en 3 vatten deze achtergrondinformatie samen op een wiskundigere manier.

Wat heb ik dan wel zelf gedaan? Mijn bijdrage is grofweg op te splitsen in drie delen.

Nieuwe procesalgebra. Ten eerste heb ik een nieuwe procesalgebra (MAPA) ontwikkeld om modellen te genereren die zowel non-determinisme, kansen als tijdsduren bevatten (de zogeheten *Markov-automaten*). Ik heb voortgeborduurd op een eerdere niet-kwantitatieve aanpak die gebruikmaakte van *data*: een methode om op een efficiëntere manier nog complexere toestandsruimtes te kunnen genereren. Tot dusver bestond er nog geen procesalgebra die dit kon doen; het was daarom zonder mijn werk nog niet mogelijk om op een eenvoudige wijze Markov-automaten te genereren.

Ik heb bovendien een variant op MAPA ontwikkeld (de MLPE), die erg beperkt is en daarom voor computers handiger is om mee te werken. Zo heeft de MLPE geen parallelle compositie en moet iedere actie direct gevolgd worden door een procesnaam—onze voorbeelden hierboven zouden dus al niet eens voldoen. Aangezien de volledige procesalgebra handiger is voor mensen om mee te werken, heb ik een algoritme (een soort recept) ontwikkeld dat automatisch ieder model van de gebruiksvriendelijke procesalgebra MAPA omzet in een simpelere representatie in de taal MLPE. Daar kan de computer vervolgens makkelijker mee overweg.

Dit alles is beschreven in hoofdstuk 4.

Nieuwe reductietechnieken. Ten tweede heb ik verscheidene reductietechnieken ontwikkeld, die ervoor zorgen dat het aantal toestanden van een model binnen de perken blijft. Hoewel het theoretisch mogelijk zou zijn om eerst de originele toestandsruimte van een systeem te genereren, is dat natuurlijk niet zo efficiënt. Mijn aanpak maakt daarom gebruik van de eenvoud van de

MLPE (een procesalgebraïsche beschrijving). Er worden allerlei analyses op deze beknopte representaties uitgevoerd, waarna ze automatisch herschreven worden tot specificaties die uiteindelijk een veel kleinere toestandsruimte opleveren. Zo proberen we de toestandsruimte-explosie enigszins te omzeilen.

Om een idee te krijgen van hoe dergelijke technieken werken pakken we nog eenmaal de menukaart erbij. Stel dat er aan de kassa berekend moet worden wat een gekozen menu kost, en dat we weten dat alle toetjes even duur zijn. In dat geval maakt het niets uit welk toetje er gekozen wordt, en kunnen we net zo goed doen alsof iedereen de verse vruchtjes heeft genomen. Dat beperkt het aantal mogelijkheden, en maakt het model van de situatie kleiner (terwijl het voor de eigenschappen die bekeken worden—in dit geval de totale kosten—nog identiek functioneert).

Drie van zulke door mij ontwikkelde technieken worden besproken in hoofdstuk 4, en twee uitgebreidere technieken in hoofdstuk 5 en hoofdstuk 6.

Vergelijking van reductietechnieken. Ten derde heb ik een van de door mij ontwikkelde technieken (*confluence reduction*) vergeleken met een eerdere reductietechniek voor kwantitatieve systemen (*partial order reduction*). Een overduidelijk voordeel van mijn techniek is dat deze de enige techniek in zijn soort is die werkt voor systemen met zowel kansen als tijdsduren. Dus, als een dergelijk model gebruikt wordt, is er geen andere keus—partial order reduction werkt alleen als er hooguit kansen gebruikt worden, maar niet in de aanwezigheid van tijdsduren. Om toch meer inzicht te krijgen in het verband tussen de technieken, heb ik gekeken in hoeverre ze overeenkomen als we ons beperken tot systemen zonder tijdsduren. Ik heb bovendien precies de verschillen ertussen geïdentificeerd. Hierdoor is ook direct duidelijk hoe de niet-kwantitatieve varianten van de technieken (die al wel eerder bestonden) relateren; dat was ook nog niet bekend. Het blijkt dat mijn techniek net iets beter presteert voor systemen die al dan niet voorzien zijn van kansen—onder de aanname dat we een bepaalde verzameling van eigenschappen willen bewaren. De vergelijkingen tussen confluence reduction en partial order reduction zijn te vinden in hoofdstuk 7 en hoofdstuk 8.

Het proefschrift wordt afgerond in hoofdstuk 9 met een bespreking van het programma dat ik heb gemaakt om automatisch en efficiënt modellen met kansen en tijdsduren te genereren op basis van procesalgebraïsche beschrijvingen. Ook bespreek ik daar case studies die laten zien dat mijn procesalgebra hier inderdaad prima voor gebruikt kan worden, en laat ik zien dat mijn reductietechnieken ook in de praktijk voor aanzienlijke snelheidswinst zorgen.

M. A. Abam. *New Data Structures and Algorithms for Mobile Data*. Faculty of Mathematics and Computer Science, TU/e. 2007-19

W. Pieters. *La Volonté Machinale: Understanding the Electronic Voting Controversy*. Faculty of Science, Mathematics and Computer Science, RU. 2008-01

A.L. de Groot. *Practical Automaton Proofs in PVS*. Faculty of Science, Mathematics and Computer Science, RU. 2008-02

M. Bruntink. *Renovation of Idiomatic Crosscutting Concerns in Embedded Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-03

A.M. Marin. *An Integrated System to Manage Crosscutting Concerns in Source Code*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2008-04

N.C.W.M. Braspenning. *Model-based Integration and Testing of High-tech Multi-disciplinary Systems*. Faculty of Mechanical Engineering, TU/e. 2008-05

M. Bravenboer. *Exercises in Free Syntax: Syntax Definition, Parsing, and Assimilation of Language Conglomerates*. Faculty of Science, UU. 2008-06

M. Torabi Dashti. *Keeping Fairness Alive: Design and Formal Verification of Optimistic Fair Exchange Protocols*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2008-07

I.S.M. de Jong. *Integration and Test Strategies for Complex Manufacturing Machines*. Faculty of Mechanical Engineering, TU/e. 2008-08

I. Hasuo. *Tracing Anonymity with Coalgebras*. Faculty of Science, Mathematics and Computer Science, RU. 2008-09

L.G.W.A. Cleophas. *Tree Algorithms: Two Taxonomies and a Toolkit*. Faculty of Mathematics and Computer Science, TU/e. 2008-10

I.S. Zapreev. *Model Checking Markov Chains: Techniques and Tools*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-11

M. Farshi. *A Theoretical and Experimental Study of Geometric Networks*. Faculty of Mathematics and Computer Science, TU/e. 2008-12

G. Gulesir. *Evolvable Behavior Specifications Using Context-Sensitive Wildcards*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-13

F.D. Garcia. *Formal and Computational Cryptography: Protocols, Hashes and Commitments*. Faculty of Science, Mathematics and Computer Science, RU. 2008-14

P. E. A. Dürr. *Resource-based Verification for Robust Composition of Aspects*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2008-15

E.M. Bortnik. *Formal Methods in Support of SMC Design*. Faculty of Mechanical Engineering, TU/e. 2008-16

R.H. Mak. *Design and Performance Analysis of Data-Independent Stream Processing Systems*. Faculty of Mathematics and Computer Science, TU/e. 2008-17

M. van der Horst. *Scalable Block Processing Algorithms*. Faculty of

T. Dimkov. *Alignment of Organizational Security Policies: Theory and Practice.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-04

S. Sedghi. *Towards Provably Secure Efficiently Searchable Encryption.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-05

F. Heidarian Dehkordi. *Studies on Verification of Wireless Sensor Networks and Abstraction Learning for System Inference.* Faculty of Science, Mathematics and Computer Science, RU. 2012-06

K. Verbeek. *Algorithms for Cartographic Visualization.* Faculty of Mathematics and Computer Science, TU/e. 2012-07

D.E. Nadales Agut. *A Compositional Interchange Format for Hybrid Systems: Design and Implementation.* Faculty of Mechanical Engineering, TU/e. 2012-08

H. Rahmani. *Analysis of Protein-Protein Interaction Networks by Means of Annotated Graph Mining Algorithms.* Faculty of Mathematics and Natural Sciences, UL. 2012-09

S.D. Vermolen. *Software Language Evolution.* Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2012-10

L.J.P. Engelen. *From Napkin Sketches to Reliable Software.* Faculty of Mathematics and Computer Science, TU/e. 2012-11

F.P.M. Stappers. *Bridging Formal Models – An Engineering Perspective.* Faculty of Mathematics and Computer Science, TU/e. 2012-12

W. Heijstek. *Software Architecture Design in Global and Model-Centric Software Development.* Faculty of Mathematics and Natural Sciences, UL. 2012-13

C. Kop. *Higher Order Termination.* Faculty of Sciences, Department of Computer Science, VUA. 2012-14

A. Osaiweran. *Formal Development of Control Software in the Medical Systems Domain.* Faculty of Mathematics and Computer Science, TU/e. 2012-15

W. Kuijper. *Compositional Synthesis of Safety Controllers.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-16

H. Beohar. *Refinement of Communication and States in Models of Embedded Systems.* Faculty of Mathematics and Computer Science, TU/e. 2013-01

G. Igna. *Performance Analysis of Real-Time Task Systems using Timed Automata.* Faculty of Science, Mathematics and Computer Science, RU. 2013-02

E. Zambon. *Abstract Graph Transformation – Theory and Practice.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2013-03

B. Lijnse. *TOP to the Rescue – Task-Oriented Programming for Incident Response Applications.* Faculty of Science, Mathematics and Computer Science, RU. 2013-04

G.T. de Koning Gans. *Outsmarting Smart Cards.* Faculty of Science, Mathematics and Computer Science, RU. 2013-05

M.S. Greiler. *Test Suite Comprehension for Modular and Dynamic*

Quantitative model checking is an important research area, aimed at analysing performance and dependability aspects of a wide variety of systems. Applications include biological systems, power management controllers, communication protocols and security algorithms. The scope of quantitative model checking is limited mostly by the expressivity of the underlying modelling formalism and the omnipresent state space explosion.

This thesis contributes to the solution of both problems, building on the notion of Markov automata. First, we introduce a novel process-algebraic language that allows the modelling of systems incorporating nondeterminism, stochastic timing and discrete probabilistic choice. Second, we define and investigate several reduction techniques to speed-up state space generation as well as model checking.

Case studies show that our techniques greatly reduce the impact of the state space explosion: a major step forward in efficient quantitative verification.