Reasoning about Time, Action and Knowledge in Multi-Agent Systems

Ji Ruan

Ph.D. Thesis

UNIVERSITY OF LIVERPOOL

Reasoning about Time, Action and Knowledge in Multi-Agent Systems

A thesis submitted in accordance with the
requirements of the University of Liverpool
for the degree of Doctor of Philosophy by

Ji Ruan

Department of Computer Science
University of Liverpool
December 2008

To my grandmother Yaluo He (1925 - 2008)

First Supervisor:	Wiebe van der Hoek, University of Liverpool, UK
Second Supervisor:	Michael Wooldridge, University of Liverpool, UK
Advisor:	Clare Dixon, University of Liverpool, UK
Internal Examiner:	Clare Dixon, University of Liverpool, UK
External Examiner:	Alessio Lomuscio, Imperial College London, UK

Acknowledgements

This section is to thank the people who have helped me on the journey of completing my thesis. I owe my greatest appreciation to all of them.

More specifically, I would like to thank my first supervisor Prof. Wiebe van der Hoek and my second supervisor Prof. Michael Wooldridge. Both of them have been extremely helpful and kind to me. I started to work with Wiebe since I was in New Zealand in 2005; later he helped greatly in getting funding for my PhD study. He has been always patient and made time available for me whenever I wanted to discuss research or even personal problems. He does not only give high-level guidance, but also pays attention to the details. I clearly remember we had some lengthy discussions and he went into great depth of the problems with me without complaining that it should be my own work. Such meetings could go beyond the usual working hours. Apart from our professional relationship, Wiebe and his wife Marta also treated me as a real friend. They invited me to their cozy house on several occasions, and also to their wonderful wedding. Mike has been in most of the supervisory meetings for me with Wiebe. I am very impressed by his overall view and understanding of the research area of Multi-Agent Systems. It was Mike who suggested me the topic of General Game Playing and Strategic Logics. The research followed in this direction makes up a considerable part of my PhD work. His guidance has been always precise, concise and prompt. And he also made connections between me and other researchers in the area.

I thank Dr. Hans van Ditmarsch for his kind support all years along, since our first collaboration in the project of Cryptography for Ideal Agents in 2005, at the University of Otago, New Zealand. Part of my PhD work was also with his guidance and his great help. During my years in Liverpool, our communication was mainly over Internet or telephone. I am very impressed by his pages-long email comments and his pleasant tone over the phone. His short stays in Liverpool also provided me inspirations. Moreover he is a real friend in need.

I thank my other two collaborators, Prof. Rineke Verbrugge and Prof. Ron

van der Meyden, for their stimulating input. Part of our respective joint work is presented in this thesis, and is acknowledged separately in Chapter 1.

I thank Dr. Clare Dixon and Dr. Alessio Lomuscio for being my examiners, and giving me helpful comments and suggestions on my whole thesis. They also made my viva to happen in a friendly, open and critical atmosphere. Clare has been my advisor for the past three years, and provided continuous support to my work.

I thank all of the colleagues who read and commented on various pieces of my PhD or related work: Dr. Dirk Walther for numerous discussions on ATL and on research in general, Dr. Nicolas Troquard for commenting main chapters of my thesis, Prof. Johan van Benthem for the discussions when I visited Amsterdam and attended the LORI'07 Beijing workshop, Prof. Jan van Eijck for further development on the work I did for my master's thesis, Prof. Michael Thielscher for comments on GDL, and finally the anonymous reviewers for their positive or negative comments on my various submissions.

I thank all the colleagues and friends for providing me general guidance and support on professional or non-professional activities, especially Prof. Wiebe van der Hoek, Dr. Katie Atkinson, Dr. Boris Konev, Dr. Bernard Diaz, Dr. Peter McBurney for having me as their teaching assistant, Warden Ian Willis, Iain Young and sub-warden Adam Mannis for having me as an honorary tutor in the Carnatic Hall of Residences, and all my friends in Liverpool for their accompany. I do not have enough space to list all their names here but many of their faces could be found on my Facebook or in my Flickr photos (ID: jiruan).

I thank my parents and my other family members for their selfless support for all the years. My grandmother Yaluo He, who guided me through my childhood, used to ask me when I would complete my PhD finally. But when I could tell her for sure, she had passed away two months earlier. I felt great pain when I learned about her death, and decided to contribute this thesis solely to her.

Last, but not the least, I thank the finical supports from the Department of Computer Science in the University of Liverpool and the University of Liverpool Graduate Association (Hong Kong). It would not be possible for me to study in UK without such supports.

Ji Ruan

Liverpool, December 2008

Contents

Chapter 1

Introduction

This thesis is on reasoning about multi-agent systems. It provides a logic-based account of the specification and verification of multi-agent systems, in terms of time, action and knowledge. This chapter describes the general area of multi-agent systems, the motivation for this research, the methods chosen, and concludes with an overview of the thesis.

Multi-Agent Systems Research What are Multi-Agent Systems? A Multi-Agent System (MAS) is a system that consists of multiple interacting agents. An agent is an entity that could act on its own or on behalf of another entity. We may call them *intelligent* agents, or *rational* agents, if the agents could observe, think and direct their activities to achieve goals. There are generally four characteristics of MASs [100]: firstly, each agent has incomplete information or capabilities for solving the problem; secondly, there is no global system control; thirdly, data is decentralized; and fourthly, computation is asynchronous. For comprehensive surveys of MASs, refer to [98, 72, 100].

One could easily associate agents with human beings, and associate a MAS with a team or community. Human beings are indeed an important source of our inspiration and motivation. For instance, the ultimate goal of RoboCup, a successful initiative for the research in Robotics and Artificial Intelligence, is stated as follows:

> By mid-21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, comply with the official rule of the FIFA, against the winner of the most recent World Cup.
>
> From http://www.robocup.org.

The robot soccer players are agents. In order for a robot team to actually play a soccer game, various technologies must be incorporated including: sensor-fusion, multi-agent communication and collaboration, planning and strategy acquisition, real-time reasoning, robotics, design principles of autonomous agents, etc [99, 60]. In the broader area of multi-agent systems research, the study also involves knowledge representation of beliefs, desires and intentions, cooperation and coordination, organization, communication, negotiation, distributed problem solving, multi-agent learning, and so on.

This thesis focuses on the specification and verification of multi-agent systems using a logic-based approach. So, what is the motivation of this research, and why did we choose a logic-based approach? The first question is addressed in Section 1.1, and the second question in Section 1.2.

1.1 Time, Action and Knowledge in MAS

In this section, we show that 'Time', 'Action', and 'Knowledge' are three important aspects for reasoning about multi-agent systems, with several relevant examples.

Systems Changing Over Time What is time? There are a lot of philosophical debates on the nature of time. But these debates are not the subjects of this thesis. We simply take the following view: *time is part of a fundamental intellectual structure (together with space) within which humans sequence and compare events.*

In our daily life, we use time to record past activities or events, and to plan or organize future ones. Similarly, when we study multi-agent systems, we also want to use time to sequence and compare changes of the systems. Conceptually, time, like space, may be continuous, and this may even lead to some controversial debates, such as in Zeno's Arrow Paradox:

> "If everything when it occupies an equal space is at rest, and if that which is in locomotion is always occupying such a space at any moment, the flying arrow is therefore motionless. "
>
> Aristotle, Physics VI:9, 239b5

But here, we adopt the following view: *time is discrete and systems change one step at a time.* This view is a pragmatic one, because in practice we usually refer

to discrete time points when we talk about a system. For example, we say "The train leaves platform 1 at 5pm" or "The dinner starts at 7pm". Moreover, in multi-agent systems research, the agents are usually computational systems, in which instructions are executed step by step. Therefore this view naturally fits the modelling of changes in such systems.

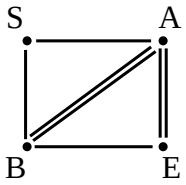
We assume that a multi-agent system is in certain state at each discrete time point. The granularity of the time points may depend on the subjects that we model. For example, in the University of Liverpool, PhD students and their supervisors have to complete a progress review on an annual basis, and students are allowed to be transferred to the next year only after positive feedback, while the research meetings between students and supervisors are on a monthly or a weekly basis. Once we fix the granularity of time, the changes of the system then are modeled as transitions from one time point to another.

Now the questions are how could we represent states of a multi-agent system and the transitions between them, and how to express the desired properties that we want such systems to have.

Agents Having Power To Act A multi-agent system evolves over time and the changes come from the actions made by the agents. So the agents must have the power to act, and they will typically be interested in how to act effectively, i.e., in such a way as to achieve their goals. The agents may have conflicting goals, so their power to act may also depend on how other agents act.

To illustrate these aspects, we introduce the game of *Student vs. Teacher* [77] as follows.

Example 1.1 (Student vs. Teacher)**.** *There is a student and a teacher. They are playing a game as shown in the following diagram.*



The student is located at position S and wants to reach the position of escape, E, but the teacher wants to prevent him from escaping. Each line segment is a link that can be traveled. The teacher starts first and they play in turn. At each round of the game, the teacher can cut one connection anywhere in the diagram, while the student can and must travel one link still open to him at his current

position. The game stops if the student escapes in E (in which case he wins) or he cannot move anymore (in which case he loses). Question: does the student have a way to win this game?

The answer is *no* if the teacher plays smartly. If the teacher first cuts a link between A and E, then no matter whether the student goes to A or B, the teacher can then cut the link from E to that position, and in the next round cuts the last link from E. So there is no link from any other position to E, therefore the student cannot escape. We can say that the teacher has a *strategy* to win this game.

But if the teacher does not play smartly in the first round, then he might lose. For example, if the teacher first cuts the link between S and A, then the student has no choice but go to B and subsequently gets a winning strategy. The strategy can be described as follows: if the teacher cuts any link but the B-E link, then the student wins through this B-E link; if the teacher cuts the B-E link, then the student goes to A, and because there are two links between A and E, no matter which link the teacher cuts in the next round, the student can always reach E.

There are situations in which agents not only compete but also collaborate with each other. For example, in football, the members of one team compete with those of the other team; within the same team, members work together to send the ball into the other team's goal. In the current global economy, countries are competing with each other; but to slow down global warming, countries have to collaborate as well, e.g. cutting greenhouse gas emissions together.

The questions are how can we model the single agent's strategy as well as the strategy of a group of agents, and how can agents reason about their coalition powers.

Agents' Knowledge Evolving Over Time In many scenarios, agents do not necessarily have complete information of the whole system. Therefore to represent and reason about incomplete information is essential too. The term *information* is generally understood as "facts provided or learned about something or someone". In this thesis, we assume that *the information that agents acquire through observation and learning is always accurate.* For instance, in our earlier example, if the teacher makes an action '*cut A-B link*' then the student will observe the same action. Of course, in general, this assumption does not necessarily hold. For example, the speedometer on your bicycle may tell you that the current speed is

25 km per hour, but due to measurement errors, the *real* speed is probably not exactly the same number. This relates to a choice of modelling. If the measurement error is negligible, then we could simply treat the measurement reading as an accurate one; but if it may have a non-trival impact on the system, then we could add more parameters to describe it.

We refer to accurate or truthful information as *knowledge*. In particular, we distinguish two kinds of knowledge: one is about the facts of the system environment or agents, and the other is about the knowledge of other agents' knowledge, which is also called higher-order knowledge. For instance, suppose there are two agents A and B and a lamp in the waiting room; the agent B just comes out from that room, hence he might know the status of the light. But agent A only observes that B is coming out and A is not able to peek into the room. Now we say that agent B knowing whether the lamp is on or off, refers to the first type of knowledge. Agent A knowing whether agent B knows that fact, is higher-order knowledge. Sometimes higher-order knowledge can be very subtle, and plays an important role in agents' decision making. We illustrate this by the *coordinated attack* problem, a well-known problem in the distributed systems folklore. The problem is described as follows (Page 176, [21]).

Example 1.2 (Coordinated Attack)**.** *Two divisions of an army, each commanded by a general, are camped on two hilltops overlooking a valley. In the valley awaits the enemy. It is clear that if both divisions attack the enemy simultaneously they will win the battle, while if only one division attacks it will be defeated. As a result, neither general will attack unless he is absolutely sure that the other will attack with him. In particular, a general will not attack if he receives no messages. The commanding general of the first division wishes to coordinate a simultaneous attack (at some time the next day). The general can communicate only by means of messengers. Normally, it takes a messenger one hour to get from one encampment to the other. However, it is possible that he will get lost in the dark or, worse yet, be captured by the enemy. Fortunately, on this particular night, everything goes smoothly. How long will it take them to coordinate an attack?*

So General A must send a messenger to General B with a message like "*let us attack at 6 am*". Suppose General A did so, and the message was indeed delivered, what should General B do? B did *know* A's proposal, but he could imagine that A had no information so far on whether the message had been

delivered, therefore; A *did not know* that B knew this proposal already. Hence, at that time point, B could not make a decision to attack the enemy, as A would definitely not attack the enemy. Then B had to send a messenger back to inform A that the earlier message had been delivered. Suppose B did so and the message was delivered successfully, should A decide to make an attack? Not really, because A could imagine that B did not know whether the second message was successfully delivered, although A knew that B knew the first message already. Therefore, A must send another messenger to inform B that the second message was received by A. So it is easy to conclude that no matter how many messages are passed between A and B, they could not attain enough high-order knowledge to ensure a simultaneous attack; therefore no successful coordination was possible. In other words, they need infinitely many successful deliveries of messages to attain the knowledge for this attack, which is of course not possible in practice.

One may ask why the two generals do not simply make a telephone call? It is not possible in this example, as they can only rely on the messengers. But if a telephone call is indeed allowed and the generals could do it without being noticed by the enemies, then they will be able to coordinate an attack. The higher-order knowledge they will obtain from the telephone call is called *common knowledge*. In this example, common knowledge is essential for two generals to make life-or-death decisions. Also, the example shows that knowledge can evolve over time, e.g. via communication among agents.

Now, the questions are how we can represent such different knowledge modalities and actions of communication, and how agents can reason about these knowledge and actions.

So far we have looked at three important aspects of multi-agent systems, and asked relevant questions. In the next section, we are going to introduce the methods to address these questions.

1.2 Logic-based Methods and Formal Verification

Logic-based Methods We use logic-based methods for reasoning about multi-agent systems. A logic usually consists of three parts:

- Language (Syntax): well-defined strings of symbols to serve as a language to express the properties of the systems;

- Semantics (Model theory): mathematical models which characterise the system and give meaning to the language.
- Deduction system (Proof theory): a set of formulas and rules to deduce a specific set of formulas which are called theorems.

There are several advantages of using logic-based methods in multi-agent systems modelling.

- *First*, the well-defined logical language is more precise and unambiguous compared to unstructured natural language.
- *Second*, by having rigorous mathematical models of multi-agent systems, the underlying assumptions can be nailed down to precise terms so that there will be no confusion in studying such systems.
- *Third*, the well-defined semantics enables us to specify precisely what properties hold for the models, and may help us understand why some properties fail in such models.
- *Last*, by using axiomatic deduction systems, we could single out what are the principles underlying a particular system, and what are the consequences of such principles.

In particular, we will study multi-agent systems with two classes of logical frameworks. The first class is that of temporal logic, and the second class is that of epistemic logic. A detailed review on these logics is given in Chapter 2.

Formal Verification As with every study or research, we do this research with some purpose. One important purpose is of course, the theoretical understanding of multi-agent systems, but ultimately, we would like our study to be practically useful as well. For that reason, we want to verify multi-agent systems based on our logical approaches. This leads to the formal verification of multi-agent systems.

As the designer of a system, one wants to know whether the designed system behaves as desired. A popular way of doing that is through *simulation and testing*, which are standard methods in software engineering. But there are two main limits for such methods. First, simulation and testing only explore some of the possible behaviors and scenarios of the system, leaving some bugs unexplored;

this is not acceptable for mission critical systems, as witnessed in the failure of the Rocket Ariane 5 Flight 501 in 1996, which was caused by computer software bugs[1]. Second, simulation and testing use natural language to describe the specifications of the system, leaving potential misunderstandings to the test conductors.

To overcome such limits, several approaches to formal verification have been proposed over the years. There are roughly two classes of approaches. One is *theorem proving*, which is related to proof theory mentioned earlier; and the other is *model checking*, which is related to the model theory. In this thesis, we focus solely on *model checking*.

Pioneering work in model checking of temporal logics was done by Clarke, Emerson and Sifakis et al. [15, 64] in the 1980s. For that reason, Clarke, Emerson, and Sifakis shared the 2007 *Turing Award*.

Model checking is the process of checking whether a given model satisfies a logical specification, represented as a logic formula, through *exhaustive* enumeration (explicit or implicit) of all the states reachable from the initial states of the system and the transitions that connect them. Formally, a model checking problem can be stated as follows: given a desired property which is expressed as a logic formula φ, and a model M with a state s in M, decide whether $M, s \models \varphi$.

The model checking process typically involves three main steps:

- **Modelling** The first step is to translate a specification of a system described in natural language into a formalism that is accepted by a model checking tool. The usual tasks involved are abstracting key components and eliminating irrelevant details. The judgment of whether some details are irrelevant or not depends on the design objectives.

- **Specification** The aim of this step is to produce formal specifications, normally in the form of logic formulas, to represent the desired properties that the systems should hold. This also involves logical abstractions, as the logical language is more rigid than the natural language.

- **Verification** This step usually does not need human involvement except in the situation that the model is too large to be handled by the computer. In that case, one may need rework in the Modelling and Specification steps.

This complete procedure will be shown in the case studies in Chapters 4 and 6 of this thesis.

[1]URL: http://en.wikipedia.org/wiki/Ariane_5_Flight_501

Compared with other approaches, there are several clear advantages of the model checking approach. First, model checking is fully automatic and exhaustive, so the full process will need no human intervention, and no states will be left out. Second, model checking tools may produce a counterexample when the design does not satisfy a property, which may lead to great insights on why the property fails. Third, the model checking process can handle very large state spaces that cannot be possibly handled by human beings.

However, there is also a major disadvantage in the model checking approach. As the number of components or agents in a system increases, the number of states of the system may increase exponentially. This is usually refereed to as the *state explosion problem*. There was a breakthrough, when McMillan et al. [12, 52] first used a symbolic representation for state transition graphs, so that much larger systems could be handled. The main insight was to use a symbolic representation based on Bryant's ordered binary decision diagrams (OBDDs) [11]. OBDDs provide a canonical form for Boolean formulas that is often substantially more compact than conjunctive or disjunctive normal form; very efficient algorithms have been developed for manipulating them. Such techniques are also used in the model checking tools for multi-agent systems [66, 25].

1.3 Overview

Chapter 2 of this thesis provides a logical background for the work that is going to be presented. It first introduces temporal logics from a single-agent perspective, and then from a multi-agent perspective. In the single-agent perspective, we have Linear-time Temporal Logic (LTL) and Computational Tree Logic (CTL). The former assumes that a system changes over time *deterministically*, in the sense that there is only one possible outcome for a system to transit from one state to another state; and the latter assumes that a system changes over time *non-deterministically*. In the multi-agent perspective, we have Alternating-time Temporal Logic (ATL), which is a generalization of CTL. In ATL, the changes of a system are determined by the agents' actions. The thesis then goes on to introduce epistemic logics, which are the logical frameworks of knowledge. Epistemic Logic (EL) deals with the agents' knowledge and higher-order knowledge, including common knowledge. We also present two extensions of EL. One is Temporal Epistemic Logic (TEL), which models the temporal changes of a system in addition to the modelling of the agents' knowledge. The other is Dynamic

Epistemic Logic (DEL), which also models the changes of a system but more concretely through actions, compared to the modelling of the changes caused by the flow of time in TEL. A literature review is provided, along with the definitions of languages and semantics of these logics, and the computational complexities of these logics.

The main contributions of this thesis are presented in the next four chapters, from Chapter 3 to Chapter 6.

Chapter 3 and Chapter 4 are focused on the verification of games in the Game Description Language (GDL), a declarative language used in General Game Playing Competition. The aim of the competition is to provide a platform for researchers to develop general purpose game playing systems, which shall have more flexibility compared to dedicated game playing systems such as the famous Chess-playing system Deep Blue. While GDL is specifically designed for describing games, it can also be seen as a language to describe a class of multi-agent systems. A practical problem for a designer of games or multi-agent systems using GDL is to check whether they meets desired specifications. One formalism for reasoning about games that has attracted much interest is Alternating-time Temporal Logic (ATL). Chapter 3 investigates the connection between GDL and ATL. It first demonstrates that GDL can be understood as a specification language for ATL models. Subsequently it shows that it is possible to succinctly characterise GDL game descriptions directly as ATL formulas, and that, as a corollary, the problem of interpreting ATL formulas over GDL descriptions is EXPTIME-Complete. Then in Chapter 4, this connection is explored more practically. In particular, two main contributions are made: firstly, a characterization of the playability conditions which can be used to express the correctness of the games specified in GDL, and secondly an automated tool that uses an ATL model checker to verify the playability conditions for the games described in GDL. The feasibility of our approach is demonstrated by a case study on the game called Tic-Tac-Toe.

Chapters 5 and 6 are focused on the modelling of multi-agent systems with incomplete information. In particular, a study of correspondence between Dynamic Epistemic Logic (DEL) and Temporal Epistemic Logic (TEL) is made. These two logical frameworks are capable of modelling multi-agent systems with agents' knowledge and change. However, there is an apparent difference in terms of model checking: in DEL the interpretation of a dynamic epistemic formula is over a state model, which represents a static view of a multi-agent system; while

in TEL, the interpretation of a temporal epistemic formula is over an interpreted system, in which the full history of a system is unfolded. Chapter 5 tries to link DEL and TEL, showing that DEL can be embedded into a variation of TEL. We then proceed in Chapter 6 to give a study of model checking knowledge dynamics with three state-of-the-art model checkers for multi-agent systems. We first discuss the role of protocols in model checking processes, and proceed with case studies of two problems. With the *Russian Cards Problem*, we show how to use model checking tools to specify and verify communication protocols, and how dynamic epistemic modelling and temporal epistemic modelling compare in practice. With the *Sum And Product Problem*, we show an important feature supported by the dynamic epistemic model checker DEMO, but not by the temporal epistemic model checkers MCK and MCMAS. We also compare the model checking results of different variants of this problem, and discuss the influence of model checking time by different representations of a same specification.

Finally Chapter 7 gives a conclusion of this research and provides some ideas for further research.

Sources of Materials

Chapter 3 and 4 are based on collaborations with Prof. Wiebe van der Hoek and Prof. Michael Wooldridge. The main work of Chapter 3 was presented in [82] at the Workshop on Logic, Rationality and Interaction (LORI'07) which took place at Beijing, in August 2007. Chapter 4 is yet to be published. Chapter 5 is joint work with Dr. Hans van Ditmarsch and Prof. Wiebe van der Hoek; a preliminary version [90] was presented in the Workshop of Formal Approaches to Multi-Agent Systems (FAMAS'007), which took place at Durham, in September 2007. Chapter 6 has two main sources: one is a paper [91] published in the Journal of Logic and Computation jointly with Dr. Hans van Ditmarsch and Prof. Rineke Verbrugge; the other is a paper [93] published in the proceedings of MoChArt 05 (Model Checking in Artificial Intelligence) jointly with Dr. Hans van Ditmarsch, Prof. Wiebe van der Hoek and Prof. Ron van der Meyden. The contributions and collaborations of my co-authors are highly appreciated.

Chapter 2

Background

This chapter presents the temporal and epistemic frameworks that form the basis of this thesis. We assume that the readers have a basic knowledge of propositional logic.

2.1 Temporal Logics

2.1.1 Introduction

Temporal logic is a formalisation of time. It is developed mainly from two traditions. The first is a *philosophical* tradition, which is rooted in the analysis of temporal aspects of natural language; therefore it is sometimes referred to as *tense logic*. An important milestone in contemporary temporal logic was first laid by *A. Prior* in his book *Time and Modality* [63] in 1957. Prior's approach treated time with modalities, which are expressions broadly associated with notions of possibility and necessity. Suppose we have a proposition "It is sunny" (call it proposition p); although the meaning of this proposition is constant in time, the truth value of this proposition may change over time, especially in a place like Liverpool. With the modalities like 'always', 'eventually', we can have such statements: "it is *always* sunny", "*eventually* it is sunny". They can be expressed in our logical language as $\Box p$ and $\Diamond p$ respectively. The meaning of these statements depends on their interpretation on the structures that represent time. More details will be provided in the next section.

The second is a *computational* tradition. The motivation is to apply formal tools to the verification of computer programs, or more broadly computer systems. This area was created by computer scientists, A. Pnueli and Z. Manna et al. [62,

49, 50], and further developed by E. Clarke and E. A. Emerson et al. [15, 13]. In [62], Pnueli proposed a unified approach to program verification, which applies to both sequential and parallel programs. The main proof method suggested was that of temporal reasoning, in which the time dependence of events is the basic concept. The programs are in a certain state in each time instance, and the correctness of the programs can be expressed as temporal specifications, such as "$\Box\neg$*deadlock*" meaning *the program can never enter a deadlock state*. In [15], E. Clarke and E. A. Emerson invented the method of model checking.

There are many temporal logics proposed since then. We list three categories that are concerned with this thesis:

- Linear-time Temporal Logics (LTL)
- Computational Tree Logics (CTL) - an example of Branching-time Temporal Logics
- Alternating-Time Temporal Logics (ATL)

The main difference between LTL and CTL is their view of *time flows*. LTL considers a time flow as a chain of time instances, while CTL views a time flow as a tree branching to the future, i.e., in each time instance, there may be several future instances. ATL, with multi-agent perspective, is a generalisation over CTL. We proceed with an introduction of structures of time and then present the languages and semantics of LTL, CTL and ATL in subsequent sections.

2.1.2 Structures of Time

Basic Temporal Frames and Extra Properties

Time is a basic component of the measuring system. We use it to sequence events, to compare the duration of events, and more importantly to organise our activities. One can view *a flow of time* as an object consisting of separate time instances and certain structures.

Definition 2.1. *A flow of Time is a frame:* $\langle \mathsf{T}, \mathsf{R} \rangle$ *such that* R *is binary relation over* T*. Here elements in* T *are called* time instances*, and* R *is called* precedence relation*; if a pair* (s, t) *belongs to* R*, we say that* s *is earlier than* t*, written as* $s\mathsf{R}t$*.*

This is just a basic frame, which may not meet our intuition about time. For example, $t\mathsf{R}t$ is allowed in the basic temporal frame, but our intuition tells us that time t should not be earlier than itself. To avoid such problems, we could add further restrictions on the precedence relation. Here are some first-order properties[1] representing various restrictions:

- Irreflexivity: $\forall x(\neg(x\mathsf{R}x))$;
- Transitivity: $\forall xyz(x\mathsf{R}y \wedge y\mathsf{R}z \rightarrow x\mathsf{R}z)$;
- Linearity: $\forall xy(x\mathsf{R}y \vee y\mathsf{R}x \vee x = y)$;
- Seriality: $\forall x\exists y(x\mathsf{R}y)$;
- Density: $\forall xy(x\mathsf{R}y \rightarrow \exists z(x\mathsf{R}z \wedge z\mathsf{R}y))$.

Irreflexivity says that x cannot precede itself. *Transitivity* says that if a time instance x precedes y and y precedes z, then x precedes z. *Linearity* says that for two time instances, either one precedes the other, or they are equal. *Seriality* says for any time instance, there is always a future time instance. *Density* says that for two time instances, there is always a time instance in between.

There are also some properties that can only be defined in *second-order*, involving quantification over sets of instances in time, such as:

- Continuity: "Every subset with an upper bound has a supremum(i.e. a lowest upper bound)".

The combinations of these properties define various temporal frames. For example, with *transitivity* and *irreflexivity*, we can define a temporal frame that is not circular; with linearity and transitivity, we have a linear temporal frame. Notice that the natural number frame $\langle \mathbb{N}, < \rangle$ and real number frame $\langle \mathbb{R}, < \rangle$ are linear temporal frames; while the former is discrete, and the latter is continuous.

Temporal Frames from State Machines

Apart from our intuition of what 'real time' should be, there is another perspective from computer science. In a computational perspective, time is not continuous, but discrete, as computation is made step by step. It is reasonable to assume that a computer or a machine is in a particular state in each time instance. A

[1] $\forall x$ means "For all x in the domain", and $\exists x$ means "There exists an x in the domain".

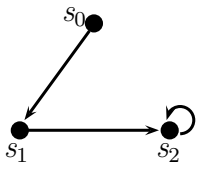


Figure 2.1: A deterministic state machine

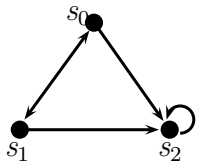


Figure 2.2: A non-deterministic state machine

binary relation among these states indicates possible transitions from one state to another. We require that for every state, there is at least one possible transition. The reason to ensure this is because we assume that machine time is infinite, i.e. computations do not stop. This is not to say that we cannot have a stopping state, in which a machine stops to operate. We can just have such a state that it only transits to itself.

Definition 2.2 (Finite State Machine)**.** *A finite state machine is a frame:* $\langle \mathsf{S}, \mathsf{R} \rangle$*, where* S *is a finite set of states of a machine, and* R *is a binary relation over* S *such that for every* $s \in \mathsf{S}$*, there exists* $s' \in \mathsf{S}$ *with* $s\mathsf{R}s'$*, i.e.* R *is serial.*

There are two types of finite state machines: *deterministic* and *non-deterministic*. If for each state, there is exactly one transition either to another state or to itself, then it is a deterministic state machine; otherwise, it is a non-deterministic state machine.

Example 2.1. *Figure 2.1 represents a deterministic state machine, and Figure 2.2 represents a non-deterministic state machine.*

It was mentioned that a main difference between LTL and CTL lies on their different view on time flow. This difference can be shown by unwinding the

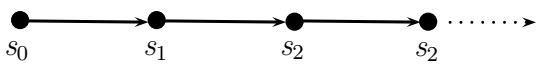


Figure 2.3: The result of unwinding the state machine in Figure 2.1

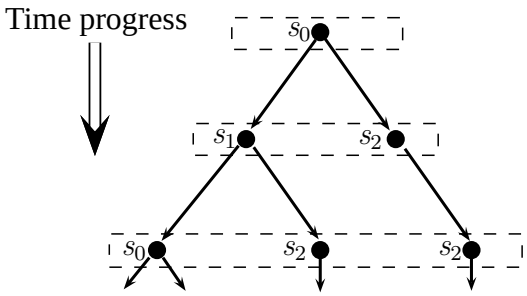


Figure 2.4: The result of unwinding the state machine in Figure 2.2.

state machines in the above examples. Here 'unwinding' is a process to make a state machine into a linear or branching structure without altering the transition relations between the states. The result of unwinding the state machine in Figure 2.1 from the state s_0 is presented in Figure 2.3, which shows a linear structure. The result of unwinding the state machine in Figure 2.2 from the state s_0 is presented in Figure 2.4, which shows a branching structure.

For these frames, we define a key concept called *computation*, which is intuitively a linear-time line or a branch in the branching-time tree.

Definition 2.3. *Given a temporal frame $\langle \mathsf{S}, \mathsf{R} \rangle$, an s_0-computation is an infinite sequence of states $\lambda = s_0, s_1, s_2, \cdots$, such that for any $i \in \mathbb{N}$, $s_i \mathsf{R} s_{i+1}$. We denote s_i as $\lambda[i]$.*

It is easy to see that in Figure 2.3, $s_0, s_1, s_2, s_2, \cdots$ is a computation starting with s_0, and in Figure 2.4, $s_1, s_2, s_2, \cdots$ is a computation starting with s_1.

Now we give a generic definition of a temporal model.

Definition 2.4. *Given a set of atomic propositions P, a temporal model M is a tuple $\langle \mathcal{F}, \mathsf{V} \rangle$ where $\mathcal{F} = \langle \mathsf{S}, \mathsf{R} \rangle$ is a temporal frame, and V is a function from S to $\wp(P)$, the power set of P.*

If $\mathcal{F}$ is a linear-time temporal frame, we call M a linear-time temporal model, and if $\mathcal{F}$ is a branching-time temporal frame, we call M a branching-time temporal model.

Given these models, we need languages to talk about them. In the following sections, we will present three temporal languages and give their interpretations to respective temporal models.

2.1.3 Linear-time Temporal Logic

The language of Linear-time Temporal Logic (LTL) is an extension of propositional logic with temporal modalities.

Definition 2.5 (LTL Language)**.** *The language of Linear-time Temporal Logic in Backus Naur Form (BNF) is as follows:*

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \psi \mid \bigcirc\varphi \mid \varphi \mathcal{U} \psi$$

where, p is a propositional atom.

The abbreviations $\top, \bot, \vee, \rightarrow$ and $\leftrightarrow$ are defined as usual. We can define the following abbreviations for two temporal modalities $\diamondsuit\varphi ::= \top \mathcal{U} \varphi$ and $\Box\varphi ::= \neg\diamondsuit\neg\varphi$.

The formulas with temporal modalities have the following intuitive meanings.

- $\bigcirc\varphi$: in the next time step φ will be true;
- $\varphi \mathcal{U} \psi$: φ will be true from now until ψ;
- $\Box\varphi$: always in the future from now, φ will be true;
- $\diamondsuit\varphi$: at least once in the future, φ will be true.

We now give a formal semantics of this language with respect to linear-time temporal models.

Definition 2.6 (LTL Semantics)**.** *Given a linear-time temporal model M, a state s, and an s-computation λ, we have:*

$$\begin{aligned}
M, s \models p &::= p \in \mathsf{V}(s) \\
M, s \models \neg\varphi &::= \textit{not } M, s \models \varphi \\
M, s \models \varphi \wedge \psi &::= M, s \models \varphi \textit{ and } M, s \models \psi \\
M, s \models \bigcirc\varphi &::= M, \lambda[1] \models \varphi \\
M, s \models \varphi \mathcal{U} \psi &::= \exists j \in \mathbb{N}(M, \lambda[j] \models \psi \textit{ and } \forall 0 \leq i < j, \lambda[i] \models \varphi)
\end{aligned}$$

Complexity

Given the language and semantic of a logic, there are two types of problems that we are interested in: the model checking problems and the satisfiability problems. A *model checking problem* is: given a model M, a state s, and a formula φ, determine whether $M, w \models \varphi$. A *satisfiability problem* is: given a formula φ, determine whether there is a model M, a state s, such that $M, w \models \varphi$. These definitions are general, and they can be tailored to LTL case just by restricting M to be a linear-time model and φ to be an LTL formula.

Complexity theory measures the difficulty of problems in terms of the resources required to solve them. Such measure is a function of the size of the input. The complexity of both the model checking problem and the satisfiability problem in LTL is PSPACE-Complete [70].

2.1.4 Computational Tree Logic

Computational Tree Logic (CTL) was first introduced by Clarke and Emerson [15] in 1981. It considers branching-time temporal models, so that there can be multiple paths starting from one state. The language introduces symbols A and E with meaning "for all paths", "for some path" respectively.

Definition 2.7 (CTL Language)**.** *The language of* CTL *in BNF is as follows:*

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \psi \mid \mathsf{A}\bigcirc\varphi \mid \mathsf{A}\varphi\,\mathcal{U}\,\psi \mid \mathsf{E}\varphi\,\mathcal{U}\,\psi$$

where p is a propositional atom.

The abbreviations $\top, \bot, \vee, \rightarrow$ and $\leftrightarrow$ are defined as usual. We can define the following abbreviations for more temporal modalities:

- $\mathsf{E}\bigcirc\varphi ::= \neg\mathsf{A}\bigcirc\neg\varphi$;
- $\mathsf{E}\diamondsuit\varphi ::= \mathsf{E}\top\,\mathcal{U}\,\varphi$;
- $\mathsf{A}\square\varphi ::= \neg\mathsf{E}\diamondsuit\neg\varphi$;
- $\mathsf{A}\diamondsuit\varphi ::= \mathsf{A}\top\,\mathcal{U}\,\varphi$;
- $\mathsf{E}\square\varphi ::= \neg\mathsf{A}\diamondsuit\neg\varphi$.

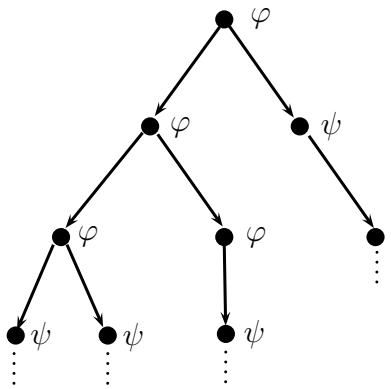


Figure 2.5: A system of which the starting state satisfies $\mathsf{A}\varphi\,\mathcal{U}\,\psi$

Definition 2.8 (CTL Semantics)**.** *Given a branching-time temporal model M, and a state s, we have:*

$$
\begin{aligned}
M, s \models p &::= p \in \mathsf{V}(s) \\
M, s \models \neg\varphi &::= \textit{not } M, s \models \varphi \\
M, s \models \varphi \wedge \psi &::= M, s \models \varphi \textit{ and } M, s \models \psi \\
M, s \models \mathsf{A}\bigcirc\varphi &::= \textit{for all } s\textit{-computation } \lambda, M, \lambda[1] \models \varphi \\
M, s \models \mathsf{A}\varphi\,\mathcal{U}\,\psi &::= \textit{for all } s\textit{-computation } \lambda, \exists j \in \mathbb{N}, M, \lambda[j] \models \psi \textit{ and} \\
&\quad\ \forall 0 \leq i < j, \lambda[i] \models \varphi \\
M, s \models \mathsf{E}\varphi\,\mathcal{U}\,\psi &::= \textit{for some } s\textit{-computation } \lambda, \exists j \in \mathbb{N}, M, \lambda[j] \models \psi \textit{ and} \\
&\quad\ \forall 0 \leq i < j, \lambda[i] \models \varphi
\end{aligned}
$$

Intuitively, $\mathsf{A}\bigcirc\varphi$ is true in s if φ is true in the *next state* of all the paths starting from s; $\mathsf{A}\varphi\,\mathcal{U}\,\psi$ is true in s if for all the paths starting from s, the formula $\varphi\,\mathcal{U}\,\psi$ is true in s; similarly, E is for the existence of one path. We illustrate the idea for CTL formula $\mathsf{A}\varphi\,\mathcal{U}\,\psi$ with Figure 2.5.

Complexity

The first algorithm for CTL model checking was presented by Clarke and Emerson [15] in 1986. Their algorithm was polynomial in both the size of the transition system and the length of the formula. A survey on model checking temporal logics including CTL can be found in [16, 14]. The complexity of the model checking

problem in CTL is PTIME-Complete, and the complexity of the staisfibility problem in CTL is EXPTIME-Complete [16, 14].

2.1.5 Alternating-time Temporal Logic

Alternating-time Temporal Logic (ATL) was first proposed by R. Alur, T. Henzinger, and O. Kupferman [5, 6]. It generalises two varieties of temporal logics: linear-time temporal logic, which assumes implicit universal quantification over all paths that are generated by system moves; and branching-time temporal logic, which allows explicit existential and universal quantification over all paths. ATL offers selective quantification over those paths that are possible outcomes of games, such as the game in which the system and the environment alternate moves.

In [31], V. Goranko and G. van Drimmelen gave a sound and complete axiomatisation of ATL, and shown that when considering formulas over a fixed finite set of players, the decidability problem is EXPTIME-Complete. ATL in [6] is only for reasoning about complete information games, while in [85], W. van der Hoek and M. Wooldridge extended ATL with an epistemic component, which can deal with incomplete information games.

Language and Semantics

We introduce the language and semantics of the ATL in [6]. The key construct in ATL is $\langle\langle C\rangle\rangle T\varphi$, where C is a coalition, (a set of agents), and $T\varphi$ a temporal formula, meaning "coalition C can act in such a way that $T\varphi$ is guaranteed to be true". Temporal formulas are built using the modalities $\bigcirc$, $\Box$, and $\mathcal{U}$, where $\bigcirc$ means "in the next state", $\Box$ means "always", and $\mathcal{U}$ means "until".

Definition 2.9 (ATL Language)**.** *Given a set of agents Ag, and a set of atomic propositions Φ, the language of* ATL *is given in BNF as follows:*

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \psi \mid \langle\langle C\rangle\rangle \bigcirc \varphi \mid \langle\langle C\rangle\rangle \Box\varphi \mid \langle\langle C\rangle\rangle \varphi \,\mathcal{U}\, \psi$$

where $p \in \Phi$ is a propositional variable and $C \subseteq Ag$ is a set of agents.

ATL has a number of equivalent semantics; since *moves*, or *actions*, play such a prominent role in game playing, we use *Action-based Alternating Transition Systems* following [6].

Definition 2.10 (ATL Model)**.** *An* Action-based Alternating Transition System *(*AATS*) is a tuple*

$$\mathcal{A} = \langle Q, q_0, Ag, Ac_1, \ldots, Ac_n, \rho, \tau, \Phi, \pi \rangle$$

where: Q is a finite, non-empty set of states*; $q_0 \in Q$ is the* initial state*; $Ag = \{1, \ldots, n\}$ is a finite, non-empty set of* agents*; Ac_i is a finite, non-empty set of* actions*, for each $i \in Ag$, where $Ac_i \cap Ac_j = \emptyset$ for all $i \neq j \in Ag$; $\rho : Ac_{Ag} \to 2^Q$ is an* action precondition function*, which for each action $a \in Ac_{Ag}$ defines the set of states $\rho(a)$ from which a may be executed; $\tau : Ac_1 \times \cdots \times Ac_n \times Q \to Q$ is a partial* system transition function*, which defines the state $\tau(\vec{a}, q)$ that would result by the performance of $\vec{a}$ from state q – note that, as this function is partial, not all joint actions are possible in all states (see the precondition function above); Φ is a finite, non-empty set of* atomic propositions*; and $\pi : Q \to 2^\Phi$ is a valuation function, which gives the set of atomic propositions satisfied in each state: if $p \in \pi(q)$, then this means that the propositional variable p is satisfied in state q.*

It is required that AATSs satisfy the following coherence constraints: (*Non-triviality*) agents always have at least one legal action – $\forall q \in Q, \forall i \in Ag, \exists a \in Ac_i$ s.t. $q \in \rho(a)$; and (*Consistency*) the ρ and τ functions agree on actions that may be performed: $\forall q, \forall \vec{a} = \langle a_1, \cdots, a_n \rangle, (\vec{a}, q) \in \text{dom } \tau$ iff $\forall i \in Ag, q \in \rho(a_i)$.

Given an agent $i \in Ag$ and a state $q \in Q$, we denote the *options* available to i in q – the actions that i may perform in q – by $options(i, q) = \{a \mid a \in Ac_i \text{ and } q \in \rho(a)\}$. For a coalition C, we define $options(C, q) = \bigcup \{options(i, q) \mid i \in C\}$. An *action profile* for a coalition $C = \{i_1, \ldots, i_k\} \subseteq Ag$ in state q is a tuple of actions $\langle ac_1, \ldots, ac_k \rangle$, where $ac_j \in options(i_j, q)$ for each $j \in [1..k]$. We then say that a *strategy* for an agent $i \in Ag$ is a function $\sigma_i : Q \to Ac_i$ which must satisfy the *legality* constraint that $\sigma_i(q) \in options(i, q)$ for all $q \in Q$. A *strategy profile* for a coalition $C = \{i_1, \ldots, i_k\} \subseteq Ag$ is a tuple of strategies $\langle \sigma_1, \ldots, \sigma_k \rangle$, one for each agent $i \in C$. We denote by Σ_C the set of all strategy profiles for coalition $C \subseteq Ag$; if $\sigma_C \in \Sigma_C$ and $i \in C$, then we denote i's component of σ_C by σ_C^i. Given a strategy profile $\sigma_C \in \Sigma_C$ and state $q \in Q$, let $out(\sigma_C, q)$ denote the set of possible states that may result by the members of the coalition C acting as defined by their components of σ_C for one step from q:

$$out(\sigma_C, q) = \{q' \mid \tau(\vec{a}, q) = q' \text{ where } (\vec{a}, q) \in \text{dom } \tau \text{ and } \sigma_C^i(q) = a_i \text{ for } i \in C\}$$

Notice that the set $out(\sigma_{Ag}, q)$ is a singleton because we assume that when all agents choses their actions, a unique outcome is determined. Also, $out(\cdot,\cdot)$ only deals with one-step successors, and we interchangeably write $out(\sigma_C, q)$ and $out(Ac_C, q)$, where Ac_C is an action profile of coalition C in state q. The idea is that for the one-step future, a strategy carries the same information as an action. A q_0-*computation* is an infinite sequence of states $\lambda = q_0, q_1, \ldots$. Given $u \in \mathbb{N}$, we use $\lambda[u]$ as the state indexed by u in the computation λ.

Given a strategy profile σ_C for some coalition C, and a state $q \in Q$, we define $comp(\sigma_C, q)$ to be the set of possible runs that may occur if every agent $i \in C$ follows the corresponding strategy σ_i, starting when the system is in state $q \in Q$. That is, the set $comp(\sigma_C, q)$ will contain all possible q-computations that the coalition C can "enforce" by cooperating and following the strategies in σ_C.

$$comp(\sigma_C, q) = \{\lambda \mid \lambda[0] = q \text{ and } \forall u \in \mathbb{N} : \lambda[u+1] \in out(\sigma_C, \lambda[u])\}.$$

Again, note that for any state $q \in Q$ and any grand coalition strategy σ_{Ag}, the set $comp(\sigma_{Ag}, q)$ will be a singleton, consisting of exactly one infinite computation.

Definition 2.11 (ATL Semantics)**.** *Given an* AATS $\mathcal{A}$ *and a state* q*, we have,*

$\mathcal{A}, q \models p$ *iff* $p \in \pi(q)$ *(where* $p \in \Phi$*);*

$\mathcal{A}, q \models \neg\varphi$ *iff* $\mathcal{A}, q \not\models \varphi$*;*

$\mathcal{A}, q \models \varphi \wedge \psi$ *iff* $\mathcal{A}, q \models \varphi$ *and* $\mathcal{A}, q \models \psi$*;*

$\mathcal{A}, q \models \langle\langle C\rangle\rangle \bigcirc \varphi$ *iff* $\exists \sigma_C \in \Sigma_C$*, such that* $\forall \lambda \in comp(\sigma_C, q)$*, we have* $\mathcal{A}, \lambda[1] \models \varphi$*;*

$\mathcal{A}, q \models \langle\langle C\rangle\rangle \Box \varphi$ *iff* $\exists \sigma_C \in \Sigma_C$*, such that* $\forall \lambda \in comp(\sigma_C, q)$*, we have* $\mathcal{A}, \lambda[u] \models \varphi$ *for all* $u \in \mathbb{N}$*;*

$\mathcal{A}, q \models \langle\langle C\rangle\rangle \varphi \mathcal{U} \psi$ *iff* $\exists \sigma_C \in \Sigma_C$*, such that* $\forall \lambda \in comp(\sigma_C, q)$*, there exists some* $u \in \mathbb{N}$ *such that* $\mathcal{A}, \lambda[u] \models \psi$*, and for all* $0 \leq v < u$*, we have* $\mathcal{A}, \lambda[v] \models \varphi$*.*

The abbreviations $\top, \bot, \vee, \rightarrow$ and $\leftrightarrow$ are defined as usual. And $\langle\langle C\rangle\rangle \diamondsuit \varphi$ is defined as $\langle\langle C\rangle\rangle \top \mathcal{U} \varphi$. For readability, we omit set brackets in cooperation modalities, for example writing $\langle\langle 1\rangle\rangle$ instead of $\langle\langle \{1\}\rangle\rangle$.

Complexity

The complexity of the model checking problem in ATL is PTIME-Complete [6], and the complexity of the satisfiability problem in ATL is EXPTIME-Complete [19, 97]. These results are based on the assumption that the game models are given explicitly as mathematical structures. The complexity might be different if the game models are represented differently. R. Alur [2] *et al.* developed a model checking tool called MOCHA to specify and verify multi-agent systems with ATL. In MOCHA, the game modes are represented in the form of reactive modules and such representation is much more compact than the explicit representation. In [80] W. van der Hoek et al. showed that the complexity of the model checking ATL with respect to reactive modules representation could be exponential in the size of the reactive modules and the length of the formula.

Alternating Bisimulation

We now give an equivalence relation between two AATSs, called *Alternating Bisimulation*. The purpose is to characterise the AATS structures that can not be distinguished by ATL formulas. Here by 'distinguish', we mean there exists an ATL formula such that it is true in one structure but false in another structure. The following definition is based on [4].

Definition 2.12 (Alternating Bisimulation)**.** *Let* $\mathcal{A}_1 = \langle Q_1, q_1, Ag, Ac_1^1, \ldots, Ac_n^1, \rho_1, \tau_1, \Phi, \pi_1 \rangle$ *and* $\mathcal{A}_2 = \langle Q_2, q_2, Ag, Ac_1^2, \ldots, Ac_n^2, \rho_2, \tau_2, \Phi, \pi_2 \rangle$ *be two* AATS*'s. Then a relation* $\mathcal{R} \subseteq Q_1 \times Q_2$ *is called an* alternating bisimulation *if* $\mathcal{R} q_1 q_2$ *and, for every two states* t_1 *and* t_2 *for which* $\mathcal{R} t_1 t_2$*,we have:*

- **Invariance***: For all* $p \in \Phi$*,* $p \in \pi(t_1)$ *iff* $p \in \pi(t_2)$*.*
- **Zig***: For every coalition* $C \subseteq Ag$*, and every* $ac_C^1 \in options(C, t_1)$*, there exists* $ac_C^2 \in options(C, t_2)$ *such that for every* $t_2' \in out(ac_C^2, t_2)$*, there is a* $t_1' \in out(ac_C^1, t_1)$ *so that* $\mathcal{R} t_1' t_2'$*.*
- **Zag***: For every coalition* $C \subseteq Ag$*, and every* $ac_C^2 \in options(C, t_2)$*, there exists* $ac_C^1 \in options(C, t_1)$ *such that for every* $t_1' \in out(ac_C^1, t_1)$*, there is a* $t_2' \in out(ac_C^2, t_2)$ *so that* $\mathcal{R} t_1' t_2'$*.*

Note that the set of agents in both structures are the same, while the actions in both structures do not have to be the same, since in ATL, one cannot directly refer to actions in the object language. We have:

Theorem 2.1 ([4]). *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be such that there is an alternating bisimulation $\mathcal{R}$ between them, with $\mathcal{R}q_1q_2$. Then, for all* ATL *formulas φ:*

$$\mathcal{A}_1, q_1 \models \varphi \quad \Leftrightarrow \quad \mathcal{A}_2, q_2 \models \varphi$$

2.2 Epistemic Logics

2.2.1 Introduction

Epistemic logic is a formalisation of knowledge. Its philosophical root can be dated back to Ancient Greece with epistemology. While philosophers since Aristotle have discussed Modal Logic, it was C.I. Lewis who introduced the first symbolic and systematic approach to the topic in 1912 (see [45]). Modal logic continued to mature as a field, reaching its modern form in 1963 with the work of S. Kripke [44]. Many papers were written in the 1950s on a logic of knowledge, but it was von Wright's book *An Essay in Modal Logic* (1951) [96] that is seen as a founding document.

Hintikka's Seminal work *Knowledge and Belief* [39] is the first book-length work to suggest using modalities to capture the semantics of knowledge. Since then, many philosophers have been interested in further developing the notions of knowledge and belief using possible world semantics, which is also called Kripke semantics, due to the contribution of S. Kripke [44].

In the late 1980s, there was a merge of temporal frameworks and epistemic frameworks [33, 21]. This development was originally motivated by the need to reason about communication protocols. One is typically interested in what different parties to a protocol know before, during and after a run (an execution sequence) of the protocol. It brought multi-agent systems into perspective, as different parties can be typically modeled as agents. This interest in change of knowledge over time is already eminent in this area for twenty years. Fagin, Halpern, Moses and Vardi's seminal *Reasoning about Knowledge* [21] is a culmination of several earlier papers in this area, and also incorporates Halpern and Vardi's 1986 paper [33] *The Complexity of Reasoning about Knowledge and Time.*

Dynamic Epistemic Logic studies what kinds of events are responsible for change of knowledge in a multi-agent setting. A quizmaster may publicly announce the winning lot, or whisper it in the ear of his assistant. Both result in a change of knowledge for everybody present, although the change is different in

either case. Where belief revision [1] is interested in describing the effect of expansion, contraction and revision of a belief set of one agent, dynamic epistemic logic treats all of knowledge, higher-order knowledge, and its dynamics on the same level, and it gives a fine-tuned analysis of the *way* the revision is brought about, ranging from a private insight by one agent to a public announcement in a group.

Unlike temporal epistemic logics, where the meaning of a temporal shift only appears from the underlying model, in dynamic epistemic logic this change is specified 'directly' in the dynamic operators. Starting with a few somewhat isolated contributions in the late 1980s [61, 74], the area strongly developed from the late 1990s onward [28, 10, 92]. A general theory only now and partially emerges. We will base our treatment of dynamic epistemic logic on [92].

In the following, we will first introduce epistemic logic with its most popular semantics: possible world semantics. Then we present two extensions of epistemic logic, namely temporal epistemic logic, which extends it with time; and dynamic epistemic logic, which extends it with actions, or events.

2.2.2 Basic Epistemic Logic

Possible World Model

The basic idea of possible world semantics is simple: apart from the current world, we could consider different possible worlds; if something is known, then it must be true in *all* the worlds that we consider possible. This can be illustrated by the following example.

Example 2.2. *Suppose that Alice's forehead is painted with either a white or black dot. But she cannot see her forehead so she does not know which color it is; in other words, she considers both cases possible.*

Here we give a formal definition.

Definition 2.13 (Possible World Model)**.** *Given a set of atomic propositions P and a set of n agents, a possible world model is a tuple $\langle W, R_1, \cdots, R_n, \pi \rangle$, where W is a finite non-empty set of worlds(states), R_i is a binary relation, and π is a valuation function from W to $\wp(P)$.*

Possible World Models are also called Kripke Models. Let us go back to Example 2.2. In Figure 2.6 is a graphic representation of the Kripke model

associated with this example. We have two possible worlds: one is indicated by a black dot, the other by a white dot. The white-dot world is underlined, indicating the *current* world. Alice considers that the other world is also possible, indicated by the arrow from right to left. Of course, Alice also considers the current world possible, so there is a reflexive arrow on it. Moreover she could imagine that she could be in the other world, therefore we have a reflexive arrow on the black-dot world, and a left-to-right arrow as well. In other words, Alice could not distinguish these two worlds, which, as we call, are in a same equivalence class. Suppose, we have a proposition p that says "Alice has a white dot on the forehead". It is easy to see that p is true in the white-dot world, and false in the black-dot world. Accordingly in Figure 2.6, the white-dot world is labelled with p indicating p is true in this world.

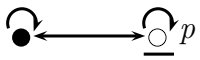


Figure 2.6: A Kripke model for Example 2.2

Language and Semantics

Definition 2.14 (EL Language)**.** *Given a set of agents Ag, the language of Epistemic Logic (EL) is as follows, in BNF,*

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \psi \mid K_i\varphi \mid C_B\varphi$$

where, $i \in Ag$, $B \subseteq Ag$.

Since we consider Multi-Agent Systems, we assume there are always two or more agents in Ag, i.e. $|Ag| \geq 2$.

Definition 2.15 (EL Semantics)**.** *Given a Kripke model $\langle W, R_1, \cdots, R_n, \pi\rangle$, and a possible world w, we have:*

$$
\begin{aligned}
M, w \models p &::= p \in \pi(s) \\
M, w \models \neg\varphi &::= \textit{not } M, w \models \varphi \\
M, w \models \varphi \wedge \psi &::= M, w \models \varphi \textit{ and } M, w \models \psi \\
M, w \models K_i\varphi &::= \textit{for all } v \in W \textit{ such that } wR_iv, M, v \models \varphi \\
M, w \models C_B\varphi &::= \textit{for all } v \in W \textit{ such that } wR_B^*v, M, v \models \varphi
\end{aligned}
$$

where R_B^* *is the transitive and reflexive closure of* $\bigcup_{i\in B} R_i$.

Similar to the relations in temporal models, the relation R in a Kripke model can also have more constraints. Here we mention three:

- Reflexivity: $\forall x(xRx)$;
- Symmetry: $\forall xy(xRy \rightarrow yRx)$;
- Transitivity: $\forall xyz(xRy \wedge yRz \rightarrow xRz)$.

For any relation R, if reflexivity, symmetry and transitivity all hold, then we call R an S5 relation, or an equivalence relation. In the rest of this thesis, we give S5 relations a special symbol '$\sim$'. It will be specifically used in the models of knowledge.

Complexity

The complexity of the model checking problem in EL is PTIME-Complete and the complexity of the satisfiability problem in EL is EXPTIME-Complete [35].

2.2.3 Temporal Epistemic Logic

The central notion in the work of Fagin et al. [21], is that of an *interpreted system*. When compared to Kripke (possible worlds) models, interpreted systems have at least two appealing features: a natural accessibility relation between domain objects, that can be summarised as 'each agent knows its own state', and an equally natural notion of dynamics, modelled by *runs*. The accessibility relation as we know it from the possible worlds model is in this case *grounded*; it has a direct and natural interpretation, as follows. In an interpreted system, the role of

possible worlds is performed by global states, which are constituted by the agents' local states and the state of the environment. Each agent knows exactly its own local state: two global states are indistinguishable for an agent if his local states are the same. Secondly, an interpreted system defines a number of *runs* through such global states (i.e. a sequence of global states). Each run corresponds to a possible computation allowed by a protocol. In an object language with temporal and epistemic operators one can then express temporal properties such as *liveness* (i.e. something will eventually happen), and temporal epistemic properties such as *perfect recall* (i.e. the agents remember what have happened).

The temporal epistemic logic proposed in [21] is based on linear-time structures. Rather than linear-time, one may consider branching time logic, and apart from *synchrony* (i.e. the agents know what the time is) and perfect recall, one may consider properties with or without assuming a unique *initial state*, and with or without the principle of *no learning* (i.e. the agent will not learn anything that will allow him to distinguish two states that he could not distinguish before). Varying only these parameters already yield 96 different logics: for a comprehensive overview of the linear case we refer to [32], and for the branching time case, to [87]. Moreover, apart from the interpreted systems stance there have been several other and related approaches to knowledge and time, like the distributed processes approach of Parikh and Ramanujam [56].

Interpreted System

We formally define an *interpreted system* $\mathcal{I}$ for n agents.

Definition 2.16 (Interpreted System)**.** *A global state s is a tuple $s = (s_e, s_1, \ldots, s_n)$ where s_e is the state of the environment and where for $i = 1 \ldots n$, s_i is the local state of agent i. The set of global states of interest will be denoted $\mathcal{G}$. A* run *over $\mathcal{G}$ is a sequence of states, or, rather, a function r from time $\mathbb{N}$ to global states. The pair (r, m) consisting of a run and a time point is also referred to as a point. Let $r(m) = s$ be the global state at time m in run r, then with $r_i(m)$ we mean local state s_i. An* interpreted system $\mathcal{I} = (\mathcal{R}, \pi)$ *over $\mathcal{G}$ is a system $\mathcal{R}$ of runs over a set $\mathcal{G}$ of global states with a valuation π which gives (r, m) a subset of atoms in Q that are true in (r, m). Two points (r, m) and (r', m') are indistinguishable for i, written $(r, m) \sim_i (r', m')$, if $r_i(m) = r'_i(m')$.*

With the definition of interpreted systems, we can formally define *perfect recall* and *synchrony*. Perfect recall means that if the agent considers run r' possible at

the point (r, n), in that there is a point (r', n') that the agent cannot distinguish from (r, n), then the agent must have considered r' possible at all times in the past (i.e., at all points (r, k) with $k \leq n$). Synchrony means that if agent considers a point (r', n') possible at the point (r, n), then these two points must have the same clock value, i.e. $n' = n$.

Language and Semantics

Definition 2.17 (TEL Language)**.** *Given a set of agents Ag, the language Temporal Epistemic Logic* TEL *is as follows, in BNF,*

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \psi \mid K_i\varphi \mid C_B\varphi \mid \bigcirc\varphi \mid \varphi \mathcal{U} \psi$$

where, $i \in Ag$, $B \subseteq Ag$

We interpret the language of LTL over the synchronous interpreted systems, which are the interpreted systems satisfying synchrony property. The reason to use synchrony is that the interpreted systems we are going to use in the following chapters will be synchronous. Please refer to [21] for a more general semantics.

Definition 2.18 (TEL Semantics over Synchronous Interpreted Systems)**.** *Given a synchronous interpreted system $\mathcal{I}$, a run r, and a time point m, we have:*

$$\begin{aligned}
(\mathcal{I}, r, m) \models p &::= p \in \pi(r(m)) \\
(\mathcal{I}, r, m) \models \neg\varphi &::= \textit{not } (\mathcal{I}, r, m) \models \varphi \\
(\mathcal{I}, r, m) \models \varphi \wedge \psi &::= (\mathcal{I}, r, m) \models \varphi \textit{ and } (\mathcal{I}, r, m) \models \psi \\
(\mathcal{I}, r, m) \models K_i\varphi &::= \textit{for all runs } r' \textit{ such that } r'_i(m) = r_i(m) : (\mathcal{I}, r', m) \models \varphi \\
(\mathcal{I}, r, m) \models C_B\varphi &::= \textit{for all runs } r' \textit{ such that there is a line of runs } r^0, r^1 \cdots r^k \\
&\quad\; r^0 = r, r^k = r', \forall j \exists i \in B(r_i^j(m) = r_i^{j+1}(m)) : (\mathcal{I}, r', m) \models \varphi \\
(\mathcal{I}, r, m) \models \bigcirc\varphi &::= (\mathcal{I}, r, m+1) \models \varphi \\
(\mathcal{I}, r, m) \models \varphi \mathcal{U} \psi &::= \exists m' \geq m : (\mathcal{I}, r, m') \models \psi \textit{ and } \forall m \leq m'' < m' : (\mathcal{I}, r, m'') \models \varphi
\end{aligned}$$

Complexity

The complexity of model checking problem in TEL is PSPACE-Complete [95]. For more results on temporal epistemic model checking, refer to [84, 83, 46, 80].

The complexity of satisfiability problem in TEL over the synchronous interpreted system is EXPTIME-COMPLETE [33]. The complexity of satisfiability of several temporal epistemic logics allowing the combination of different properties such as synchrony, perfect recall, no learning etc are discussed in [33].

2.2.4 Dynamic Epistemic Logic

Our treatment of Dynamic Epistemic Logic (DEL) is based on [92]. The main new features introduced, compared to epistemic logic, are in two aspects. The first is in syntax; it adds a new type of modality, called action modalities, which allows us to express properties with actions explicitly. The second relates to the semantics; it adds a new semantic structure called *action models*.

Language

Definition 2.19 (DEL Language)**.** *The logical language $\mathcal{L}_{\mathsf{DEL}}$ of Dynamic Epistemic Logic is inductively defined as*

$$\varphi \quad ::= \quad q \mid \neg\varphi \mid (\varphi \wedge \psi) \mid K_i\varphi \mid C_B\varphi \mid [\mathsf{M}, \mathsf{w}]\varphi$$

where q is a propositional atom, i is an agent, and B is a group of agents.

Semantics

To capture a static view of a multi-agent system, we use state models. They are essentially possible world models. Here is the formal definition.

Definition 2.20 (State Model)**.** *A* state model *M for n agents is a structure*

$$\langle W, \sim_1, \cdots, \sim_n, \pi \rangle$$

where W is a finite non-empty set of states, $\sim_i$ is an equivalence relation on W, and π is a valuation function from W to $\wp(P)$.

The introduction of action models is based on this insight: just like agents have uncertainty about possible worlds, they can also have uncertainty about possible actions. Moreover, instead having valuation for worlds, we can have preconditions to actions, indicating the conditions that they could be executed.

Definition 2.21 (Action Model)**.** *An* action model M *for n agents is a structure*

$$\langle \mathsf{W}, \sim_1, \ldots, \sim_n, \mathsf{pre} \rangle$$

where W *is a finite non-empty set of* action points*,* $\sim_i$ *is an equivalence relation on* W*, and* $\mathsf{pre} : \mathsf{W} \to \mathcal{L}_{\mathsf{DEL}}$ *is a precondition function that assigns a* precondition $\mathsf{pre}(\mathsf{w})$ *to each* $\mathsf{w} \in \mathsf{W}$*.*

Let MOD be the class of state models and ACT the class of $\mathcal{L}_{\mathsf{DEL}}$ models. Then $\mathcal{L}_{\mathsf{DEL}}$-update is an operation of the following type:

$$\otimes : \mathrm{MOD} \times \mathrm{ACT} \to \mathrm{MOD}.$$

The operation $\otimes$ and the truth definition for $\mathcal{L}$ are defined by mutual recursion, as follows.

Definition 2.22 (Update, Truth)**.** *Given a state model* M *and an action model* M*, we define*

$$M \otimes \mathsf{M}$$

as

$$(W', R', \pi'),$$

where

$$\begin{aligned} W' ::= &\quad \{(w, \mathsf{w}) \mid w \in W_M, \mathsf{w} \in W_{\mathsf{M}}, M \models_w \mathsf{pre}(\mathsf{w})\}, \\ \pi'(w, \mathsf{w}) ::= &\quad \pi_M(w), \\ (w, \mathsf{w}) \sim_i (w', \mathsf{w}') \in R' ::= &\quad w \sim_i w' \in R_M, \mathsf{w} \sim_i \mathsf{w}' \in R_{\mathsf{M}}, \end{aligned}$$

and where the truth definition is given by:

$$\begin{aligned} M, w \models p &::= p \in V_M(w) \\ M, w \models \neg\varphi &::= \textit{not } M, w \models \varphi \\ M, w \models \varphi \wedge \psi &::= M, w \models \varphi \textit{ and } M, w \models \psi \end{aligned}$$

$$\begin{aligned} M, w \models K_i\varphi &::= \textit{for all } w' \textit{with } w \sim_i w' \ M, w' \models \varphi \\ M, w \models C_B\varphi &::= \textit{for all } w' \textit{with } w \sim_B^* w' \ M, w' \models \varphi \\ M, w \models [\mathsf{M}, \mathsf{w}]\varphi &::= M, w \models \mathsf{pre}(\mathsf{w}) \textit{ implies } M \otimes \mathsf{M}, (w, \mathsf{w}) \models \varphi \end{aligned}$$

where $\sim_B^*$ is the reflexive and transitive closure of $\bigcup_{i \in B} \sim_i$.

Public Announcement Logic

A public announcement is an action which informs the whole group of agents with a public message. Public Announcement Logic (PAL) is an extension of standard multi-agent epistemic logic with dynamic modal operators to model the effects of public announcements. It was originally proposed by Plaza [61]. Plaza used a different notation, without dynamic modal operators, and did not incorporate common knowledge. Later milestones, with common knowledge and also involving further generalizations, are by Gerbrandy et al. [28] and Baltag et al. [10].

Here we treat PAL as a special case of DEL, since the public announcements can be modelled using action models. In the following, we give the language of PAL and its semantics.

Definition 2.23 (PAL Language)**.** *The language of Public Announcement Logic is inductively defined as*

$$\varphi \ ::= \ q \mid \neg\varphi \mid (\varphi \wedge \psi) \mid K_i\varphi \mid C_B\varphi \mid [\varphi]\psi$$

where q is a propositional atom, i is an agent, and B is a group of agents.

Like the language of DEL, the language of PAL is also interpreted over state models. Moreover, a singleton action model with universal access for all agents represents a public announcement. To be more specific, an action model

$$(\mathsf{M}_0, \mathsf{w}_0) = (\langle \{\mathsf{w}_0\}, \sim_1, \cdots, \sim_n, \mathsf{pre} \rangle, \mathsf{w}_0)$$

where $\sim_i = \{(\mathsf{w}_0, \mathsf{w}_0)\}$ for $1 \leq i \leq n$ and $\mathsf{pre}(\mathsf{w}_0) = \varphi$, represents a public announcement of φ. In DEL, formula $[\mathsf{M}_0, \mathsf{w}_0]\psi$ stands for 'after executing action $(\mathsf{M}_0, \mathsf{w}_0)$ it holds that ψ'; while in PAL, formula $[\mathsf{M}_0, \mathsf{w}_0]\psi$ stands for 'after *public*

announcement of φ it holds that ψ'. We simplify the representation of $[\mathsf{M}_0, \mathsf{w}_0]\psi$ to $[\varphi]\psi$.

For the semantics, it basically follows from that of the DEL. We only mention the case with $[\varphi]\psi$.

$$M, w \models [\varphi]\psi ::= M, w \models \varphi \text{ implies } M \otimes \mathsf{M}_0, (w, \mathsf{w}_0) \models \varphi$$

where $(\mathsf{M}_0, \mathsf{w}_0) = (\langle \{\mathsf{w}_0\}, \sim_1, \cdots, \sim_n, \mathsf{pre} \rangle, \mathsf{w}_0)$ with $\sim_i = \{(\mathsf{w}_0, \mathsf{w}_0)\}$ for $1 \leq i \leq n$ and $\mathsf{pre}(\mathsf{w}_0) = \varphi$.

Complexity

As for as I know, the complexities of the model checking and satisifibility problems in DEL are still under investigation, but some concrete results in PAL have been made. In [48], C. Lutz showed that the complexity of the satisifibility problem in PAL is EXPTIME-Complete.

2.3 Summary

This chapter provided a logical background for the work that is going to be presented. It first introduced temporal logics from a single-agent perspective and a multi-agent perspective. From the single-agent perspective, Linear-time Temporal Logic (LTL) and Computational Tree Logic (CTL) were introduced. From the multi-agent perspective, Alternating-time Temporal Logic (ATL) was introduced. It then turned to epistemic logics, which are the logical frameworks of knowledge. Epistemic Logic (EL) was firstly introduced to deal with the agents' knowledge and higher-order knowledge, including common knowledge. Two extensions of EL were then introduced. One is Temporal Epistemic Logic (TEL), which models the temporal changes of a system in addition to the agents' knowledge. The other is Dynamic Epistemic Logic (DEL), which also models the changes of a system but through actions. Chapter 3 and 4 will be related to ATL; Chapter 5 and 6 will be related to TEL and DEL.

Chapter 3

Bridging GDL and ATL

3.1 Introduction

Game playing competitions, particularly between humans and computers, have long been part of the culture of artificial intelligence. Indeed, the victory of Deep Blue over then world champion chess player Gary Kasparov in 1997 is regarded as one of the most significant events in the history of AI. However, a common objection to such specialised competitions and dedicated game playing systems is that they explore only one very narrow aspect of intelligence and rationality. To overcome these objections, in 2005 AAAI introduced a *general game playing competition*[1], intended to test *the ability to play games in general*, rather than just the ability to play a specific game [59, 26]. Participants in the competition are computer programs, which are provided with the rules to previously unknown games during the competition itself; they are required to play these games, and the overall winner is the one that fared best overall. Note that the participant programs were required to interpret the rules of the games *themselves*, without human intervention or interpretation. The Game Description Language (GDL) is a special purpose, computer processable language, which was developed in order to define the games played by participant programs. Thus, a participant must be able to interpret game descriptions expressed in GDL, and then play the game autonomously.

Since GDL is a language for defining games, it seems very natural to investigate the problem of reasoning about games defined in GDL. Just as the designer of a computer communications protocol might want to use model checking tools

[1]URL: http://games.stanford.edu

to investigate the properties of the protocol (ensure it is deadlock-free, etc [16]), so the GDL game designer will typically want to investigate the properties of games. In addition to checking protocol-like properties such as deadlock-freeness, the fact that GDL is used for describing *games* suggests a whole new class of properties to check: those relating to the *strategic properties* of the game being defined, such as the properties showing whether a particular agent or a coalition of agents have a strategy to win.

One formalism for reasoning about games that has attracted much interest is Alternating-time Temporal Logic (ATL) (see Section 2.1.5). The basic construct of ATL is the *cooperation modality*, $\langle\langle C\rangle\rangle\varphi$, where C is a collection of agents, meaning that coalition C can cooperate to achieve φ; more precisely, that C have a winning strategy for φ. ATL has been widely applied to reasoning about game-like multi-agent systems in recent years, and has proved to be a powerful and expressive tool for this purpose [6, 30, 57, 58, 86, 81].

The aim of this chapter is to make a concrete link between ATL and GDL. Specifically, it shows that GDL descriptions can be interpreted as specifications of an ATL model, and that ATL can thus be interpreted over GDL descriptions.

This chapter is structured as follows. In Section 3.2, we introduce the Game Description Language, and the construction of game models from GDL descriptions. In Section 3.3, we show it is possible to translate a propositional GDL description into an ATL formula that is equivalent up to alternating-bisimulation, and which is only polynomially larger than the original GDL description. As a corollary, we are able to characterise the complexity of ATL reasoning about propositional GDL games: the problem is EXPTIME-Complete.

3.2 Game Description Language and Game Models

GDL is a specialised language, intended[2] for defining games [26]. A game description must define the states of the game, a unique initial state, and the players in the game ("roles" in GDL parlance). For every state and every player, the game description must define the moves (a.k.a. actions) available to that player in that state, as well as the state transition function of the game – how moves

[2]Please note that GDL can also be used as a specification language for a large class of multi-agent environments (see [68]).

```
01 (role xplayer)
02 (role oplayer)
03 (init (cell 1 1 b))
    ...
11 (init (cell 3 3 b))
12 (init (control xplayer))
13 (<= (next (cell ?m ?n x))
14     (does xplayer (mark ?m ?n))
15     (true (cell ?m ?n b)))
    ...
28 (<= (next (control oplayer))
29     (true (control xplayer)))
30 (<= (row ?m ?x)
31     (true (cell ?m 1 ?x))
32     (true (cell ?m 2 ?x))
33     (true (cell ?m 3 ?x)))
   ...
54  (<= (legal ?w (mark ?x ?y))
55      (true (cell ?x ?y b))
56      (true (control ?w)))
57  (<= (legal oplayer noop)
58      (true (control xplayer)))
    ...
61  (<= (goal xplayer 100)
62      (true (line x)))
    ...
77  (<= terminal
78      (line x))
79  (<= terminal
80      (line o))
81  (<= terminal
82      (not open))
```

Figure 3.1: A fragment of a game in the Game Description Language

transform the state of play. Finally, it must define what constitutes a win, and when a game is over. The approach adopted by GDL is to use a *logical* definition of the game. We introduce GDL by way of an example (Figure 3.1): a version of "Tic-Tac-Toe" . In this game, two players take turns to mark a 3×3 grid, and the player who succeeds in placing three of its marks in a row, column, or diagonal wins.

GDL uses a prefix rule notation based on LISP. The Tic-Tac-Toe game in Figure 3.1 consists of 82 lines. The first two lines, (`role xplayer`) and (`role oplayer`), define the two players in this game. The following `init` lines (lines 03-12) define facts true in the initial state of the game (all the cells are blank, and `xplayer` has the control of the game). The following rule (line 13-15) defines the effect of making a move: if $cell(m, n)$ is blank (`cell ?m ?n b`), and `xplayer` marks it, then in the `next` state, it will be true that $cell(m, n)$ is marked by `x`: (`cell ?m ?n x`). The next rule (line 28-29) says that if the current state is under the control of `xplayer`, then the next state will be under the control of `oplayer`. Lines 30-33 define what it means to have a `row` of symbols (we omit a number of related rules). The `legal` rule (line 54-56) defines when it is `legal` for a player `?w` to perform a mark action. The `goal` rule (line 61-62) defines the aim of the game: it says that the `xplayer` will get a reward of 100 if it brings about a line marked by `x`. The final, `terminal` rules (line 77-82) define when the game has ended.

Overall, a GDL description consists of a list of such rules, and the semantics of these rules are similar to logic programming languages. Certain operators in a GDL description have a special meaning: `role` (used to define the players

of the game); `init` (defining initial facts); `legal` (defining pre-conditions for actions); and `goal` (defining rewards for agents). An additional operator, `true`, is sometimes used, to make explicit that a particular expression should be true in the current state of the game.

While GDL in [26] permits predicates such as (`cell ?m ?n b`), we simplify this by allowing only nullary predicates, i.e. propositions. We can do this via instantiation of the predicates, i.e. replacing variables with their values. For example, variables like ?`m`, ?`n` are replaced by elements in their domain $\{1, 2, 3\}$. Thus (`cell ?m ?n b`) is instantiated as (`cell 1 1 b`), (`cell 1 2 b`), $\cdots$, (`cell 3 3 b`). It is easy to see that the rule in (line 13-15) is replaced by 9 rules with no predicates, and in general, there will inevitably be an undesirable blow-up in the number of rules when translating from arbitrary predicate form; nevertheless, the translation is possible, a point that is implicitly used in what follows. We refer to (`cell 1 1 b`) as a nullary predicate, or an atomic proposition. We will refer to our fragment of GDL as propositional GDL in the remainder of this thesis.

In what follows, we will formally define the interpretation of GDL descriptions with respect to game models. As GDL is based on DATALOG, a logical programming language, we begin by introducing DATALOG.

3.2.1 Datalog Programs

DATALOG is a query and rule language for deductive databases that, syntactically, is a subset of PROLOG [17]. GDL uses DATALOG as a basis to specify game rules. As we mentioned above, we deal with propositional GDL. Accordingly we only introduce the propositional fragment of DATALOG. We will give the syntax and semantics of DATALOG rules, and then illustrate how to build a game model from a GDL description, based on the DATALOG semantics.

Definition 3.1 (DATALOG: Language, Rules and Programs)**.** *The basic unit of the* DATALOG *Language consists of a set of atomic propositions* $\Pi = \{p, q, \dots\}$. *Let* $\ell(\Pi)$ *be the set of literals over* Π*:* $\ell(\Pi) = \Pi \cup \{\neg p \mid p \in \Pi\}$.

A DATALOG *rule is of the form* $(\Leftarrow p, \ell_1, \dots, \ell_n)$ *where* $p \in \Pi$ *and* $\ell_i \in \ell(\Pi)$ $(i \leq n)$. *If the displayed rule is called* r*, we call* p *its head* $(p = hd(r))$ *and the body of* r*,* $bd(r)$*, is the set* $\{\ell_1, \dots, \ell_n\}$. *Note that a body can be empty.*

A DATALOG *Program is a set of* DATALOG *rules.*

Definition 3.2 (Dependency Graphs for DATALOG Programs)**.** *Let a* DATALOG *program* Δ *be given. Its* Dependency Graph $\mathcal{DG}(\Delta)$ *is a labelled directed graph* $\langle \Pi, lab^+, lab^- \rangle$*, where*

- Π *is the set of atoms in* Δ*; Each atom serves as a node in the graph.*
- $lab^+(p, h)$ *(i.e. an edge from* p *to* h *labelled with* $+$*) iff there is a rule* $r \in \Delta$ *with* $h = hd(r)$ *and* $p \in bd(r)$*.*
- $lab^-(p, h)$ *iff there is a rule* $r \in \Delta$ *with* $h = hd(r)$ *and* $\neg p \in bd(r)$*.*

A model for a DATALOG program is a set of atomic propositions.

Definition 3.3 (Models for DATALOG Programs)**.** *Given a* DATALOG *program* Δ*,* $\Sigma \subseteq \Pi$ *is a model of* Δ *if and only if it satisfies the following conditions:*

- *if* $(\Leftarrow p) \in \Delta$*, then* $p \in \Sigma$*;*
- *if* $(\Leftarrow p, bd) \in \Delta$ *and* $pos(bd) \subseteq \Sigma$ *and* $neg(bd) \cap \Sigma = \emptyset$*, then* $p \in \Sigma$*, where* $pos(bd)$ *is the set of positive literals in* bd *and* $neg(bd)$ *is the set of negative literals.*

Notice that there can be several models for a given DATALOG program. For example, the program $\{(\Leftarrow p, q), (\Leftarrow q, \neg p)\}$ has two models: $\{p\}$ and $\{p, q\}$. But for some DATALOG programs, there is a unique model.

Definition 3.4 (Stratified and Acyclic DATALOG Programs)**.** *A* DATALOG *program* Δ *is called* stratified *if its dependency graph* $\mathcal{DG}(\Delta)$ *contains* no cycles with a "$-$" label*. An atom* p *is said to be in stratum* $i \in \mathbb{N}$ *if the maximum number of edges labelled "*$-$*" on any path ending at* $p \in \mathcal{DG}(\Delta)$ *is* i*. A rule* $r \in \Delta$ *is of stratum* i *if* $hd(r)$ *is in stratum* i*. A* DATALOG *program* Δ *is called* acyclic *if* $\mathcal{DG}(\Delta)$ *contains no cycles.*

Definition 3.5 (DATALOG Semantics)**.** *Given a stratified* DATALOG *program* Δ*, we define its model* $s = DPMod(\Delta)$ *as follows. First of all, let* $t_0 = \{p \mid (\Leftarrow p) \in \Delta\}$*. Suppose* t_i *is defined, initialise* s_i *to* t_i *and, as long as there is a rule* $(\Leftarrow p, \ell_1, \ldots, \ell_n)$ *in stratum* i *such that* $s_i \models \ell_1 \wedge \cdots \wedge \ell_n$*, add* p *to* s_i*. After this, put* $t_{i+1} = s_i$*. If the maximum stratum of* Δ *is* k*, put* $s = t_{k+1}$*.*

Stratification guarantees that, when computing a model for Δ, whenever we have a literal $\neg q$ in the body of a rule r, we will consider all rules r' with $hd(r') =$

q before considering r. Δ is acyclic iff there is a level mapping $f : \ell(\Pi) \to \mathbb{N}$ for which $f(p) = f(\neg p)$ and for every rule $(\Leftarrow p, \ell_1, \ldots, \ell_n)$ in Δ, $f(p) > f(\ell_i)$, for all $i \leq n$.

Theorem 3.1 ([8]). *The model s defined in Definition 3.5 is a unique model for Δ, and it does not depend on the particular stratification.*

Definition 3.6 (Completion of Δ). *Given an acyclic* DATALOG *program Δ, the completion of Δ is a set of formulas $CP(\Delta)$ as follows. Let the* definition $\mathcal{D}(\Delta, p)$ *of p be the set of rules r in Δ for which $hd(r) = p$. Then let*

$$cp(p) = (p \leftrightarrow \bigvee_{r \in \mathcal{D}(\Delta,p)} \bigwedge bd(r))$$

where $bd(r) = \{\ell_1, \cdots, \ell_n\}$ and $\bigwedge bd(r) = \ell_1 \wedge \cdots \wedge \ell_n$; for every empty body $bd(r)$, $\bigwedge bd(r) = \top$. Note that, if p does not occur as a head in any rule in Δ, we have $cp(p) = \neg p$. Finally, the Clark completion $CP(\Delta)$ of a DATALOG *program Δ over Π is simply $\{cp(p) \mid p \in \Pi\}$.*

Theorem 3.2. *Let Δ be an acyclic program, and Π be the set of atoms in Δ. For all $p \in \Pi$, we have*

$$p \in DPMod(\Delta) \quad \textit{iff} \quad CP(\Delta) \models_{cl} p,$$

where $DPMod(\Delta)$ is the unique model of the stratified program Δ, and the set $CP(\Delta)$ is the Clark completion of Δ and $\models_{cl}$ denotes consequence in classical logic.

3.2.2 GDL Game Descriptions

We now formally define GDL game descriptions.

Definition 3.7 (GDL Syntax). *Let a primitive set of proposition symbols $Prim = \{\hat{\mathtt{p}}, \hat{\mathtt{q}} \cdots\}$, a set of agents Ag, a set of actions Ac, a set of strings S, and a set of integers [0..100] be given. The set of* atomic propositions *of* GDL, $\mathsf{At}_{\mathrm{GDL}}$, *is the minimal set satisfying the following conditions: $Prim \subseteq \mathsf{At}_{\mathrm{GDL}}$; a special atom $\mathtt{terminal} \in \mathsf{At}_{\mathrm{GDL}}$; for two strings $\mathtt{s}_1, \mathtt{s}_2 \in S$, $(\mathtt{distinct}\ \mathtt{s}_1\ \mathtt{s}_2) \in \mathsf{At}_{\mathrm{GDL}}$; for every agent $\mathtt{i} \in Ag$ and action $\mathtt{a} \in Ac$, $(\mathtt{legal}\ \mathtt{i}\ \mathtt{a}) \in \mathsf{At}_{\mathrm{GDL}}$; for every agent $\mathtt{i}$ and an $\mathtt{v}$ integer in [0..100]), $(\mathtt{goal}\ \mathtt{i}\ \mathtt{v}) \in \mathsf{At}_{\mathrm{GDL}}$.*

The set of atomic expressions $\mathsf{AtExpr}_{\mathrm{GDL}}$ *of* GDL*, is the minimal set satisfying the following conditions:*

- *for* $\mathtt{p} \in \mathsf{At}_{\mathrm{GDL}}$, $\{\mathtt{p}, (\mathtt{init\ p}), (\mathtt{next\ p}), (\mathtt{true\ p})\} \subseteq \mathsf{AtExpr}_{\mathrm{GDL}}$;
- *for every agent* $\mathtt{i}$ *and action* $\mathtt{a}$, $\{(\mathtt{role\ i}), (\mathtt{does\ i\ a})\} \subseteq \mathsf{AtExpr}_{\mathrm{GDL}}$.

$\mathsf{LitAt}_{\mathrm{GDL}}$ *is* $\{\mathtt{p}, (\mathtt{true\ p}), (\mathtt{not\ p}), (\mathtt{not\ (true\ p)}) \mid \mathtt{p} \in \mathsf{At}_{\mathrm{GDL}}\}$. $\mathsf{LitExpr}_{\mathrm{GDL}}$ *is* $\mathsf{AtExpr}_{\mathrm{GDL}} \cup \mathsf{LitAt}_{\mathrm{GDL}}$.

A *game description* specifies the atoms from $\mathsf{At}_{\mathrm{GDL}}$ that are true, either in the initial state, or as a result of global constraints, or as the effect of performing some joint actions in a given state.

Definition 3.8 (GDL Game Descriptions)**.** *A* GDL *game description* Γ *is a set of* DATALOG *rules* r *of the form* [3] $(\Leftarrow (\mathtt{h})(\mathtt{e}_1)\ldots(\mathtt{e}_m))$ *where* $\mathtt{h}$*, the head* $hd(r)$ *of the rule, is an element of* $\mathsf{AtExpr}_{\mathrm{GDL}}$ *and each* $\mathtt{e}_i$ *(*$i \in [1..m]$*) in the body* $bd(r)$ *of* r *is a literal expression from* $\mathsf{LitExpr}_{\mathrm{GDL}}$*. If* $m = 0$*, we say that* r *has* an empty body*. We can split every game description* Γ *into four different types of rules where:*

- $\Gamma_{\mathtt{role}}$ *contains all claims of the form* $(\Leftarrow (\mathtt{role\ x}))$*. They specify the agents in the game.*
- $\Gamma_{\mathtt{init}}$ *is a set of initial rules of the form* $(\Leftarrow (\mathtt{init\ p}))$*, which has an empty body and its head represent an initial constraints of the game.*
- $\Gamma_{\mathtt{glob}}$ *is a set of global rules of the form* $(\Leftarrow (\mathtt{p})\ (\mathtt{e}_1)\ \ldots\ (\mathtt{e}_m))$*, where* $\mathtt{p} \in \mathsf{At}_{\mathrm{GDL}}$ *and each body* $\mathtt{e}_i$ *(*$i \in [1..m]$*) is from* $\mathsf{LitAt}_{\mathrm{GDL}}$*.*
- $\Gamma_{\mathtt{next}}$ *is a set of next rules of the form* $(\Leftarrow (\mathtt{next\ p})(\mathtt{e}_1)\ldots(\mathtt{e}_m))$ *where each* $\mathtt{e}_i (i \in [1..m])$ *is from* $\mathsf{LitAt}_{\mathrm{GDL}}$ *or of the form* $(\mathtt{does\ i\ a})$*.*

3.2.3 GDL Game Models

Given a GDL game description, we specify how to compute the corresponding game model. In general, a game model can be seen as a game tree, where a set of nodes represent states of the game, and a labelled edge from one state to another

[3] We do not allow disjunction in the body of a GDL rule as in [26]. A rule like $(\Leftarrow (\mathtt{h})(\mathtt{e}_1 \vee \mathtt{e}_2))$ can be replaced by its equivalence, two rules $(\Leftarrow (\mathtt{h})(\mathtt{e}_1))$ and $(\Leftarrow (\mathtt{h})(\mathtt{e}_2))$.

represents a transition from one state to another caused by the performance of actions/moves by players.

We will shortly consider how to compute such game states from game descriptions.

For the description of Game Models G, our approach is equivalent to that of [26]. Instead of *roles* we will refer to a set $Ag = \{1, \ldots, n\}$ of *agents* or *players*. Given the set of atomic propositions $\mathsf{At}_{\mathrm{GDL}}$, a *Game Model* is a structure:

$$G = \langle S, s_0, Ag, Ac_1, \cdots, Ac_n, \tau, \pi \rangle$$

where S is a set of game states; $s_0 \in S$ is the initial state of G; Ag denotes the set of agents, or players in the game; Ac_i is the set of possible actions or moves for agent i; $\tau : Ac_1 \times \cdots \times Ac_n \times S \to S$ is such that $\tau(\langle a_1, \ldots, a_n \rangle, s) = u$, means that if in game state s, agent i chooses action a_i, $(i \leq n)$, the system will change to its successor state u – we require all states, except the initial state, have only one predecessor; and finally, $\pi : S \to 2^{\mathsf{At}_{\mathrm{GDL}}}$ is a valuation function, which associates with each state the set of atomic propositions in $\mathsf{At}_{\mathrm{GDL}}$ that are true in that state. We will often abbreviate an action profile $\langle a_1, \ldots, a_n \rangle$ to $\vec{a}$. (Note that we do not include the subset $T \subseteq S$ included in the game models of [26]. This subset is supposed to denote the terminal states: we can obtain this set in G by simply collecting all the states that satisfy `terminal`.)

Now we specify when a game model G is a model for a game description Γ; this makes precise the informal description of [26], and in fact represents a formal semantics for GDL.

We compute the game model $GMod(\Gamma)$ for a game description Γ as follows. The main idea is that every state $s \in S$ of $GMod(\Gamma)$ is associated with the unique model under the stratified semantics of some DATALOG Program Δ that is derived from Γ. In particular, let $\delta(\Gamma_{\texttt{glob}})$ be derived from $\Gamma_{\texttt{glob}}$ by replacing every occurrence of `true p` by `p`. Since we assume that Δ does not contain `init` or `next` in any body of any rule, $\delta(\Gamma_{\texttt{glob}})$ is indeed a DATALOG Program. Also, let $\delta(\Gamma_{\texttt{init}})$ be $\{\Leftarrow \texttt{p} \mid (\Leftarrow \texttt{init p}) \in \Gamma_{\texttt{init}}\}$. The set Ag of agents, and Ac_i of actions for agent i in $GMod(\Gamma)$ are immediately read off from Γ: $Ag = \{i \mid (\Leftarrow (\texttt{role i})) \in \Gamma_{\texttt{role}}\}$ and $Ac_i = \{\texttt{a} \mid (\texttt{legal i a}) \text{ occurs in } \Gamma\}$.

In the following, we construct S, τ, and π step by step.

- *First*, we define the initial state s_0. Put

$$\pi(s_0) = DPMod(\delta(\Gamma_{\texttt{init}}) \cup \delta(\Gamma_{\texttt{glob}}))$$

- *Next*, suppose a game state $s \in S$ has already been defined. If this is not a terminal state, i.e. $\texttt{terminal} \notin \pi(s)$, each agent should have at least one legal action available. An action a_i is legal for agent i in state s, if and only if $(\texttt{legal i a}_\texttt{i}) \in \pi(s)$. If $\texttt{terminal} \notin \pi(s)$, we define, for every profile of legal actions $\langle a_1, \cdots, a_n \rangle$, a successor u of s by first computing the atoms that need to be true due to $\Gamma_{\texttt{next}}$.

$$\begin{aligned} F_\Gamma(\langle a_1, \ldots, a_n \rangle, s) = \quad & \{\Leftarrow \texttt{ p} \mid \exists\, (\Leftarrow (\texttt{next p})\ (\texttt{e}_1) \cdots (\texttt{e}_m)) \in \Gamma_{\texttt{next}} \& \\ & \pi(s) \cup \{(\texttt{does i a}_i) \mid i \in Ag\} \models_{cl} \texttt{e}_1 \wedge \cdots \wedge \texttt{e}_m\} \end{aligned}$$

  So, $F_\Gamma(\langle a_1, \ldots, a_n \rangle, s)$ computes those atoms that need to be true in the next state (the F is for 'forward'), given that each agent i performs a_i. Now we add:

$$u = \tau(\langle a_1, \ldots, a_n \rangle, s)\ \&\ \pi(u) = DPMod(F_\Gamma(\langle a_1, \ldots, a_n \rangle, s) \cup \delta(\Gamma_{\texttt{glob}}))$$

- *Iteration:* we repeat the above procedure to all the descendants of the initial state, until we reach all the terminal states.

Note that atoms of the form $(\texttt{does i a}_\texttt{i})$ are not added to the game model $GMod(\Gamma)$, as they are only used to calculate different successors for a given game state s. So, they incorporate a kind of hypothetical reasoning of the form: "suppose player i were to do a_i, what would be the resulting next state?"

We illustrate the above procedure partially by the following example related to Tic-Tac-Toe.

Example 3.1. *Suppose that we already have a propositional version of the* GDL *description presented in figure 3.1, i.e. all the variables have been instantiated. As* $(\texttt{control xplayer}) \in \delta(\Gamma_{\texttt{init}})$*, we use* $\Gamma_{\texttt{glob}}$ *and get* $(\texttt{legal xplayer (mark 1 1)}) \in \pi(s_0)$*, and* $(\texttt{legal oplayer noop}) \in \pi(s_0)$*. We also see that* $\texttt{terminal} \notin \pi(s_0)$*, because the bodies of all the global rules with head* terminal *cannot be satisfied. Thus we have an action profile* $\vec{a} = \langle \texttt{mark 1 1}, \texttt{noop} \rangle$*. It is easy to verify that* $(\texttt{cell 1 1 x})$ *and* $(\texttt{control oplayer}) \in F_\Gamma(\vec{a}, s_0)$*, due to the* next *rules.*

We can now compute a game model from a GDL game description. Next we give meanings to the literal atomic propositions in $\mathsf{LitAt}_{\mathrm{GDL}}$ and the GDL rules with respect to such game models.

Definition 3.9 (GDL Semantics)**.** *Let $G = \langle S, s_0, Ag, Ac_1, \cdots, Ac_n, \tau, \pi\rangle$ be a game model derived from a game description Γ. Let $\{i_1, \ldots, i_k\} = Ag'$ be a set of agents $\subseteq Ag$, each i_x with an action a_x ($x \leq k$). Then we say that t is an $i_1 : a_1, \ldots, i_k : a_k$ successor of s if there is a choice for any agent j in $Ag \setminus Ag'$ for an action b_j from Ac_j such that $\tau(\langle c_1, \ldots, c_n\rangle, s) = t$, where $c_v = a_x$ if $v = i_x \in Ag'$, and $c_v = b_j$ if $v = j \in Ag \setminus Ag'$. For any state $s \in S$, $\mathtt{p} \in \mathsf{At}_{\mathrm{GDL}}$, we have:*

- $G, s \models_{\mathrm{GDL}} \mathtt{p}$ *iff* $\mathtt{p} \in \pi(s)$;
- $G, s \models_{\mathrm{GDL}} \mathtt{not\ p}$ *iff* $G, s \not\models_{\mathrm{GDL}} \mathtt{p}$;
- $G, s \models_{\mathrm{GDL}} \mathtt{true\ p}$ *iff* $G, s \models_{\mathrm{GDL}} \mathtt{p}$;
- $G, s \models_{\mathrm{GDL}} \mathtt{not\ (true\ p)}$ *iff* $G, s \not\models_{\mathrm{GDL}} \mathtt{true\ p}$;

and, for any rules in $\Gamma_{\mathtt{init}} \cup \Gamma_{\mathtt{glob}} \cup \Gamma_{\mathtt{next}}$, we have,

- $G \models_{\mathrm{GDL}} (\Leftarrow (\mathtt{init\ p}))$ *iff* $G, s_0 \models_{\mathrm{GDL}} \mathtt{p}$;
- $G \models_{\mathrm{GDL}} (\Leftarrow\ (\mathtt{p})\ (\mathtt{e}_1) \ldots (\mathtt{e}_m))$ *iff* $\forall s : (\forall i \in [1..m] :\ G, s \models_{\mathrm{GDL}} \mathtt{e}_i) \Rightarrow G, s \models_{\mathrm{GDL}} \mathtt{p}$;
- $G \models_{\mathrm{GDL}} (\Leftarrow (\mathtt{next\ p})(\mathtt{e}_1) \ldots (\mathtt{e}_m)(\mathtt{does\ i_1\ a_1}) \ldots (\mathtt{does\ i_k\ a_k}))$ *iff* $\forall s, t :$ $(\forall i \in [1..m] :\ G, s \models_{\mathrm{GDL}} \mathtt{e}_i$ *and* t *is an* $i_1 : a_1, \ldots i_k : a_k$ *successor of* $s)$ $\Rightarrow G, t \models_{\mathrm{GDL}} \mathtt{p}$.

Note that we do not interpret the rules in $\Gamma_{\mathtt{role}}$ as the agents are already fixed with the game model.

3.3 Linking GDL and ATL

From previous sections, we can see that GDL and ATL are intimately related at the semantic level: GDL is a language for defining games, while ATL gives a language for expressing properties of such games. The difference between the two languages is that GDL takes a relatively *constructive*, *internal* approach to

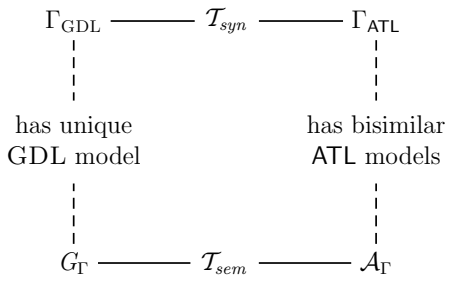


Figure 3.2: The relation between a GDL description of a game Γ_{GDL} and its related ATL-theory Γ_{ATL}.

a game description, essentially defining how states of the game are constructed and related by possible moves. In contrast, ATL takes an *external*, *strategic* view: while it seems an appropriate language with which to express potential strategic properties of games, it is perhaps not very appropriate for defining games.

In this section, we answer the following question: how complex is it to interpret a property, represented by an ATL formula, over a game represented by a GDL description? We do this by building two links between GDL and ATL:

- On the semantic level, every GDL description Γ has an ATL *model* associated with it.
- On the syntactic level, every GDL description Γ has an ATL *theory* associated with it.

Let us now be more precise about the links between GDL and ATL (see Figure 3.2). We start from any game G with GDL description Γ_{GDL}. On the semantic level, we use stratified semantics and the tree representation of [26] to construct a unique model G_Γ from Γ_{GDL}. This model has a natural associated ATL-model $\mathcal{A}_\Gamma = \mathcal{T}_{sem}(G_\Gamma)$. In Section 3.3.1, we will show $\mathcal{T}_{sem}$. On the syntactic level, we provide a translation $\mathcal{T}_{syn}$ that transforms the GDL specification Γ_{GDL} into an ATL theory $\Gamma_{\mathsf{ATL}} = \mathcal{T}_{syn}(\Gamma_{\mathrm{GDL}})$. In Section 3.3.2, we will describe our tool that implements $\mathcal{T}_{syn}$. We further show that this transformation is correct, in the following sense: all ATL-models satisfying Γ_{ATL} are bisimilar to $\mathcal{A}_\Gamma$. So Γ_{ATL} can be said to characterise the ATL-theory of the game G. And, one has, for any GDL-formula γ, that $G_\Gamma \models_{\mathrm{GDL}} \gamma$ iff $\mathcal{T}_{sem}(G_\Gamma) \models_{\mathsf{ATL}} \mathcal{T}_{syn}(\gamma)$, where $\models_{\mathrm{GDL}}$ denotes the semantics of GDL and $\models_{\mathsf{ATL}}$ denotes the semantics of ATL.

We now explore these two links in more detail.

3.3.1 $\mathcal{T}_{sem}$: From GDL Game Models to ATL Game Models

Suppose that we have already constructed a game model G from a GDL description Γ, using the methods in Section 3.2.3. It is not yet possible to interpret an ATL formula on this model G. In this section, we transform G into an AATS, the ATL game structure on which we can interpret ATL formulas.

Given a GDL game model $G = \langle S, s_0, Ag, Ac_1, \cdots, Ac_n, \tau, \pi \rangle$ and a set of atomic propositions $\mathsf{At}_{\mathrm{GDL}}$, we can define an associated AATS, $\mathcal{A}_G = \langle Q, q_0, Ag, Ac_1 \cup \{fin_1\}, \ldots, Ac_n \cup \{fin_n\}, \rho, \tau', \Phi, \pi' \rangle$, with the same sets of agents Ag such that Φ is constructed from $\mathsf{At}_{\mathrm{GDL}}$ in the following manner.

Definition 3.10 (Translation t and t_{old})**.** *Define a translation* $t : \mathsf{At}_{\mathrm{GDL}} \to \mathsf{At}_{\mathsf{ATL}}$*, where we associate every atom in* $\mathsf{At}_{\mathrm{GDL}}$ *with an atom in* $\mathsf{At}_{\mathsf{ATL}}$*.*

$t(\hat{\mathtt{p}})$	$::=$	$\hat{p}$	$t(\mathtt{goal\ i\ v})$	$::=$	$goal(i, v)$
$t(\mathtt{legal\ i\ a})$	$::=$	$legal(i, a)$	$t(\mathtt{terminal})$	$::=$	$terminal$
$t(\mathtt{distinct\ s_1\ s_2})$	$::=$	$distinct(s_1, s_2)$			

Let t_{old} *be as follows:* $t_{old}(\mathtt{p}) ::= p_{old}$*.*

We add four types of atomic propositions to Φ.

i. atoms representing the current state of the game: for every $\mathtt{p}$ in $\mathsf{At}_{\mathrm{GDL}}$, add $t(\mathtt{p})$ to Φ.

ii. atoms representing the previous state of the game: for every $\mathtt{p}$ in $\mathsf{At}_{\mathrm{GDL}}$, add $t_{old}(\mathtt{p})$ to Φ.

iii. atoms representing actions that are done in the transition from previous state to current state: add atom $done(i, a)$ to Φ for each $(\mathtt{does\ i\ a})$.

iv. atoms distinguishing the initial and end states of the game: add *init* for initial state and a special atom $z_\perp$. It denotes a special 'zink state', z, which we add to $\mathcal{A}_G$ in order to make it a proper AATS. The idea is that z is the only successor of every terminal state and itself. ;

The other elements of $\mathcal{A}_G$ are:

- $Q = S \cup \{z\}$, where z is a *zink state*, and $q_0 = s_0$;
- $\rho : Ac_{Ag} \to 2^Q$ is the *action precondition function*, which agrees, for each agent, with $legal(i, a_i)$, i.e., $\rho(a_i) = \{q \mid q \models_{\mathsf{ATL}} legal(i, a_i) \wedge \neg terminal, q \in Q\}$. Moreover, $\rho(fin_i) = \{z\} \cup \{q \mid q \models_{\mathsf{ATL}} terminal, q \in Q\}$, for every agent i.
- $\tau' : Ac_1 \times \cdots \times Ac_n \times Q \to Q$ is based on τ. We keep all the mappings in τ and add these: $\tau'(\langle fin_1, \ldots, fin_n\rangle, q) = z$, for all $q \in \{z\} \cup \{q \mid q \models_{\mathsf{ATL}} terminal, q \in Q\}$;
- $\pi' : Q \to 2^\Phi$ is such that $\pi'(q)$ is the minimal set satisfying the following conditions:
  - $init \in \pi(q_0)$, and $z_\perp \in \pi(z)$,
  - $\pi'(q) \supseteq \{t(\mathtt{p}) \mid \mathtt{p} \in \pi(q)\}$ for all $q \in Q \setminus \{z\}$,
  - $\forall q, q' \in Q \setminus \{z\}$ and an action profile $\vec{a} = \langle a_1, \cdots, a_n\rangle$ such that $q' = \tau'(\vec{a}, q)$, we require $\forall i \in Ag, done(i, a_i) \in \pi'(q')$, and $\{t(\mathtt{p})_{old} \mid \mathtt{p} \in \pi(q)\} \subseteq \pi'(q')$. Moreover $\forall i \in Ag, done(i, fin_i) \in \pi'(z)$.

Our intuition behind π' is that each state, except q_0 and z, has exactly one *done*-proposition for each agent to record the action made in its unique predecessor, and a set of p_{old} to record the atomic propositions that is true in that same predecessor.

Given a game description Γ, we have two game models G and $\mathcal{A}_G$. To show that they correspond in all the games rules in Γ, we first define a translation from GDL rules to ATL formulas.

Definition 3.11 (Translation from GDL rules to ATL formulas)**.** *Let* Γ *be a* GDL *game description. A translation from any* GDL *rules in* $\Gamma_{\mathtt{init}} \cup \Gamma_{\mathtt{glob}} \cup \Gamma_{\mathtt{next}}$ *to* ATL *formulas* $\mathcal{R} : \text{GDL} \to \mathsf{ATL}$ *is defined as follows:*

- $\mathcal{R}(\Leftarrow (\mathtt{init\ p})) ::= init \to t(\mathtt{p})$
- $\mathcal{R}(\Leftarrow (\mathtt{p})(\mathtt{e}_1) \ldots (\mathtt{e}_m)) ::= \langle\langle\rangle\rangle\Box(\neg z_\perp \wedge \mathcal{R}(\mathtt{e}_1) \wedge \cdots \wedge \mathcal{R}(\mathtt{e}_m) \to t(\mathtt{p}))$
- $\mathcal{R}(\Leftarrow (\mathtt{next\ p})(\mathtt{e}_1) \ldots (\mathtt{e}_m)(\mathtt{does\ i_1\ a_1}) \ldots (\mathtt{does\ i_k\ a_k})) ::= \langle\langle\rangle\rangle\Box(\neg z_\perp \wedge \mathcal{R}(\mathtt{e}_1) \wedge \cdots \wedge \mathcal{R}(\mathtt{e}_m) \to \langle\langle\{i_1, \ldots i_k\}\rangle\rangle \bigcirc (t(\mathtt{p}) \wedge done(i_1, a_1) \wedge \cdots \wedge done(i_k, a_k)))$

where $t : \mathsf{At}_{\mathrm{GDL}} \to \mathsf{At}_{\mathsf{ATL}}$ *is as in Definition 3.10, and for a* GDL *expression* $\mathtt{e}_i$*, we stipulate:* $\mathcal{R}(\mathtt{e}_i) ::= t(\mathtt{p})$ *if* $\mathtt{e}_i =$ `p` *or* `true p`*, and* $\mathcal{R}(\mathtt{e}_i) ::= \neg t(\mathtt{p})$ *if* $\mathtt{e}_i =$ `not p` *or* `not (true p)`*.*

Now we are ready to show that two game models G and $\mathcal{A}_G$ of game description Γ do correspond in all the games rules in Γ.

Theorem 3.3. *Let* Γ *be a* GDL *game description,* $G = GMod(\Gamma)$ *be its game model, and* $\mathcal{A}_G$ *be the associated* AATS *structure. For each rule* $r \in \Gamma_{\mathtt{init}} \cup \Gamma_{\mathtt{glob}} \cup \Gamma_{\mathtt{next}}$*, each* $s \in S$ *and all* $\mathtt{e} \in \mathsf{AtExpr}_{\mathrm{GDL}}$*, we have*

$$G, s \models_{\mathrm{GDL}} \mathtt{e} \text{ iff } \mathcal{A}_G, s \models_{\mathsf{ATL}} \mathcal{R}(\mathtt{e}) \text{ and } G \models_{\mathrm{GDL}} r \text{ iff } \mathcal{A}_G \models_{\mathsf{ATL}} \mathcal{R}(r)$$

Proof. Let a game description Γ, its game model $G = \langle S, s_0, Ag, Ac_1, \cdots, Ac_n, \tau, \pi\rangle$, and its associated AATS, $\mathcal{A}_G = \langle Q, s_0, Ag, Ac_1 \cup \{fin_1\}, \ldots, Ac_n \cup \{fin_n\}, \rho, \tau', \Phi, \pi'\rangle$ be given.

Step 1: to show $G, s \models_{\mathrm{GDL}} \mathtt{e}$ iff $\mathcal{A}_G, s \models_{\mathsf{ATL}} \mathcal{R}(\mathtt{e})$, we just need to show it holds for four different cases, namely $\mathtt{e} =$ `p` , `true p` , `not p` , or `not (true p)`. Here we only show the last case $\mathtt{e} =$ `not (true p)`, as other cases are very similar.

By GDL semantics (Definition 3.9), $G, s \models_{\mathrm{GDL}}$ `not (true p)` iff $G, s \not\models_{\mathrm{GDL}}$ `p`. And by the semantic translation $\mathcal{T}_{sem}$, $G, s \not\models_{\mathrm{GDL}}$ `p` iff $\mathcal{A}_G, s \not\models_{\mathsf{ATL}} p$. Then by $\mathcal{R}$, $\mathcal{A}_G, s \not\models_{\mathsf{ATL}} p$ iff $\mathcal{A}_G, s \models_{\mathsf{ATL}} \mathcal{R}($`not (true p)`$)$.

Step 2: to show $G \models_{\mathrm{GDL}} r$ iff $\mathcal{A}_G \models_{\mathsf{ATL}} \mathcal{R}(r)$, we show it is the case for three type of rules in $\Gamma_{\mathtt{init}} \cup \Gamma_{\mathtt{glob}} \cup \Gamma_{\mathtt{next}}$.

- Case: $r = (\Leftarrow ($`init p`$))$.

  We have $G \models_{\mathrm{GDL}} (\Leftarrow ($`init p`$))$ iff $G, s_0 \models_{\mathrm{GDL}}$ `p` iff $\mathcal{A}_G, s_0 \models_{\mathsf{ATL}} p$. Since *init* is only true in s_0 of $\mathcal{A}_G$, we have $\mathcal{A}_G, s_0 \models_{\mathsf{ATL}} p$ iff $\forall s \in Q(\mathcal{A}_G, s \models_{\mathsf{ATL}} init \to p)$ iff $\mathcal{A}_G \models_{\mathsf{ATL}} init \to p$ iff $\mathcal{A}_G \models_{\mathsf{ATL}} \mathcal{R}(r)$.

- Case: $r = (\Leftarrow (\mathtt{p})(\mathtt{e}_1) \ldots (\mathtt{e}_m))$.

  We have,
  $G \models_{\mathrm{GDL}} r$
  iff by GDL Semantics
  $\forall s \in S(\forall i \in [1..m](G, s \models_{\mathrm{GDL}} \mathtt{e}_i) \Rightarrow G, s \models_{\mathrm{GDL}} \mathtt{p})$
  iff by Step 1
  $\forall s \in S(\forall i \in [1..m](\mathcal{A}_G, s \models_{\mathsf{ATL}} \mathcal{R}(\mathtt{e}_i)) \Rightarrow \mathcal{A}_G, s \models_{\mathsf{ATL}} p)$
  iff by $Q = S \cup \{z\}$ and the fact that $z_\perp$ is only true in z

$\forall s \in Q(\mathcal{A}_G, s \models_{\mathsf{ATL}} \mathcal{R}(\mathtt{e}_1) \wedge \cdots \wedge \mathcal{R}(\mathtt{e}_m) \wedge \neg z_\perp) \Rightarrow \mathcal{A}_G, s \models_{\mathsf{ATL}} p$
iff
$\forall s \in Q(\mathcal{A}_G, s \models_{\mathsf{ATL}} \mathcal{R}(\mathtt{e}_1) \wedge \cdots \wedge \mathcal{R}(\mathtt{e}_m) \wedge \neg z_\perp \rightarrow p)$
iff
$\mathcal{A}_G \models_{\mathsf{ATL}} \langle\langle\rangle\rangle\Box(\mathcal{R}(\mathtt{e}_1) \wedge \cdots \wedge \mathcal{R}(\mathtt{e}_m) \wedge \neg z_\perp \rightarrow p)$
iff
$\mathcal{A}_G \models_{\mathsf{ATL}} \mathcal{R}(r)$.

- Case: $r = (\Leftarrow (\mathtt{next\ p})(\mathtt{e}_1) \ldots (\mathtt{e}_m)(\mathtt{does\ i_1\ a_1}) \ldots (\mathtt{does\ i_k\ a_k}))$.

We have,
$G \models_{\mathrm{GDL}} r$
iff by GDL Semantics
$\forall s, t \in S(\forall i \in [1..m](G, s \models_{\mathrm{GDL}} \mathtt{e}_i)$ and t is an $i_1 : a_1, \ldots i_k : a_k$ successor of $s \Rightarrow G, t \models_{\mathrm{GDL}} \mathtt{p})$
iff by Step 1
$\forall s, t \in S(\forall i \in [1..m](\mathcal{A}_G, s \models_{\mathsf{ATL}} \mathcal{R}(\mathtt{e}_i))$ and t is an $i_1 : a_1, \ldots i_k : a_k$ successor of $s \Rightarrow \mathcal{A}_G, t \models_{\mathsf{ATL}} \mathcal{R}(\mathtt{p}))$
iff by the construction procedure of $\mathcal{A}_G$
$\forall s, t \in S(\forall i \in [1..m](\mathcal{A}_G, s \models_{\mathsf{ATL}} \mathcal{R}(\mathtt{e}_i))$ and t is an $i_1 : a_1, \ldots i_k : a_k$ successor of $s \Rightarrow \mathcal{A}_G, t \models_{\mathsf{ATL}} t(\mathtt{p}) \wedge done(i_1, a_1) \wedge \cdots \wedge done(i_k, a_k))$.
iff
$\forall s \in S(\forall i \in [1..m](\mathcal{A}_G, s \models_{\mathsf{ATL}} \mathcal{R}(\mathtt{e}_i)) \Rightarrow \mathcal{A}_G, s \models_{\mathsf{ATL}} \langle\langle\{i_1, \ldots i_k\}\rangle\rangle \bigcirc (t(\mathtt{p}) \wedge done(i_1, a_1) \wedge \cdots \wedge done(i_k, a_k)))$
iff
$\forall s \in S(\mathcal{A}_G, s \models_{\mathsf{ATL}} \mathcal{R}(\mathtt{e}_1) \wedge \cdots \wedge \mathcal{R}(\mathtt{e}_m)) \rightarrow \langle\langle\{i_1, \ldots i_k\}\rangle\rangle \bigcirc (t(\mathtt{p}) \wedge done(i_1, a_1) \wedge \cdots \wedge done(i_k, a_k))$
iff by $Q = S \cup \{z\}$ and the fact that $z_\perp$ is only true in z
$\forall s \in Q(\mathcal{A}_G, s \models_{\mathsf{ATL}} \neg z_\perp \wedge \mathcal{R}(\mathtt{e}_1) \wedge \cdots \wedge \mathcal{R}(\mathtt{e}_m)) \rightarrow \langle\langle\{i_1, \ldots i_k\}\rangle\rangle \bigcirc (t(\mathtt{p}) \wedge done(i_1, a_1) \wedge \cdots \wedge done(i_k, a_k))$
iff
$\mathcal{A}_G \models_{\mathsf{ATL}} \langle\langle\rangle\rangle\Box(\neg z_\perp \wedge$
$\mathcal{R}(\mathtt{e}_1) \wedge \cdots \wedge \mathcal{R}(\mathtt{e}_m) \rightarrow \langle\langle\{i_1, \ldots i_k\}\rangle\rangle \bigcirc (t(\mathtt{p}) \wedge done(i_1, a_1) \wedge \cdots \wedge done(i_k, a_k)))$
iff
$\mathcal{A}_G \models_{\mathsf{ATL}} \mathcal{R}(r)$.

□

3.3.2 $\mathcal{T}_{syn}$: From GDL Descriptions to ATL Theories

Now we turn to the syntactic level of the correspondence. Given a GDL description Γ, we translate it into an ATL-theory Γ_{ATL} which characterises the same game. Here by 'same game' we mean this. From the description Γ, we can derive a game model $GMod(\Gamma)$, and hence a unique AATS $\mathcal{A}_{GMod(\Gamma)}$. And for Γ_{ATL}, there might be several AATSs that satisfy it. They all amount to the same in the sense that there is no ATL formulas that can distinguish these AATSs and $\mathcal{A}_{GMod(\Gamma)}$. We will prove this formally later by showing that there is an alternating bisimulation between them.

Given Γ, we define the ATL theory Γ_{ATL} as a conjunction of ATL formulas:

$$\Gamma_{\mathsf{ATL}} = \mathbf{INIT} \wedge \mathbf{MEM} \wedge \mathbf{ONE\_DONE} \wedge \mathbf{LEGAL} \wedge \mathbf{STRAT} \wedge \mathbf{TERM}$$

First **INIT** is to characterise the initial state. Next, **MEM** is to remember the previous state; **ONE_DONE** and **LEGAL** are to make sure that for each non-terminal state, there is a legal action selected by each agent. Combined with **MEM**, **ONE_DONE** and **LEGAL**, **STRAT** is to compute the current state given the old state and the actions have been made. Finally, **TERM** ensures all terminal states will go to the the special state z.

A state corresponds to a set of atoms which are true in that state. These formulas force atoms to satisfy the wanted constraints among the states. Here by 'force', we mean the following. Suppose we want to make an atom true, say p, in the initial state q_0, and false in all subsequent states i.e. in $Q \setminus \{q_0\}$. This can be expressed as a constraint among states $CONS(p) ::= q_0 \in \pi(p) \wedge \forall q_i \in Q(q_i \neq q_0 \rightarrow q_i \notin \pi(p))$. We can use the formula $\varphi(p) ::= p \wedge \langle\langle\rangle\rangle \bigcirc \langle\langle\rangle\rangle \Box \neg p$ to force a class of AATSs to satisfy the constraint $CON(p)$. More precisely, it is the collection of AATSs $\mathcal{A}$ that satisfy the following equivalence:

$$\mathcal{A}, q_0 \models_{\mathsf{ATL}} \varphi(p) \text{ iff } CONS(p) \text{ holds } for \mathcal{A}.$$

We now explain Γ_{ATL} with more details. Let $S_0 = DPMod(\delta(\Gamma_{\texttt{init}}) \cup \delta(\Gamma_{\texttt{glob}}))$, which gives the minimal set of atomic consequences (using the global rules) of all (`init p`) expressions. We want an ATL formula that characterises the full initial state. Consider:

$$\mathbf{INIT} = init \wedge \langle\langle\rangle\rangle \bigcirc \langle\langle\rangle\rangle \Box \neg init \wedge P_{S_0} \wedge \bigwedge_{p_{old} \in \mathsf{At}_{\mathsf{ATL}}} \neg p_{old} \wedge N_{done} \wedge \neg z_{\perp}$$

where

$$P_{S_0} = \bigwedge_{\mathtt{p} \in S_0} p \wedge \bigwedge_{\mathtt{p} \notin S_0} \neg p,$$

and

$$N_{done} = \bigwedge_{i \in Ag} \bigwedge_{a \in Ac_i \cup \{fin_i\}} \neg done(i, a).$$

This ensures that the special atom *init* is true in the initial state, and is false everywhere else, and that the truth values of the other atoms in the initial state of $GMod(\Gamma)$ are reflected properly. It also ensures that all the *old*− and *done*− propositions are false, since there is no previous state, and this is not a z state.

The intended use of an atom p_{old} is that it records the old, i.e. previous, truth-value of p. This is captured by the principle **MEM**:

$$\mathbf{MEM} = \langle\langle\rangle\rangle\Box \bigwedge_{\mathtt{p} \in \mathsf{At}_{\mathrm{GDL}}} ((t(\mathtt{p}) \wedge \neg terminal \rightarrow \langle\langle\rangle\rangle \bigcirc t(\mathtt{p})_{old}) \wedge$$
$$(\neg t(\mathtt{p}) \wedge \neg terminal \rightarrow \langle\langle\rangle\rangle \bigcirc \neg t(\mathtt{p})_{old}))$$

The following constraint makes sure that for all non-initial states, one action is done by each agent:

$$\mathbf{ONE\_DONE} = \langle\langle\rangle\rangle\Box(\neg init \rightarrow \bigwedge_{i \in Ag} XOR_{a \in Ac_i \cup \{fin_i\}} done(i, a))$$

where *XOR* is the *exclusive OR* operator, a Boolean operator that returns a value of TRUE only if just one of its operands is TRUE.

One assumption for playing GDL games is that each agent must play legal moves. This is captured by the following principal:

$$\mathbf{LEGAL} = \langle\langle\rangle\rangle\Box \bigwedge_{i \in Ag, a_i \in Ac_i} \bigwedge_{X = \{i\}, Ag} (legal(i, a_i) \wedge \neg terminal \leftrightarrow \langle\langle X \rangle\rangle \bigcirc done(i, a_i))$$

This principle says that, when an action a_i is legal for agent i, and the current state is not a terminal state, then agent i should have a strategy to enforce it, and vice versa.

Let $bd_1, bd_2, \ldots$ be variables over possible bodies of rules, that is, sets of literals, but not including any (`does i a`). We assume that atoms like (`does i a`) only occur in rules which have a head of the form `next p`. This is their intended use:

to enable players to compute the next state, given the moves of all players. Let $\mathtt{p} \in \mathsf{At}_{\mathrm{GDL}}$. Now suppose that *all* the rules r in Γ with $hd(r) \in \{(\mathtt{p}), (\mathtt{next\ p})\}$ are the following:

$$
\begin{array}{lcllll}
r_1: & \Leftarrow & (\mathtt{p}) & bd_1 & & \\
\ldots & \Leftarrow & (\mathtt{p}) & \ldots & & \\
r_h: & \Leftarrow & (\mathtt{p}) & bd_h & & \\
s_1: & \Leftarrow & (\mathtt{next\ p}) & bd'_1 & & (\mathtt{does\ i}_{1_1}\ \mathtt{a}_{1_1}) \ldots (\mathtt{does\ i}_{m_1}\ \mathtt{a}_{m_1}) \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
s_k: & \Leftarrow & (\mathtt{next\ p}) & bd'_k & & (\mathtt{does\ i}_{k_1}\ \mathtt{a}_{k_1}) \ldots (\mathtt{does\ i}_{m_k}\ \mathtt{a}_{m_k})
\end{array}
$$

We map all these rules for $\mathtt{p}$ to an ATL formula $\varphi(\mathtt{p})$. For this, we first translate the symbols from GDL to those of ATL using the functions t and t_{old} defined in Definition 3.10. For convenience, we denote $t(bd_i)$ as the translation of all the expressions in bd_i by t, and similar for $t_{old}(bd_j)$. For each atom $\mathtt{p} \in \mathsf{At}_{\mathrm{GDL}}$, we can now define an ATL constraint $MIN(\mathtt{p})$, as follows:

$$
MIN(\mathtt{p}) = t(\mathtt{p}) \leftrightarrow \Big(\bigvee_{i \leq h} t(bd_i) \vee \bigvee_{j \leq k} (t_{old}(bd'_j) \wedge done(i_{j_1}, a_{j_1}) \wedge \cdots \wedge done(i_{j_m}, a_{j_m}))\Big)
$$

And if $\mathtt{p}$ does not occur in a head of any rule in Γ, we define $MIN(\mathtt{p}) = \neg p$.

The semantics of stratified program Γ is now captured by the following constraint:

$$
\mathbf{STRAT} = \langle\langle\rangle\rangle\Box \bigwedge_{\mathtt{p} \in \mathsf{At}_{\mathrm{GDL}}} (\neg init \wedge \neg z_\perp \rightarrow MIN(\mathtt{p}))
$$

When a terminal state is reached, no further 'real' moves are played by agents, i.e. they always play the fin_i actions:

$$
\mathbf{TERM} = \langle\langle\rangle\rangle\Box \bigwedge_{i \in Ag} (terminal \vee z_\perp) \leftrightarrow \langle\langle\rangle\rangle \bigcirc (z_\perp \wedge done(i, fin_i)))
$$

In section 3.3.1, we have shown that we can conceive a GDL game model as an AATS. The following is essentially a soundness result for our transformation. Let Γ be a game description, and $G = GMod(\Gamma)$ be its game model with initial state s_0; $\mathcal{A}_G$ is the corresponding AATS. We have:

$$
\mathcal{A}_G, s_0 \models_{\mathsf{ATL}} \Gamma_{\mathsf{ATL}}
$$

In the following, we add a requirement resulting in *uniform* AATS structures:

$$(uni) \quad \forall s \in Q \forall C \subseteq Ag \forall \sigma_C \forall s', s'' \in out(\sigma_C, s) \forall i \in C \forall a \in Ac_i : \\ (done(i,a) \in \pi(s') \Leftrightarrow done(i,a) \in \pi(s''))$$

This requirement says that, in the outcome states of a coalition C executing a strategy, for the agents in C, the related *done* propositions are uniformly true or false. Notice that $\mathcal{A}_{GMod(\Gamma)}$ satisfies this requirement.

Lemma 3.1. *Satisfiability checking with respect to a uniform* AATS *is in ExpTime.*

Proof. It is shown that checking the satisfiability/validity of ATL formulas with respect to AATS is in EXPTIME [97]. The uniform AATS is a subclass of AATS, and we can apply similar method to show that satisfiability/validity of ATL formulas with respect to uniform AATS is also in EXPTIME. □

Note that the translation of the GDL description Γ_{GDL} into the ATL specification Γ_{ATL} can be done in polynomial time.

Now we prove an important result: *every model for* Γ_{ATL} *is alternating-bisimilar to* $\mathcal{A}_{GMod(\Gamma)}$.

Theorem 3.4. *Let* $G = GMod(\Gamma)$ *be the model for a game description* Γ*, and let* $\mathcal{A}_1 = \langle Q_1, q_1, Ag, \{Ac_i | i \in Ag\}, \rho_1, \tau_1, \Phi, \pi_1 \rangle$ *be its associated* AATS *structure. Let* $\mathcal{A}_2 = \langle Q_2, q_2, Ag, \{Ac_i | i \in Ag\}, \rho_2, \tau_2, \Phi, \pi_2 \rangle$ *be an uniform* AATS *that satisfies* Γ_{ATL}*. There exists an alternating bisimulation* $\mathcal{R}$ *between* $\mathcal{A}_1$ *and* $\mathcal{A}_2$*, with* $\mathcal{R} q_1 q_2$.

Proof. We define a relation $\mathcal{R} \subseteq Q_1 \times Q_2$ as follows,

$$\mathcal{R} z_1 z_2 \text{ iff } \pi_1(z_1) = \pi_2(z_2).$$

We show that $\mathcal{R}$ is an alternating bisimulation which connects q_1 and q_2.

By **INIT**, one could easily check that $\mathcal{R} q_1 q_2$.

Suppose we have established $\mathcal{R} z_1 z_2$ for some $z_1 \in Q_1$ and $z_2 \in Q_2$ (see Figure 3.3). It is easy to see that $\mathcal{R}$ satisfies the invariance condition in definition 2.12. We need to show that it satisfies both the **Zig** and **Zag** conditions in definition 2.12 as well.

We first show the **Zig** condition. In the case that $\mathcal{A}_1, z_1 \models_{\mathsf{ATL}}$ *terminal*, the only actions available to the agents are the fin_i actions, and this will leads to

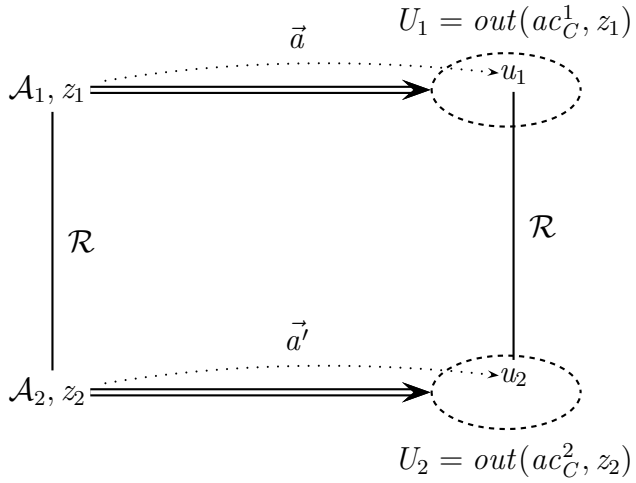


Figure 3.3: Alternating bisimulation between $\mathcal{A}_1$ and $\mathcal{A}_2$

the unique z state. The condition is then easy to complete. In the following, we suppose $\mathcal{A}_1, z_1 \not\models_{\mathsf{ATL}} \mathit{terminal}$. By assumption, we then also have $\mathcal{A}_2, z_2 \not\models_{\mathsf{ATL}} \mathit{terminal}$. Take an arbitrary coalition C, with a joint action $ac_C^1 \in \mathit{options}(C, z_1)$, and consider $U_1 = \mathit{out}(ac_C^1, z_1) \subseteq Q_1$ in $\mathcal{A}_1$. We need to find $ac_C^2 \in \mathit{options}(C, z_2)$ such that for every $u_2 \in \mathit{out}(ac_C^2, z_2)$, there is a $u_1 \in U_1$ so that $\mathcal{R}u_1u_2$.

It follows from $ac_C^1 \in \mathit{options}(C, z_1)$ that $\mathcal{A}_1, z_1 \models_{\mathsf{ATL}} \mathit{legal}(i, a_i^1)$ for all $i \in C, a_i^1 \in ac_C^1$. Therefore, $\mathcal{A}_2, z_2 \models_{\mathsf{ATL}} \mathit{legal}(i, a_i^1)$ for all $i \in C, a_i^1 \in ac_C^1$. And by **LEGAL**, we have $\mathcal{A}_2, z_2 \models_{\mathsf{ATL}} \langle\langle i \rangle\rangle \bigcirc \mathit{done}(i, a_i^1)$ for all $i \in C$. So, for each $i \in C$, there is $ac_i^2 \in \mathit{options}(i, z_2)$ such that for all $x \in \mathit{out}(ac_i^2, z_2) \subseteq Q_2$, $\mathcal{A}_2, x \models_{\mathsf{ATL}} \mathit{done}(i, a_i^1)$. Let ac_C^2 be an action profile that consists of a_i^2 for all $i \in C$ and $U_2 = \mathit{out}(ac_C^2, z_2) \subseteq Q_2$. It is easy to see that for all $x \in U_2$, we have $\mathcal{A}_2, x \models_{\mathsf{ATL}} \mathit{done}(i, a_i^1)$ for $i \in C$. We pick an arbitrary $u_2 \in U_2$. We are done if we can show that there is a $u_1 \in U_1$ for which $\mathcal{R}u_1u_2$.

By **ONE_DONE**, there is one and only one $\mathit{done}(i, a)$ true in u_2 for each $i \in Ag$. We already know $\mathcal{A}_2, u_2 \models_{\mathsf{ATL}} \mathit{done}(i, a_i^1)$ for $i \in C$, and we assume $\mathcal{A}_2, u_2 \models_{\mathsf{ATL}} \mathit{done}(j, b_j^1)$ for all $j \in Ag \setminus C$. As u_2 is a successor of z_2, we have $\mathcal{A}_2, z_2 \models_{\mathsf{ATL}} \langle\langle Ag \rangle\rangle \bigcirc \mathit{done}(j, b_j^1)$ for all $j \in Ag \setminus C$, and by **LEGAL**, we have $\mathcal{A}_2, z_2 \models_{\mathsf{ATL}} \mathit{legal}(j, b_j^1)$ for all $j \in Ag \setminus C$, hence $\mathcal{A}_1, z_1 \models_{\mathsf{ATL}} \mathit{legal}(j, b_j^1)$ for all $j \in Ag \setminus C$. We collect the actions a_i^1 for $i \in C$, and b_j^1 for $j \in Ag \setminus C$ to make a complete action profile $\vec{a}$.

Now go back to $\mathcal{A}_1$ and consider $u_1 = out(\vec{a}, z_1)$. We claim that this u_1 is the state we are looking for: it satisfies $\mathcal{A}_1, u_1 \models_{\mathsf{ATL}} p$ iff $\mathcal{A}_2, u_2 \models_{\mathsf{ATL}} p$, for all $p \in \Phi$. By **MEM**, we have $p_{old} \in \pi_1(u_1)$ iff $p_{old} \in \pi_2(u_2)$ for all $p_{old} \in \Phi$. By **ONE_DONE**, we have $done(i, a_i) \in \pi_1(u_1)$ iff $done(i, a_i) \in \pi_2(u_2)$ for all $done(i, a_i) \in \Phi$.

We now claim:

$$\forall \mathtt{p} \in \mathsf{At}_{\mathrm{GDL}}, G, u_1 \models_{\mathrm{GDL}} \mathtt{p} \text{ iff } \mathcal{A}_1, u_1 \models_{\mathsf{ATL}} t(\mathtt{p}) \text{ iff } \mathcal{A}_2, u_2 \models_{\mathsf{ATL}} t(\mathtt{p}) \tag{3.1}$$

The first 'iff' immediately follows from Theorem 3.3, and we will use it to know 'why' a certain atom is true in G, u_1. Since $u_1 = out(\vec{a}, z_1)$, we know that in G, we have $u_1 = \tau(\vec{a}, z_1)$, i.e. u_1 is calculated from Γ as $DPMod(F_\Gamma(\vec{a}, z_1) \cup \delta(\Gamma_{\mathtt{glob}}))$.

We distinguish two cases:

- Either there is no rule $r \in F_\Gamma(\vec{a}, z_1) \cup \delta(\Gamma_{\mathtt{glob}})$ with $hd(r) = \mathtt{p}$. Then $\mathtt{p} \notin DPMod(F_\Gamma(\vec{a}, z_1) \cup \delta(\Gamma_{\mathtt{glob}}))$ and hence $G, u_1 \not\models_{\mathrm{GDL}} \mathtt{p}$, and, by Theorem 3.3, $\mathcal{A}_1, u_1 \models_{\mathsf{ATL}} \neg p$. Now consider the axiom **STRAT**, which says that $MIN(\mathtt{p})$ is true everywhere in $\mathcal{A}_2$ except the initial state and the zink state. In case that $\mathtt{p}$ does not appear in the head of any rule in Γ, $MIN(\mathtt{p}) = \neg p$, which implies that $\mathcal{A}_2, u_2 \models_{\mathsf{ATL}} \neg p$, as desired. Otherwise, $\mathtt{p}$ must appear in the head of some rule $r \in \Gamma$. Since in this case we assume this is not so for $\delta(\Gamma_{\mathtt{glob}})$, the only way to make p true, using $MIN(\mathtt{p})$, is that we have some bd'_j, generated by some rule

  $$s_j : (\Leftarrow \mathtt{next(p)}\ bd'_j\ \mathtt{does(j_1, a_{j_1})} \cdots \mathtt{does(j_m, a_{j_m})})$$

  for which $\mathcal{A}_2, u_2 \models_{\mathsf{ATL}} t_{old}(bd'_j) \wedge done(j_1, a_{j_1}) \wedge \ldots done(j_m, a_{j_m})$. And by **ONE_DONE**, we know that for any $i \in Ag$, the only action b_i for which $\mathcal{A}_2, u_2 \models_{\mathsf{ATL}} done(i, b_i)$ is true is $b_i = a_i$. By **MEM**, since $\mathcal{A}_2, u_2 \models_{\mathsf{ATL}} t_{old}(bd'_j)$, we have that $\mathcal{A}_2, z_2 \models_{\mathsf{ATL}} bd'_j$. Using the induction hypothesis we get $\mathcal{A}_1, z_1 \models_{\mathsf{ATL}} bd'_j$. Now looking at $\mathcal{A}_1$ as a game model G for Γ, we see that $(\Leftarrow\ \mathtt{p}) \in F_\Gamma(\vec{a}, z_1)$, contradicting our assumption that there is no rule in $F_\Gamma(\vec{a}, z_1) \cup \delta(\Gamma_{\mathtt{glob}})$ with $\mathtt{p}$ as a head.

- Or, for some rule $r \in F_\Gamma(\vec{a}, z_1) \cup \delta(\Gamma_{\mathtt{glob}})$, we have $hd(r) = \mathtt{p}$. We distinguish two sub-cases:

  - $r \in F_\Gamma(\vec{a}, z_1)$. It follows that $r = (\Leftarrow \mathtt{p})$. Since $u_1 = DPMod(F_\Gamma(\vec{a}, z_1) \cup$

$\delta(\Gamma_{\mathtt{glob}})$), we have $G, u_1 \models_{\mathsf{ATL}} \mathtt{p}$. It also follows from $r \in F_\Gamma(\vec{a}, z_1)$, that $G, z_1 \cup \{\mathtt{does\ 1\ a_1} \wedge \cdots \wedge \mathtt{does\ n\ a_n}\} \models_{\mathsf{ATL}} bdl(r') \wedge bda(r')$ for some `next` rule r' in the form of $(\Leftarrow \mathtt{next(p)}\ bdl\ bda)$, where bdl is the literal part of this rule, and bda is the action part. This means that $a_x \in \vec{a}$ for all $\mathtt{does\ x\ a_x} \in bda$. By construction of G, we have $G, z_1 \models_{\mathrm{GDL}} bdl(r')$, and, by Theorem 3.3, we have $\mathcal{A}_1, z_1 \models_{\mathsf{ATL}} t(bdl(r'))$ which gives, by the induction hypothesis, $\mathcal{A}_1, z_2 \models_{\mathsf{ATL}} t(bdl(r'))$ and, by **MEM**, $\mathcal{A}_2, u_2 \models_{\mathsf{ATL}} t_{old}(bdl(r'))$. By choice of u_2, we also have $\mathcal{A}_2, u_2 \models_{\mathsf{ATL}} done(1, a_1) \wedge \cdots \wedge done(n, a_n)$, thus $\mathcal{A}_2, u_2 \models_{\mathsf{ATL}} t_{old}(bda(r'))$. By $MIN(p)$, we then have $\mathcal{A}_2, u_2 \models_{\mathsf{ATL}} p$.

– $r \in \delta(\Gamma_{\mathtt{glob}})$ and $r \notin F_\Gamma(\vec{a}, s)$. Now we consider a level mapping $f : \mathtt{e}(\mathsf{At}_{\mathrm{GDL}}(\Gamma)) \to \mathbb{N}$. We claim for all $n \in \mathbb{N}$,

$$f(\mathtt{x}) = n \Rightarrow (G, u_1 \models_{\mathrm{GDL}} \mathtt{x} \Leftrightarrow \mathcal{A}_1, u_1 \models_{\mathsf{ATL}} x \Leftrightarrow \mathcal{A}_2, u_2 \models_{\mathsf{ATL}} x)$$

We do induction on $f(\mathtt{p})$.

* *Base case:* $f(\mathtt{p}) = 0$. There must be a global rule $(\Leftarrow \mathtt{p})$ in $\delta(\Gamma_{\mathtt{glob}})$, thus $G, u_1 \models_{\mathrm{GDL}} \mathtt{p}$ and $\mathcal{A}_1, u_1 \models_{\mathsf{ATL}} p$. And by $MIN(\mathtt{p})$, we have $\mathcal{A}_2, u_2 \models_{\mathsf{ATL}} p$. This proves the claim.
* *Induction step:* Suppose $f(\mathtt{p}) = k + 1$, and the claim proven for all $\mathtt{q}$ with $f(\mathtt{q}) \leq k$. We have to show that $G, u_1 \models_{\mathsf{ATL}} \mathtt{p} \Leftrightarrow \mathcal{A}_1, u_1 \models_{\mathsf{ATL}} p \Leftrightarrow \mathcal{A}_2, u_2 \models_{\mathsf{ATL}} p$. The fact $G, u_1 \models_{\mathrm{GDL}} \mathtt{p}$ is true if and only if there exists a rule $r = (\Leftarrow \mathtt{p}\ bd) \in \delta(\Gamma_{\mathtt{glob}})$ such that $G, u_1 \models_{\mathrm{GDL}} bd$. For any atom $\mathtt{q} \in bd$, $f(q) < k + 1$, so by induction hypothesis, we know that $G, u_1 \models_{\mathsf{ATL}} \mathtt{q} \Leftrightarrow \mathcal{A}_1, u_1 \models_{\mathsf{ATL}} q \Leftrightarrow \mathcal{A}_2, u_2 \models_{\mathsf{ATL}} q$, for all $\mathtt{q} \in bd$. It follows that $G, u_1 \models_{\mathsf{ATL}} bd \Leftrightarrow \mathcal{A}_1, u_1 \models_{\mathsf{ATL}} t(bd) \Leftrightarrow \mathcal{A}_2, u_2 \models_{\mathsf{ATL}} t(bd)$. And by $MIN(\mathtt{p})$, we have $G, u_1 \models_{\mathrm{GDL}} \mathtt{p}$ if and only if $\mathcal{A}_2, u_2 \models_{\mathsf{ATL}} p$.

We now show the **Zag** condition.

Take an arbitrary coalition C and with a joint action ac^2_C and consider $U_2 = out(ac^2_C, z_2) \subseteq Q_2$ in $\mathcal{A}_2$. Pick an arbitrary $u_2 \in U_2$, we apply **ONE_DONE**, so for $i \in C$, we have an unique $done(i, a^1_i)$ true in u_2. Due to the *uniform requirement*, we have that for all $u \in U_2$ and all $i \in C$, $\mathcal{A}_2, u \models_{\mathsf{ATL}} done(i, a^1_i)$. Take a^1_i into ac^1_C, we have an action profile for C. And let $U_1 = out(ac^1_C, z_1)$.

We want to demonstrate that for every $u_1 \in U_1$ there is a $u_2 \in U_2$ for which

$\mathcal{R}u_1u_2$. Choose $u_1 \in U_1$ arbitrarily. Let $\vec{a}$ be the action profile for which $u_1 = out(\vec{a}, z_1)$, it is easy to see that in G this means that $u_1 = \tau(\vec{a}, z_1)$, i.e., $u_1 = DPMod(F_\Gamma(\vec{a}, z_1) \cup \delta(\Gamma_{\mathtt{glob}}))$. Now we have $\mathcal{A}_1, z_1 \models_{\mathsf{ATL}} legal(j, a_j^1)$ for $j \in Ag \setminus C$ and $a_j^1 \in \vec{a}$. And by assumption, $\mathcal{A}_2, z_2 \models_{\mathsf{ATL}} legal(j, a_j^1)$ as well. Hence, by **LEGAL**, we have $\mathcal{A}_2, z_2 \models_{\mathsf{ATL}} \langle\langle j \rangle\rangle \bigcirc done(j, a_j^1)$ for every $j \in Ag \setminus C$. For each $done(j, a_j^1)$, we can find an action a_j^2 in $\mathcal{A}_2$ such that for all $u \in out(a_j^2, z_2)$, we have $\mathcal{A}_2, u \models_{\mathsf{ATL}} done(j, a_j^1)$. We collect a_j^2 for all $j \in Ag \setminus C$, and combine them with a_i^2 for all $i \in Ag$, then we get a complete action profile $\vec{a'}$.

Now let $u_2 = out(\vec{a}', z_2)$, and we claim that this is the one to complete $\mathcal{R}u_1u_2$. The proof that $\forall \mathsf{p} \in \mathsf{At}_{\mathrm{GDL}}, G, z_1 \models_{\mathrm{GDL}} \mathsf{p}$ iff $\mathcal{A}_1, u_1 \models_{\mathsf{ATL}} p$ iff $\mathcal{A}_2, u_2 \models_{\mathsf{ATL}} p$ is very similar to the proof of (3.1) above. □

One way of interpreting the result above is as follows: GDL can be viewed as a *model specification language*, suitable for use in a *model checker* [16]. This gives rise to the formal decision problem of ATL *model checking problem over* GDL *game descriptions*, which can be described as follows: *Given an* ATL *formula* φ *and a* GDL *game description* Γ*, is it the case that* $\mathcal{A}_{GMod(\Gamma)} \models_{\mathsf{ATL}} \varphi$*?*

Theorem 3.5. ATL *model checking over propositional* GDL *game descriptions is* EXPTIME-Complete.

Proof. Membership in EXPTIME follows from Theorem 3.4 and Lemma 3.1. Given game description Γ, and ATL formula φ, construct Γ_{ATL}, and then check whether $\Gamma_{\mathsf{ATL}} \wedge \neg\varphi$ is not satisfiable; the correctness of this procedure follows from Theorem 3.4. The fact that ATL unsatisfiability is in EXPTIME is from [19, 97].

EXPTIME-hardness may be proved by reduction from the problem of determining whether a given player has a winning strategy in the two-player game PEEK-G_4 [71, p.158].

An instance of PEEK-G_4 is a quad:

$$\langle X_1, X_2, X_3, \varphi \rangle$$

where:

- X_1 and X_2 are disjoint, finite sets of Boolean variables, with the intended interpretation that the variables in X_1 are under the control of agent 1, and X_2 are under the control of agent 2;

- $X_3 \subseteq (X_1 \cup X_2)$ are the variables deemed to be true in the initial state of the game; and
- φ is a propositional logic formula over the variables $X_1 \cup X_2$, representing the winning condition.

The game is played in a series of rounds, with the agents $i \in \{1, 2\}$ alternating (with agent 1 moving first) to select a value (true or false) for one of their variables in X_i, with the game starting from the initial assignment of truth values defined by X_3. Variables that were not changed retain the same truth value in the subsequent round. An agent wins in a given round if it makes a move such that the resulting truth assignment defined by that round makes the winning formula φ true. The decision problem associated with PEEK-G_4 involves determining whether agent 2 has a winning strategy in a given game instance $\langle X_1, X_2, X_3, \varphi \rangle$. Notice that PEEK-$G_4$ only requires "memoryless" (Markovian) strategies: whether or not an agent i can win depends only on the current truth assignment, the distribution of variables, the winning formula, and whose turn it is currently. As a corollary, if agent i can force a win, then it can force a win in $O(2^{|X_1 \cup X_2|})$ moves. Given an instance $\langle X_1, X_2, X_3, \varphi \rangle$ of PEEK-G_4.

Encoding PEEK-G_4 in GDL is a straightforward exercise in GDL programming, and the question of whether there exists a winning strategy is directly encoded in an ATL formula to model check (see [80]). □

Note that, although this seems a negative result, it means that interpreting ATL over propositional GDL descriptions is no more complex than interpreting ATL over apparently simpler model specification languages such as the Simple Reactive Systems Language [80].

3.4 Summary

There is much interest in the connections between logic and games, and in particular in the use of strategic logics for reasoning about multi-agent systems. This chapter investigated the connections between ATL and GDL, a declarative language intended for defining games. We first demonstrated that GDL can be understood as a specification language for ATL models, and subsequently that it is possible to succinctly characterise GDL game descriptions directly as ATL formulas, and that, as a corollary, the problem of interpreting ATL formulas over

GDL descriptions is EXPTIME-Complete. This chapter provides a theoretical foundation for the next chapter. We will show that our work can be applied to formal verification of GDL descriptions: the GDL game designer can express properties of games using ATL, and then automatically check whether these properties hold of their GDL descriptions.

Chapter 4

Playability Verification for Games in GDL

4.1 Introduction

Apart from the theoretical interest in drawing a link between ATL and GDL, and characterising the complexity of interpreting ATL formulas over GDL descriptions, which was explained in Chapter 3, we now want to explore this topic more practically.

Our purpose is perhaps best explained by analogy with the literature on temporal logic for reactive systems [20]. Temporal logics, in various forms, have been used for reasoning about reactive systems for several decades, and a large literature has been established on classifying the properties of such systems via temporal formulas of various types; probably the best-known classification is that of liveness and safety properties, although many more properties have been classified [50, 51, p.298]. Our ultimate aim is, in much the same way, to use ATL to derive a similar classification of game properties and to verify them using model checking tools. Note that ATL is, of course, a temporal logic, and we might expect the classification to include liveness and safety properties and similar; but the more novel aspect of the classification, (and, crucially, the part of the classification which simply cannot be done in conventional temporal logic) is a classification of *strategic* properties of games. To be more specific, we try to answer the following questions:

- What are the conditions which characterise when a given GDL description defines a (meaningful) game? We refer to these properties as *playability*

conditions. Although some useful playability conditions have been discussed in the GDL literature [26], these conditions are in fact very basic.

- How can we check whether a game specified in GDL satisfies such playability conditions?

This chapter is organised as follows. In section 4.2, we give a systematic classification of GDL playability conditions, and show how these conditions may be characterised as ATL formulas. This classification extends the discussion and formalisation of playability conditions given in [59, 26] considerably. In section 4.3, we describe the GDL2RML translator, a prototype tool for model checking of ATL properties over GDL descriptions. In Section 4.4, we present a case study of a game specified in GDL using the GDL2RML translator and the ATL model checker MOCHA. A brief summary is given in Section 4.5.

4.2 Characterising Playability Conditions in ATL

When we design a game, there are qualities or properties that we wish the game will have. Some are quite subjective, such as 'breathtaking', 'fun', etc.; some are more objective, such as 'terminal', 'turn-based' etc. Here we will focus on the objective properties, and more specifically we would like to characterise them formally. Here by 'characterising' we mean the properties will be expressed precisely and unambiguously in ATL logical formulas. A GDL game description satisfies such a formal property if and only if the ATL game model that arises from such description satisfies this property under ATL semantics.

We begin our top-level classification of game properties by distinguishing between properties relating to the *coherence* of a game and those relating to its *strategic* structure. We assume to have a stock of state atoms $\mathsf{SAt} = \{p, q, \dots\}$ (in Tic-Tac-Toe, an example would be (`cell 1 2 x`)), old atoms $\mathsf{OAt} = \{p_{old} \mid p \in \mathsf{SAt}\}$ and *done atoms* $\mathsf{DAt} = \{done(i, a) \mid i \in Ag, a \in Ac_i\}$. Throughout this chapter, unless stated otherwise, properties that we discuss are evaluated in the beginning of the game.

4.2.1 Coherence Properties

Roughly, coherence properties simply ensure that the game has a "sensible" interpretation. To illustrate what we mean by this, we introduce a vocabulary of

atomic propositions that we use within game property formulas. These propositions play an analogous role to propositions such as $at_i(\ell)$ in the temporal axiomatisation of programs [29, p.70].

- $turn_i$ will be true in a state if it is agent i's turn to take a move in that state;
- $legal(i, a)$ will be true in a state if action (move) a is legal for agent i in that state;
- $has\_legal\_move_i$ will be true in a state s if agent i has at least one legal move in that state;
- *terminal* will be true in a state if that state is terminal, i.e. the game is over.
- win_i will be true in a state if agent i has won in that state;
- $lose_i$ will be true in a state if agent i has lost in that state;
- *draw* will be true in a state if the game is drawn in that state;

Note that the specific *interpretation* of these atomic propositions will depend on the game at hand, but they will typically be straightforward to derive. In the context of GDL, we might have $win_i = goal(i, 100)$, $lose_i = goal(i, 0)$ and $draw = \bigwedge_{i \in Ag} goal(i, 50)$.

Now that we have such a vocabulary in place, we can start to define specific properties. Perhaps the most important question is whether a game is "balanced", in that all players have in some sense an equal chance to win. As it turns out, this apparently intuitive property is surprisingly hard to define, but we will see various notions of balancedness in what follows.

From the perspective of designing a game, the general game playing competition [26] suggests the following criteria to be a necessity: it should first of all be *playable*: every player has at least one move in every non-terminal state. We represent this constraint as follows.

$$\langle\langle\rangle\rangle\Box(\neg terminal \rightarrow \bigwedge_{i \in Ag} has\_legal\_move_i) \qquad \text{(Playability)}$$

where $has\_legal\_move_i = \bigvee_{a \in Ac_i} legal(i, a)$.

The second of these relate to terminal states. In a finite extensive game, the terminal states are exactly those in which no player can perform a move. This signals a fundamental difference with ATL, where computations are by definition infinite. We can bridge this gap by letting a terminal state in a game correspond with a 'zink-state', from which transitions are possible, but only to (copies of) itself. So, our first property says that a terminal state really is terminal: once we reach a terminal state, nothing subsequently changes. For all properties only involving state atoms, we have:

$$\langle\langle\rangle\rangle\Box((terminal \wedge \varphi) \rightarrow \langle\langle\rangle\rangle\Box(terminal \wedge \varphi)) \qquad \text{(GameOver)}$$

The above property involves a scheme φ, and as such it would lend itself more naturally for the theorem proving paradigm, rather than that of model checking. However, we can deal with this as follows. Every agent i has an action *noop* at his disposal. This is helpful to define turn-based games, by the following.

$$\langle\langle\rangle\rangle\Box(turn_i \leftrightarrow \neg legal(i, noop)) \qquad \text{(Turn)}$$

Let p be a state atom in SAt. We assume that state atoms cannot be changed by the players' *noop* actions. So true state atoms still remain true and false atoms remain false if all agents do *noop* actions, i.e. do nothing. This is captured by the following property:

$$\langle\langle\rangle\rangle\Box(l \rightarrow \langle\langle\rangle\rangle \bigcirc (\bigwedge_{i \in Ag} done(i, noop) \rightarrow l)) \qquad \text{(No Change)}$$

where l is a literal.

Now, we can establish (GameOver) by imposing the following, from which (GameOver) would then follow by induction over φ:

$$\langle\langle\rangle\rangle\Box(terminal \rightarrow \langle\langle\rangle\rangle \bigcirc \langle\langle\rangle\rangle\Box(terminal \wedge \bigwedge_{i \in Ag} done(i, noop))) \qquad \text{(Ind)}$$

Next, we often have that a state is terminal if the game is either won or drawn.

$$\langle\langle\rangle\rangle\Box((draw \vee \bigvee_{i \in Ag} win_i) \rightarrow terminal)$$

Note that we may or may not have the converse implication, as we can specify more subtle results using $goal(i, x)$.

There will typically be some coherence relation between win_i, $lose_i$, and $draw$ propositions, although the exact relationship will depend on the game. For example, the following says that a draw excludes a win.

$$\langle\langle\rangle\rangle\Box(draw \rightarrow \bigwedge_{i \in Ag} \neg win_i) \qquad \text{(Draw)}$$

Finally, one might add conditions like *termination*, which says that a game will eventually end:

$$\langle\langle\rangle\rangle\Diamond terminal \qquad \text{(Termination)}$$

4.2.2 Fair Playability Conditions

All of the above conditions only take the coalition modalities with empty set of agents, i.e. of the form $\langle\langle\rangle\rangle T\varphi$. Recall that $\langle\langle C\rangle\rangle T\varphi$ means that the agents in C can choose a strategy such that no matter what the agents in $Ag \setminus C$ do, $T\varphi$ will hold. In particular, $\langle\langle\rangle\rangle T\varphi$ then means that no matter what the agents in Ag do, $T\varphi$ will hold. Thus these conditions define invariants, i.e. safety properties, over games. Such properties could thus be specified using conventional temporal logics, eg. Computational Tree Logics, and verified using conventional temporal logic model checkers. We now turn to a fundamentally different class of properties – those relating to the strategic structure of a game; as we argued above, such properties cannot be specified using conventional temporal logics, whence our interest in logics such as ATL for this purpose.

In general, the kinds of properties we might typically hope for in a game relate to "fairness"[1] – intuitively, the idea that no player has an inherent advantage in

[1]the term "fairness" is already used in a technical sense in the temporal logic/verification community, to mean something related but slightly different. Here when we talk about fairness,

the game. In fact, it turns out to be rather hard to give a useful formal meaning to the term, let alone to capture such a meaning logically. Nevertheless, there are some useful fairness-related playability conditions that we can capture.

We first define the notion of *winnability*. A game is *strongly winnable* iff:

> "for some player, there is a sequence of individual moves of that player that leads to a terminal state of the game where that player's goal value is maximal. [26, p.9].

Formally, strong winnability may be captured as follows. Notice that this can not be expressed in CTL if the number of agents in Ag is more than one.

$$\bigvee_{i \in Ag} \langle\langle i \rangle\rangle \Diamond win_i \qquad \text{(Strong Winnability)}$$

The Strong Winnability is too strong for games involving multiple players, as if it would hold in the initial state, then perfect play by that player would guarantee a win by that player, which makes the game inherently unfair. So, we have a more relaxed requirement, called *weak winnability*, for multi-player games:

> "A game is weakly winnable if and only if, for every player, there is a sequence of joint moves of all players that leads to a terminal state where that player's goal is maximal." [26, p.9].

We capture this as follows:

$$\bigwedge_{i \in Ag} \langle\langle Ag \rangle\rangle \Diamond win_i. \qquad \text{(Weak Winnability)}$$

In general game playing, every game should be weakly winnable, and all single player games are strongly winnable. This means that in any general game, *every player at least has a chance of winning.*

One might also impose "Weak Losability", which would be like the property (Weak Winnability), but with win_i replaced by $lose_i$: at least, in principle, every player could lose.

---

we are appealing to the everyday meaning of the term, rather than the technical meaning as in [22].

There are many other notions of fairness one can impose on a game. We say a game is *fair* if no player can lose without himself at least being involved. To put it another way, a player can only lose with less than perfect play.

$$\bigwedge_{i \in Ag} \neg\langle\langle Ag \setminus \{i\}\rangle\rangle \Diamond lose_i \qquad \text{(Fair)}$$

4.2.3 Characterising Different Games

The notions we just discussed can be considered as examples of minimal requirements to call a system a "meaningful game". We show how ATL can be used to characterise different kinds of games. In fact we have already seen such a general property: our (Strong Winnability) is in the literature known as *determinacy* of the game. Other examples would include (Sequential): everywhere, the next state is determined by one agent. In ATL, such a situation is called *turn-based* ([6]). Although the characteristic formula refers to arbitrary φ again, it can also be related to $\langle\langle\rangle\rangle\Box(XOR_{i \in Ag} turn_i)$, together with (Turn) and (Ind).

$$\langle\langle\rangle\rangle\Box(\langle\langle Ag\rangle\rangle \bigcirc \varphi \rightarrow \bigvee_{i \in Ag} \langle\langle i\rangle\rangle \bigcirc \varphi) \qquad \text{(Sequential)}$$

In many sequential games, the *order* in which to take turns is crucial, and although [18, page 56] claims that 'young children are obsessed with making sure that they go first in any and every game that they play', sometimes, rather than a first-mover advantage, there is a second-mover advantage (like cutting a cake and choosing a piece) or an advantage to enter the game in any specific round. Defining the advantage of agent i as the payoff of agent i is strictly larger than other agents:

$$adv_i = \bigvee_{x \in \{0..100\}} (goal(i, x) \wedge \bigwedge_{j \in Ag \setminus \{i\}, x > y \geq 0} goal(j, y))$$

Second-mover advantage might be defined as follows:

$$\bigwedge_{i \in Ag} ((\neg turn_i \wedge \langle\langle\rangle\rangle \bigcirc turn_i) \rightarrow \langle\langle i\rangle\rangle \Diamond adv_i) \qquad \text{(Second-mover advantage)}$$

Other examples include (Zero-sum) (here given for a two-player game):

$$\langle\langle\rangle\rangle\Box(terminal \rightarrow ((win_1 \wedge lose_2)XOR(draw_1 \wedge draw_2)XOR(lose_1 \wedge win_2)) \quad \text{(Zero-sum)}$$

Note that although we currently have modelled the outcome propositions as booleans, one can do this easily with numbers as well, enabling also easy representations of constant-sum games.

Finally, we have the following characterises one-shot strategic form games with symmetry payoffs:

$$Strategic \wedge \bigwedge_{x,y \in \{0..100\}} Symmetry(x, y) \quad \text{(Strategic Symmetry)}$$

where,

$$\bigwedge_{i \in Ag} turn_i \wedge \langle\langle\rangle\rangle \bigcirc terminal \quad \text{(Strategic)}$$

$$(\langle\langle Ag\rangle\rangle \bigcirc (goal(1, x) \wedge goal(2, y)) \rightarrow \langle\langle Ag\rangle\rangle \bigcirc (goal(1, y) \wedge goal(2, x)) \quad \text{(Symmetry}(x, y))$$

Note that, since we assume that all Ac_i and Ac_j are disjoint when $i \neq j$, in (Symmetry(x, y)) agents do not need to be able to 'swap actions', they only need swap outcomes.

4.2.4 Special Properties for Tic-Tac-Toe

We now consider properties specific to our running example, Tic-Tac-Toe. For this game, we denote the players with Xplayer and Oplayer, respectively. The atom $turn_i$ says that it is player i's turn: $turn_i \leftrightarrow (\neg terminal \wedge \langle\langle i\rangle\rangle \bigcirc \neg done(i, noop))$, i.e., it is player i's turn if it is not in a terminal state, and he can assure to have done anything else than a *noop* action. Let $c(i, j, w)$ (with $1 \leq i, j \leq 3$ and $w \in \{o, x, b\}$) abbreviate $cell(i, j) = w$ ('$cell(i, j)$ shows currently symbol w'). Finally, *XOR* denotes exclusive or.

Now, our game designer may want to verify that the property that certain configurations on the board will never be reached (e.g., (iCell) expresses the invariant that we don't have two o's and one x in the game in the first row and

only blanks everywhere else – recall that the player who starts is Xplayer). Such properties need not be about invariants, but can also be, for instance, about the progress in the game, or about persistence of written cells or non-written ones (Persistence(x)) (saying that a written X is persistent over any move, a non-written X is persistent under any move of the other player(s)). Using our atoms that recall what is true in the previous state, we can even specify the exact effect of any action: (Write(x)) says that wherever we are in the game, saying that it is x's turn is equivalent to saying that in every next state, there is exactly one cell that is written with an x now, but was blank before.

$$\langle\langle\rangle\rangle\Box\neg(c(1,1,o) \wedge c(1,2,o) \wedge c(1,3,x) \wedge \bigwedge_{i\neq 1} c(i,j,b) \qquad \text{(iCell)}$$

$$\langle\langle\rangle\rangle\Box(\bigwedge_{1\leq i,j\leq 3}(c(i,j,x) \rightarrow \langle\langle\rangle\rangle\bigcirc c(i,j,x)) \wedge (\neg turn_{Xplayer} \rightarrow$$
$$\bigwedge_{1\leq i,j\leq 3}(\neg c(i,j,x) \rightarrow \langle\langle\rangle\rangle\bigcirc\neg c(i,j,x)))) \qquad \text{(Persistence}(x)\text{)}$$

$$\langle\langle\rangle\rangle\Box(turn_{Xplayer} \leftrightarrow \langle\langle\rangle\rangle\bigcirc XOR_{1\leq i,j\leq 3}(c(i,j,x) \wedge c(i,j,b)_{old})) \qquad \text{(Write}(x)\text{)}$$

Regarding game playing, of course it is interesting to know what parties can achieve, in a given game. (A designer of) player i might in particular be interested whether the following instantiation of (Strong Winnability) holds: is it the case that $\langle\langle i\rangle\rangle\Diamond win_i$? In Tic-Tac-Toe, no player is guaranteed a win, i.e., (Strong Winnability) is not true for Tic-Tac-Toe. Indeed, for most interesting games, (Strong Winnability) does not hold.

Let $happy(C) = \bigwedge_{i\in C}(win_i \vee draw_i)$. For instance, (Coalition) expresses that coalition C has some reason to cooperate: by doing so, everybody is reasonably happy, while there is no subset of C that guarantees that. As another example, (R(i, a)) considers whether a is a reasonable move for i: that is, it cannot achieve less than what it currently can achieve, by performing a. This is an example of a property one might want to check in several states of the game, not just the root.

$$\langle\langle C\rangle\rangle \Diamond happy(C) \wedge \neg \bigvee_{C' \subset C} \langle\langle C'\rangle\rangle \Diamond happy(C') \qquad \text{(Coalition)}$$

$$happy(i) \wedge turn_i \wedge \langle\langle i\rangle\rangle \bigcirc (done(i,a) \wedge happy(i)) \qquad (\mathrm{R}(i,a))$$

4.3 The GDL2RML Translator

In this section, we describe our work on how to verify the games in GDL using an ATL model checker. The main purpose of our work is to show a method using existing ATL model checking tools on the verification of GDL games, rather than developing a model checking tool from the scratch. In this way, we can add value to the work that has been done by other people.

We built a translator, GDL2RML, from GDL descriptions to representations in the Reactive Modules Language (RML). RML is the model description language of the ATL model checker MOCHA. Using GDL2RML, we can verify properties expressed in ATL via MOCHA.

In the following subsections, we first introduce MOCHA and RML, then explain the design of the GDL2RML translator, and finally discuss the correctness and evaluation of the GDL2RML translator.

4.3.1 MOCHA and RML

The ATL model checker we use is MOCHA, which was developed by Alur et al. [7, 2], some of whom are also the inventors of ATL. To our knowledge, this is by far the standard model checking tool for ATL. It has been applied in the verification of various systems, such as the shared-memory multiprocessor systems [37], the asynchronous processes [3], and the dataflow processors [36]. The practical model checking complexity using MOCHA was studied in [80].

The model description language of MOCHA is the Reactive Modules Language (RML), which is rich enough to model systems with heterogeneous components: synchronous, asynchronous, speed-independent or real-time, finite or infinite state, etc. Here we briefly introduce RML. An RML specification consists of a set of modules. A module can be seen as an agent; it consists of a set of variables and a set of rules to define the evolution of the variables that are controlled by the module. The input variables are called *external* variables, and

```
atom xyz
 controls x
 reads x, z
 awaits y
init
 [] true -> x':=0
 [] true -> x':=1
update
 [] y'=true & z = false -> x':=x+1
 [] y'=true & z = true -> x':= x
 [] y'=false -> x':=x-1
endatom
```

Figure 4.1: An example of an atom

the output variables are called *interface* variables. A module controls its interface variables and its private variables. Within a module, the basic construct is an atom. A simple example of an atom is given in figure 4.1.

This definition has three parts: 1) a declaration of the variables that are *controlled, read,* or *awaited*; 2) an `init` part; and 3) an `update` part. An atom can write the variables that it `controls`, read the ones it `reads` or it `awaits`. The primes besides the variables refer to their values in the next round. Here, awaiting the value of a variable y means it reads the value that y will receive in the next round, which is determined by another atom. For instance, in the example, the next value of x is evaluated using the current value of z, and *after* the next value of y is specified. The `init` part initializes the value of x by a set of guarded commands starting with a `'[]'`. A guarded command statement consists of two parts: *a guard*, that is a boolean expression specifying when the guarded command can be executed, and a list of *commands*, used to specify the next value of the controlled variables. If several guards are true, the system randomly chooses an associated command. The `next` part is different to the `init` in two ways: first, it can repeatedly execute after the first round, while `init` only executes in the first round; second, it can take boolean expressions with variables as guards, while `init` can only have the guard `true`.

The state of a system at one time point is completely captured by the valuations of the variables that the system controls. The evolution of the state of the system is decided by the initial state and the update commands in each atom.

4.3.2 Design of the GDL2RML Translator

Given a GDL description, how can we obtain a representation in RML which characterises the same system (game structure)? Basically, we have to take care of this for a *state* and the *change* of a state. Both in GDL and RML, a state is represented by a set of propositions or variables. And for the change of a state, GDL uses a set of rules in a logical programming language, while RML uses guarded commands. GDL rules are different from RML guarded commands in two ways: 1) a GDL rule can specify the value of one proposition or variable only, but an RML guarded command can specify more than one; 2) all GDL rules will be executed if their conditions are true, but only one RML guarded command will be executed within the same atom on any given round.

In RML, we need to specify which propositions or variables belong to which module, where modules can be seen as agents. So the main tasks of our GDL2RML translator are:

- to specify the roles in GDL as modules in RML,
- to specify the propositions controlled by each module,
- to specify the initial state and the corresponding update mechanism.

We cut a GDL description Γ into four parts: $\Gamma_{role}, \Gamma_{init}, \Gamma_{next}, \Gamma_{glob}$, where Γ_{role} is a collection of the rules with keyword *role*, Γ_{init} is a collection of the rules with keyword *init*, Γ_{next} is a collection of the rules with keyword *next*, and Γ_{glob} contains the rest. Our GDL2RML translator is written in Java, and processes the rules in these four categories as follows.

Roles For every rule in Γ_{role}, we associate it with a module; moreover, we introduce a special module called Gmaster, which takes the same responsibility as the game master in the General Game Playing competition (GGP) [26]. The main duties of the Gmaster are: to serve the players with the current game board state, to read the actions of the players, and to update the board state accordingly. The behaviour of Gmaster is deterministic, and it will not influence the outcome of a game. In terms of ATL, we have that, for any coalition C, $\langle\langle C \cup \{Gm\}\rangle\rangle \Diamond \varphi \leftrightarrow \langle\langle C\rangle\rangle \Diamond \varphi$. This justifies why we left out Gmaster Gm in our analysis in Section 4.2.

Propositions and Variables Each player module controls their own action variables, and all the rest are controlled by the Gmaster. In other words, the players decide about their move, and all its consequences are then determined. To be more specific, we use a variable `DONE_X` for each player X, and the scope of this variable is easily identified by the clauses with keywords `'does X'` in Γ_{next}. For example, in the Tic-Tac-Toe game in Figure 3.1, we have a clause `(does xplayer (mark ?m ?n))`, so we add `MARK_1_1,...,MARK_3_3` to the domain of `DONE_XPLAYER`, given that the scope of `?m` and `?n` are determined by the context. The reason to choose a `DONE` prefix is related to the update mechanism, which will be introduced shortly.

The propositions for the Gmaster module are directly obtained from Γ. For example, we have in Figure 3.1 a rule `(<= terminal (line x))`, so we take `TERMINAL` and `LINE_X` as propositions. Theoretically, we can represent the state using only propositions, but to make our representation more efficient, we choose some variables to have a richer domain. This will be explained in more detail in the case study in section 4.4.

The Initial State As we mentioned earlier, a state is a full characterization of the system in a particular time point. In Γ_{init}, GDL specifies the propositions that are true initially, but not necessary *all* the propositions that are true, as some global rules in Γ_{glob} might make some propositions true as well. For example, suppose Γ_1 consists of the following two rules: `(init p)` and `(<= q p)`. In the initial state, we first know `p` is true, and then know that `q` is true by the global rule `(<= q p)`. So we need to do some computation, w.r.t $\Gamma_{init} \cup \Gamma_{glob}$, to get a complete picture of the initial state. But RML does not make it possible to do computations for the initial state, as all the guards in the *init* part can only be `true` (see Figure 4.1).

We have two design choices here: either we figure out all the initial values of the variables and then specify their values directly using `[] true -> x':=value` in RML, or we add an extra round to allow the modules' `update` part to compute the full initial state. We take the later approach, as we want to delegate all the work of constructing the game system to MOCHA. Therefore, we introduce a special variable `preinit`, the idea being that we make `preinit` true initially and then false always afterwards. If we call this special state produced by RML the 'pre-init' state, the real init state is then the computed successor of the pre-init state. In the example of Γ_1 above, p would be true in pre-init, and p and q in

init. We make sure that there is only one init state. In the following, when we refer to the "init state", it should be understood that we are *not* referring to the pre-init state.

The Update Mechanism In RML, state changes are made via the update construction, which is specified with keyword `update`. There are two types of update rules in GDL, namely Γ_{glob}, which gives constrains globally, and Γ_{next}, which talks about the future. Accordingly, we have to deal with both of them in RML.

For the rules in Γ_{glob}, we add one rule in the atom which has the head of the rule as a controlled variable. And we use primed versions of the variables as they all update in the same round, and the ones in the guards are updated earlier than those in the commands. The dependency requires that there is no circularity, and this is checked by MOCHA automatically. Here is an example from Figure 3.1: for the GDL rule `(<= terminal (line x))`, we have an update rule in RML: `LINE_X' -> TERMINAL':=true`, which says that if `LINE_X` is true in the next round, then `TERMINAL` is true in the next round as well. Note that `LINE_X` is an *awaited* variable.

For the rules in Γ_{next}, we also add one rule in the atom which has the head of the rule as a controlled variable. For example: a GDL rule

```
(<= (next(cell 1 1 x)) (does xplayer (mark 1 1)) (cell 1 1 b))
```

can be translated to an update rule:

```
CELL_1_1=B & DONE_XPLAYER'=MARK_1_1 -> CELL_1_1':=X.
```

This rule says if the Cell(1,1) is blank currently, and Xplayer marks Cell(1,1), then in the next state, the Cell(1,1) becomes X.

How does the whole system evolve? In GDL, the players (roles) make a choice and the game master uses them to update the state, according to the Γ_{glob} and Γ_{next} rules. This continues until a terminal state is reached. In RML, we will do the same thing, but an important question here is how to record the players' actions in a state. For example, suppose in the current state, player X is allowed to make a move `MARK_1_1`. Shall we have a proposition `DOES_X_MARK_1_1` to indicate that player X will make this move in next state? No, because (1) this would cause the current state to only have one successor, and (2) we do not

intend to say that X *does* the `MARK_1_1` move in the current state, but only like to reason hypothetically what would happen if he *would* make that move. The fact that X makes a certain move should be recorded in the *successor* state associated with that move, and not in the current state. Therefore, we introduce the `DONE_X` variable for each player X, to record the actions made by X in the previous round. The update process in RML starts as follows: all the `DONE_X` variables are given a value in the players' module, and then the Gmaster module uses the rules from Γ_{next} to specify the variables in the head of these rules using the update construct, and finally Gmaster does the same with rules translated from Γ_{glob}. For instance, the effect of X performing a `MARK_1_1` move is captured by the following atom in the module Gmaster.

```
update
[]DONE_Xplayer'=MARK_1_1 & CELL_1_1=B -> CELL_1_1':=X
[]DONE_Oplayer'=MARK_1_1 & CELL_1_1=B -> CELL_1_1':=O
[]~(CELL_1_1=B) -> CELL_1_1' :=CELL_1_1
[]~(DONE_Xplayer'=MARK_1_1 | DONE_Oplayer'=MARK_1_1)
   & CELL_1_1=B -> CELL_1_1':=B
```

This says that if Xplayer's chosen action is to mark Cell(1,1), and this cell is currently blank, it will become marked with X, and similarly for Oplayer and the symbol O. If Cell(1,1) was already not blank, it keeps its value, and, finally, it stays blank if it was blank and nobody wrote on it in this round.

4.3.3 Correctness and Evaluation

How can we ensure the correctness of the GDL2RML translator? Here the 'correctness' means that the original GDL description specifies the same game model as its GDL2RML translation. In the previous chapter, we have formally defined the game models for the GDL descriptions. Ideally, we shall also define the game models of the RML descriptions, and formally prove that the game model of a GDL description Γ corresponds to the game model of the translation of Γ in RML. This requires a formalisation of RML with game semantics, which is out of the scope of this thesis. But we do have two approaches to ensure certain degree of the correctness. The *first approach* is to check whether all the propositions, variables and the rules have been mapped correctly. We get this

level of assurance by checking the design of the GDL2RML translator. This might still prone to human errors, so we have a *second approach*, which is to use the model checker MOCHA to verify properties of the GDL2RML translations. If the MOCHA results agree with the truth of those properties in the original game, then we get certain degree of assurance that our translation is correct. Of course when the MOCHA results do not conform the truth of those properties, this approach alone does not tell whether the original GDL description does not describe the game properly, or the GDL2RML translator is problematic.

As for the evaluation of the GDL2RML translator, we have tested it with a number of examples, such as Maze, Buttons and Tic-Tac-Toe, from the game depository[2] of the General Game Playing Website. All these examples were translated within several seconds in a Dural-Core Linux Machine, and the verification results in MOCHA are all as desired. In the next section, we will present a concrete case study to show that our translation of Tic-Tac-Toe has produced desired results (see Fig 4.2). Compared with the programming-oriented brute-force method mentioned in [26], our method has two advantages. First, we do not need to write a program to expand the game models, as the model checkers automatically generate the game models from RML descriptions. Second, we can specify the properties in ATL in a more abstract way than specifying them in a programming language, so that we do not need to deal with the details in the level of game states. Our GDL2RML translator is still a prototype tool; in theory, it shall automatically translate any GDL descriptions to RML descriptions, but in practice we still need to manually add some tags into GDL descriptions to reduce the numbers of variables in the translation, in order to reduce the model checking time in MOCHA.

4.4 Case Study and Experimental Results

4.4.1 Introduction

In this section we do a case study in the context of the game of Tic-Tac-Toe using our GDL2RML tool and MOCHA.

The game of Tic-Tac-Toe is often used as an example to introduce game complexity, which involves both *state space complexity* and *game-tree complexity*. For Tic-Tac-Toe, a simple upper bound for the size of the state space is $3^9 =$

[2]URL: http://visionary.stanford.edu:4000

19,683, as there are three states for each cell and nine cells. This is of course quite rough, since there are many illegal states, such as a state with all 'X' occupying the spaces. After removing these illegal states, we get 5,478 states. And when rotations and reflections of positions are regarded as identical, there are only 765 essentially different positions. A simple upper bound for the size of the game tree is 9! = 362,880, as there are nine positions for the first move, eight for the second, and so on. After removing the illegal states there are 255,168 possible games; and when rotations and reflections of positions are considered the same, there are only 26,830 possible games[3]. Compared with the complexity of Chess[4], the complexity of Tic-Tac-Toe is rather small. Nevertheless it has 26,830 possible games, which is not trivial.

Tic-Tac-Toe has been modelled and verified successfully by model checking tools; in [41], the SPIN model checker [40] was used, and the properties were expressed in LTL. One clear advantage of our approach is that we are able to verify a bigger class of properties, especially those can only be expressed in ATL.

4.4.2 Playability of Tic-Tac-Toe in MOCHA

For the translation from a GDL description to an RML description, we have illustrated the main idea in the previous section. What we want to stress here is the controlled variables for Gmaster. Most of them are boolean variables, and only a few can take multiple values, e.g., `CELL_1_1` $\in$ { `B,X,O`}. Alternatively, we can use three booleans: `CELL_1_1_B, CELL_1_1_X,` and `CELL_1_1_O`. Then the equivalent expression of `CELL_1_1=X` is

```
CELL_1_1_B=false & CELL_1_1_X=true & CELL_1_1_O=false.
```

We choose the former representation for the sake of compactness.

The general playability conditions are presented in section 4.2. Now we tailor them specifically for Tic-Tac-Toe. We select a few representative properties and give concrete representations that are accepted by model checker MOCHA. The purpose is to show how our work is used in practice.

[3]http://en.wikipedia.org/wiki/Tic-tac-toe

[4]C. Shannon gave an estimation in [69] with the size of state space 10^{43} and the size of game-tree 10^{120}.

Coherence Properties The first coherence property we pick is Playability:

$$\langle\langle\rangle\rangle\Box(\neg terminal \rightarrow \bigwedge_{i \in Ag} has\_legal\_move_i)$$

For $i = Xplayer$, $has\_legal\_move_i$ can be represented as

```
( LEGAL_X_MARK_11 | LEGAL_X_MARK_12 | LEGAL_X_MARK_13
| LEGAL_X_MARK_21 | LEGAL_X_MARK_22 | LEGAL_X_MARK_23
| LEGAL_X_MARK_31 | LEGAL_X_MARK_32 | LEGAL_X_MARK_33
| LEGAL_X_NOOP)
```

The rest is straightforward.

The second coherence property is *GameOver*:

$$\langle\langle\rangle\rangle\Box((terminal \wedge \varphi) \rightarrow \langle\langle\rangle\rangle\Box(terminal \wedge \varphi))$$

Let us look at an instantiation of φ: suppose φ here means that Xplayer wins. Its representation in MOCHA is

```
<<>>X(<<>>G((TERMINAL&GOALX=g100)=><<>>G(TERMINAL&GOALX=g100)))
```

Note that we have some small notional differences. Here `X` corresponds to $\bigcirc$, `G` to $\Box$, `&` to $\wedge$, and `=>` to $\rightarrow$. The reason to have `<<>>X` at the beginning is that we have an extra, "pre-initial" initial state. We have explained the reason to have such state in Section 4.3.2.

The third coherence property we pick is Turn:

$$\langle\langle\rangle\rangle\Box(turn_i \leftrightarrow \neg legal(i, noop))$$

The case with i being *Xplayer* is:

```
<<>>X <<>> G (turn=Xplayer <=> ~LEGAL_Xplayer_NOOP) .
```

The last coherence property we pick is Termination:

$$\langle\langle\rangle\rangle\Diamond terminal$$

The MOCHA representation is straightforward: `<<>> X <<>> F terminal`, where `F` is the MOCHA notation for $\Diamond$.

Fairness Properties Now we pick two fairness properties:

$$\bigvee_{i \in Ag} \langle\langle i \rangle\rangle \Diamond win_i(StrongWinnability) \text{ , and } \bigwedge_{i \in Ag} \langle\langle Ag \rangle\rangle \Diamond win_i(WeakWinnability).$$

The representations are

```
<<>>X(<<Xplayer>>F GOALX=g100 | <<Oplayer>>F GOALO=g100)
```

and

```
<<>>X(<<Xplayer,Oplayer>>F GOALX=g100 & <<Xplayer,Oplayer>>F GOALO=g100)
```

respectively.

4.4.3 Playing Tic-Tac-Toe via Model Checking

Although our main motivation in this work is to consider the analysis of games from the view point of a game designer, it is also worth speculating about the use of our approach to play GDL games via Model checking. Let us suppose the following situation in Tic-Tac-Toe (Xplayer moves first).

X	O	
	X	
		O

Now it is Xplayer's turn. The questions are:

i. Is there a winning strategy for Xplayer in the current state?

ii. If so, which move should Xplayer take?

There is indeed a winning strategy for Xplayer, namely, by marking the $Cell(2, 1)$ (see below). In that case, no matter how Oplayer responds, Xplayer can mark either $Cell(2, 3)$ or $Cell(3, 1)$ in its next turn.

X	O	
X	X	
		O

We show that we can answer these questions via model checking.

First, "*Xplayer has a winning strategy*" is expressed as $\langle\langle Xplayer\rangle\rangle\Diamond win_{Xplayer}$ in ATL and as `<<Xplayer>>F GOALX=g100` in MOCHA. Given game model G, and current state s, the question 1 amounts to checking whether $G, s \models_{\mathsf{ATL}} \langle\langle Xplayer\rangle\rangle\Diamond win_{Xplayer}$. In MOCHA, we can only check a property with respect to the initial state, namely s_0, but we can get around using the following approach. We characterise a state s by a formula $\varphi(s)$, so instead of checking $G, s \models_{\mathsf{ATL}} \langle\langle Xplayer\rangle\rangle\Diamond win_{Xplayer}$, we can check $G, s_0 \models_{\mathsf{ATL}} \langle\langle\rangle\rangle\Box(\varphi(s) \rightarrow \langle\langle Xplayer\rangle\rangle\Diamond win_{Xplayer})$. For the above example, $\varphi(s)$ can be $Cell(1,1,X) \wedge Cell(1,2,O) \wedge Cell(2,2,X) \wedge Cell(3,3,O) \wedge \bigwedge_{x,y=rest} Cell(x,y,B)$. We denote the MOCHA representation of $\langle\langle\rangle\rangle\Box(\varphi(s) \rightarrow \langle\langle Xplayer\rangle\rangle\Diamond win_{Xplayer})$ as *sXWin*.

Now, suppose we got a positive answer to the question 1. To answer the question 2, we use an action variable *DONEX* to guide the search for a proper move. The idea is to select a legal move for Xplayer, and then to check whether Xplayer still has a winning strategy under this move. If so Xplayer shall take it; if not, Xplayer will check the a different legal move; the existence of a winning strategy guarantees that there is such a move. To be more specific, suppose Xplayer chooses $mark(2,1)$, it is to check: $G, s_0 \models_{\mathsf{ATL}} \langle\langle\rangle\rangle\Box(\varphi(s) \rightarrow \langle\langle Xplayer\rangle\rangle\bigcirc(DONEX = MARK\_2\_1\langle\langle Xplayer\rangle\rangle\Diamond win_{Xplayer}))$.

We denote the MOCHA version of this formula as 'sXWin_by_mark21'. If the answer is positive, it means Xplayer's move *Mark_2_1* is indeed a move leading towards a winning position.

There is of course a question "what if there is no winning strategy in the current position?". We believe that it is interesting to explore a position evaluation function, which estimates the value or goodness of a position, and its connection with ATL properties; but we would leave this for further research.

4.4.4 Experimental Results on Tic-Tac-Toe

Here we present experimental results to show that the analysis described above can be done in reasonable time with moderate computing resources. For these experiments, we ran MOCHA under Linux kernel 2.6.20 i686 with a Dural-Core 1.8Ghz CPU and 2GB RAM. The table in Figure 4.2 gives timings for checking the various properties listed in the previous section.

These results indicate that our tool can generate correct results in a reasonable amount of time. We believe that there is much room for improvement with

Property	Results	Time
GameOver	passed	2sec
Turn	passed	0.3sec
Termination	passed	4min49sec
Playability	passed	0.4sec
Strong Winnability	failed	23sec
Weak Winnability	passed	4min01s
sXWin	passed	1min06sec
sXWin_by_mark21	passed	1min59sec

Figure 4.2: Verification results of Tic-Tac-Toe

respect to these results. In particular, it might be useful in future to consider investing some effort in optimising the translation process from GDL to RML, particularly with respect to the number of variables produced in the translation. Even moderate optimisations might yield substantial time and memory savings.

4.5 Summary

This chapter investigated the specification and verification of games described in GDL. In particular, two main contributions were made. First, we characterized a class of playability conditions that can be used to express the correctness of the games specified in GDL. Second, we developed an automated tool, the GDL2RML translator, that can transform a set of GDL descriptions into RML specifications, and we can verify the playability conditions over these RML specifications using an off-the-shelf ATL model checker, MOCHA. In future research, we believe it is worthwhile to refine this work on formal verification of GDL descriptions. The main issues are likely to be the efficiency and scalability of our automated tool.

Chapter 5

Bridging Action and Time Logics

5.1 Introduction

This chapter provides a study of logical frameworks that not only deal with action or time, but also deal with knowledge of multi-agent systems. In particular we study a correspondence between Dynamic Epistemic Logic (DEL) and Temporal Epistemic Logic (TEL). As shown in Chapter 2, these two logical frameworks are capable of modelling multi-agent systems with agents' knowledge and its change. However, there is a large difference in terms of model checking: in DEL the interpretation of a dynamic epistemic formula is over a state model, which represents a static view of a multi-agent system; while in TEL, the interpretation of a temporal epistemic formula is over an interpreted system, in which the full history of a system is unfolded.

The presented frameworks interact both on the level of logical languages and on the level of semantic objects, and it is precisely this interaction that is the subject of the underlying investigation. Various results have already been achieved. The relation between Kripke models and interpreted systems has been investigated by Lomuscio and Ryan in [47]. They focussed on an interpreted system named *Hypercube System* that corresponds to the cartesian product of all local state values, and that has no dynamic features. In the correspondence, local state values become boolean propositional variables. Their approach suits Kripke models where all states have different valuations, which is not generally the case. A recent study by Pacuit [55] compares the history-based approach by Parikh and Ramanujam [56] to interpreted systems, with runs. This addresses the relation between Kripke models *with histories consisting of event sequences* (in our

case this primitive is derived and called a forest model) and interpreted systems. Pacuit handles *partial* observability of agents, when the agents perceive only some but not all of a sequence of events. He does not address in [55] the partial observability common in dynamic epistemics, where only an aspect of an event is observable, not the full event. Other recent work by van Benthem, Gerbrandy and Pacuit [79], rooted in older work [78, 75], gives a precise relation between temporal epistemics and dynamic epistemics. In their approach, each action w in an action model M corresponds to a unique *labelled* modality $\bigcirc_{(\mathsf{M},\mathsf{w})}$, interpreted in a linear-time temporal logic, such that a dynamic epistemic formula of the form $[\mathsf{M},\mathsf{w}]\varphi$ ('after execution of event '(M,w)' it holds that φ') is true in a Kripke model with epistemic accessibility relations, if and only if a temporal epistemic formula $\bigcirc_{(\mathsf{M},\mathsf{w})}\varphi$ is true in an 'enlarged' Kripke model that is constructed using two copies of the former and an accessibility relation for $\bigcirc_{(\mathsf{M},\mathsf{w})}$-execution that connects them. This is a *forest* that we will introduce later. We have straightforwardly applied their elegant approach. Unlike them, we do not assume a protocol, but compute it based on the structure of a given formula.

Much recent work in model checking is based on temporal epistemics describing interpreted systems (MCMAS [67], MCK [25], and see also [73]), and some recent work is based on dynamic epistemics describing model updates (DEMO, [94]). Our work is intended to draw a connection between these two different model checking approaches.

This chapter is organized as follows. Section 5.2 contains definitions on logical languages and semantics that we are going to discuss. Section 5.3 presents two translations, a syntactical one and a semantical one, from PAL (a special case of DEL) to NTEL (a variant of TEL); and then we prove a correspondence between PAL and NTEL. Section 5.4 extends the results from PAL to the more general case DEL. A brief summary is given in Section 5.5.

5.2 Logical Preliminaries

In this section, we present the logical preliminaries of our work. We introduce *four* structural primitives and *two* languages. The structures are:

- *state models*, which are Kripke models with S5 accessibility relations representing agents' knowledge about states;

- *action models*, which are Kripke models with S5 accessibility relations representing agents' knowledge about actions;
- *forest models*, which are Kripke models with not just accessibility relations representing agents' knowledge of states but also accessibility relations representing state transitions;
- *action-based interpreted systems*, which are based on well-accepted interpreted systems.

The reason that we restrict the state models to have only S5 accessibility relations is because the framework we want to relate it to, namely that of action-based interpreted systems, has S5 for its accessibility relations. The first two models were introduced in Chapter 2, and the remaining two, forest models and action-based interpreted systems, will be defined soon.

The languages are those of *dynamic epistemic logic* and a variant of *temporal epistemic logic* which one could think of as 'next-time temporal epistemic logic'. The former can be given meaning both on state models and on forest models; the latter both on forest models and on interpreted systems. As global parameters to both the languages and the structures we have a set Ag of n agents, and a (countable) set Q of atoms q, and to action-based structures, we assume a finite set of W of actions w.

5.2.1 Languages

As shown in Section 2.2.4, the language $\mathcal{L}_{\mathsf{DEL}}$ of Dynamic Epistemic Logic is inductively defined as follows

$$\varphi \quad ::= \quad q \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid K_i\varphi \mid C_B\varphi \mid [\mathsf{M}, \mathsf{w}]\varphi$$

where $q \in Q$, $i \in Ag$, $B \subseteq Ag$, and (M, w) a pointed action model. Without loss of generality, we assume that all points of all action models are differently named, so that we can associate a particular w with the pointed model (M, w) whenever convenient. For the special case of singleton action models with reflexive accessibility relations for all agents, i.e. public announcements, we write $[\varphi]\psi$ where φ is the precondition (the announced formula).

The dynamic part of $\mathcal{L}_{\mathsf{DEL}}$ is the action modality and it can be seen as representing one time step. Now we want to connect it to a temporal language. Natu-

rally, we could associate each action modality with a one-step temporal modality. We define the language of Next-time Temporal Epistemic Logic as follows.

Definition 5.1 (Language $\mathcal{L}_{\mathsf{NTEL}}$)**.** *Given a set of actions* W*, the language of Next-time Temporal Epistemic Logic* $\mathcal{L}_{\mathsf{NTEL}}$ *is inductively defined as*

$$\varphi \ ::= \ q \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid K_i\varphi \mid C_B\varphi \mid \bigcirc_{\mathsf{w}} \varphi$$

where, $q \in Q$*,* $i \in Ag$*,* $B \subseteq Ag$*, and* $\mathsf{w} \in \mathsf{W}$*.*

There are some differences with the temporal language $\mathcal{L}_{\mathsf{TEL}}$(see Section 2.2.3), as the next-time temporal modalities here are labelled with actions. Also we do not need $\mathcal{U}$ ('until') operators in our investigation. We call φ an NTEL formula if $\varphi \in \mathcal{L}_{\mathsf{NTEL}}$.

5.2.2 Structures

We first recap the key parts of the definitions of state models and action models introduced in Chapter 2.

A *state model* is a structure $\langle W, \sim_1, \ldots, \sim_n, \pi\rangle$ where W is a domain of possible states, for each agent i, $\sim_i$ is an S5 accessibility relation between states expressing the states that are indistinguishable from each other for that agent, and where $\pi : W \to \wp(Q)$ is a valuation (or interpretation) that determines for each state which atoms are true in that state.

An *action model* M is a structure $\langle \mathsf{W}, \sim_1, \ldots, \sim_m, \mathsf{pre}\rangle$ such that W is a domain of *action points*, and for each $i \in Ag$, $\sim_i$ is an equivalence relation on W, and $\mathsf{pre} : \mathsf{W} \to \mathcal{L}$ is a precondition function that assigns a *precondition* $\mathsf{pre}(\mathsf{w})$ in language $\mathcal{L}$ to each $\mathsf{w} \in \mathsf{W}$.

We introduce a structure that adds an extra dimension to a state model.

Definition 5.2 (Forest Model)**.** *Given a set of atomic propositions* Q*, a set of actions* W*, a forest model is a structure*

$$\langle W, \sim_1, \ldots, \sim_n, \{\rightarrow_{\mathsf{w}} \mid \mathsf{w} \in \mathsf{W}\}, \pi\rangle$$

where

- W *is a set of states;*
- $\sim_i$ *is an S5 accessibility relation of agent* i *for each* $i \in Ag$*;*

- $\rightarrow_{\mathsf{w}}$ *is a binary relation on states expressing the execution of action* w *with an* extra condition *such that each state can only have at most one action predecessor, in other words, for each state* $w \in W$*, if there is* $w_1, w_2 \in W$ *and* $\mathsf{w}', \mathsf{w}'' \in \mathsf{W}$ *such that* $w_1 \rightarrow_{\mathsf{w}'} w$ *and* $w_2 \rightarrow_{\mathsf{w}''} w$*, then* $w_1 = w_2$*, and* $\mathsf{w}' = \mathsf{w}''$*;*

- *and* π *is a valuation function from* W *to* $\wp(Q)$*.*

We sometimes write a forest model as $\langle W, \sim_1, \ldots, \sim_n, \{\rightarrow_{\mathsf{w}}\}, \pi\rangle$ *if* W *is clear from the context.*

If we represent all states in W in the form of $(w, \mathsf{w}^1, \ldots, \mathsf{w}^m)$ where w is a state and $\mathsf{w}^1, \ldots, \mathsf{w}^m$ is a sequence of executed actions, then $w_1 \rightarrow_{\mathsf{w}} w_2$ iff $(w_1, \mathsf{w}) = w_2$. For brevity, $((w, \mathsf{w}^1, \ldots, \mathsf{w}^m), \mathsf{w})$ and $(w, \mathsf{w}^1, \ldots, \mathsf{w}^m, \mathsf{w})$ are treated as the same state.

In order to interpret the NTEL formulas, we extend the interpreted system (see Definition 2.16) with actions.

Definition 5.3 (Action-based Interpreted System)**.** *Given a set of atomic propositions* Q*, a set of actions* W*, an action-based interpreted system* $\mathcal{I} = (\mathcal{R}, \{\rightarrow_{\mathsf{w}} | \mathsf{w} \in \mathsf{W}\}, \pi)$ *over* $\mathcal{G}$ *is a system* $\mathcal{R}$ *of runs over a set* $\mathcal{G}$ *of global states with a valuation* π *which decides for each point* (r, m) *a set of atoms* $P \subseteq Q$ *that are true in* (r, m)*. Two points* (r, m) *and* (r', m') *are indistinguishable for* i*, written* $(r, m) \sim_i (r', m')$*, if* $r_i(m) = r'_i(m')$*. Two points in the same run* (r, m) *and* $(r, m+1)$ *are connected by an action* $\mathsf{w} \in \mathsf{W}$*, written as* $(r, m) \rightarrow_{\mathsf{w}} (r, m+1)$*.*

5.2.3 Semantics

In the following, we give *meanings* to the formulas of the languages over the structures we have introduced. More specifically, we interpret DEL formulas over *state models* and *forest models*, and interpret NTEL formulas over *forest models* and *action-based interpreted systems*. This can be illustrated by the diagram in Figure 5.1.

Here we distinguish four different interpretations. $\models_{\mathsf{sd}}$ denotes the interpretation of a DEL formula over a state model; $\models_{\mathsf{fd}}$ denotes the interpretation of a DEL formula over a forest model; $\models_{\mathsf{ft}}$ denotes the interpretation of an NTEL formula over a forest model; and $\models_{\mathsf{it}}$ denotes the interpretation of an NTEL formula over an action-based interpreted system. All these interpretations are

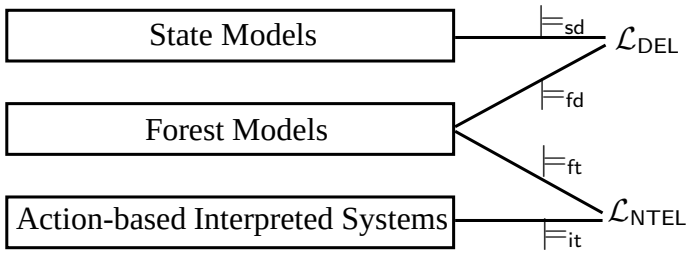


Figure 5.1: Semantics of two languages over three models

defined similarly in terms of atomic propositions, logical connectives and knowledge modalities, which have been introduced in Chapter 2. We focus on clauses of action executions, and the temporal connectives. For action executions, we also mention the special case of public announcement.

Definition 5.4 (Semantics $\models_{\mathsf{sd}}$)**.** *The semantics of* $[\mathsf{M},\mathsf{w}]\psi$ *over a state model* M *is as follows,*

$$M, w \models_{\mathsf{sd}} [\mathsf{M},\mathsf{w}]\psi \quad \textit{iff} \quad M, w \models_{\mathsf{sd}} \mathsf{pre}(\mathsf{w}) \Rightarrow M \otimes \mathsf{M}, (w,\mathsf{w}) \models_{\mathsf{sd}} \psi$$

where $\otimes$ *is the update operation.*

For the special case of public announcement, assuming that $[\varphi]$ *corresponds to the action* w_0 *in public announcement model* M_0*, we have:*

$$M, w \models_{\mathsf{sd}} [\varphi]\psi \quad \textit{iff} \quad M, w \models_{\mathsf{sd}} \varphi \Rightarrow M \otimes \mathsf{M}_0, (w,\mathsf{w}_0) \models_{\mathsf{sd}} \psi.$$

Note that this definition is essentially the same as what we have defined in Section 2.2.4. The following is new.

Definition 5.5 (Semantics $\models_{\mathsf{fd}}$)**.** *The semantics of* $[\mathsf{M},\mathsf{w}]\psi$ *over a forest model* M *is as follows,*

$$M, w \models_{\mathsf{fd}} [\mathsf{M},\mathsf{w}]\psi \quad \textit{iff} \quad M, w \models_{\mathsf{fd}} \mathsf{pre}(\mathsf{w}) \Rightarrow \exists v \textit{ s.t. } w \rightarrow_{\mathsf{w}} v \textit{ and } M, v \models_{\mathsf{fd}} \psi$$

For the special case of public announcement, assuming $[\varphi]$ *corresponds to* w*, we have:*

$$M, w \models_{\mathsf{fd}} [\varphi]\psi \quad \textit{iff} \quad M, w \models_{\mathsf{fd}} \varphi \Rightarrow \exists v \textit{ s.t. } w \rightarrow_{\mathsf{w}} v \textit{ and } M, v \models_{\mathsf{fd}} \psi$$

This needs a bit more explanation. See the following example.

Example 5.1. *Suppose we have a formula* $p \rightarrow [p]\top$. *For any state model* M *and state* w, *we have* $M, w \models_{\mathsf{sd}} p \rightarrow [p]\top$. *This is because if* p *is true in* w, *then a truthful public announcement could be made in* w, *and in the resulting state,* $\top$ *is trivially true. But if we interpret this formula in a forest model, it can be false. A simple example would be a forest model* M' *with only one state* w' *such that* p *is true in* w', *and there is no action successors of that state. So we have* $M', w' \not\models_{\mathsf{fd}} [p]\top$, *as the action relation corresponds to* $[p]$ *is empty. This property of* $\models_{\mathsf{fd}}$ *may seem a bit strange, as it may not directly fit the same expectation as in* $\models_{\mathsf{sd}}$. *Essentially, this is because we do not yet enforce any strong connection of an announcement modality and an action in the forest model. We will see later that formula* $p \rightarrow [p]\top$ *does hold in a special class of forest models that relate to this formula (see Definition 5.10).*

Next, we define the meanings of the formulas with 'next-time' temporal operators in the following way.

Definition 5.6 (Semantics $\models_{\mathsf{ft}}$)**.** *The semantics of temporal formula* $\bigcirc_{\mathsf{w}}\varphi$ *on a forest model* M *is as follows:*

$$M, w \models_{\mathsf{ft}} \bigcirc_{\mathsf{w}} \varphi \;\; \textit{iff} \;\; \exists v \; s.t. \; w \rightarrow_{\mathsf{w}} v \; \textit{and} \; M, v \models_{\mathsf{ft}} \varphi$$

According to this definition, $\bigcirc_{\mathsf{w}}\varphi \wedge \bigcirc_{\mathsf{w}}\neg\varphi$ is satisfiable, as one could imagine that there is a state w with two w successors in one of which φ is true and in the other φ is false. But we will show that this is not satisfiable in a special class of forest models in Definition 5.10.

Let $\mathcal{I} = (\mathcal{R}, \{\rightarrow_{\mathsf{w}}\}, \pi)$ be an action-based interpreted system over a set $\mathcal{G}$ of global states. "Runs r and r' are equivalent to time m" means that the initial segments of r and r' are the same from 0 to m, i.e., $r(0) = r'(0)$ up to $r(m) = r'(m)$. Choosing the bundle semantics as in [87], we now define the meaning of $\bigcirc_{\mathsf{w}}\varphi$ over an action-based interpreted system.

Definition 5.7 (Semantics $\models_{\mathsf{it}}$)**.** *The semantics for* $\bigcirc_{\mathsf{w}}\varphi$ *on an action-based interpreted systems* $\mathcal{I}$ *is as follows,*

$(\mathcal{I}, r, m) \models_{\mathsf{it}} \bigcirc_{\mathsf{w}} \varphi$ *iff there is a run* r' *that is equivalent to* r *to time* m *and* $r'(m) \rightarrow_{\mathsf{w}} r'(m+1)$ *such that:* $(\mathcal{I}, r', m+1) \models_{\mathsf{it}} \varphi$.

Note that this definition also shows a connection of action and time as in $r'(m) \rightarrow_{\mathsf{w}} r'(m+1)$.

5.3 The Case of Public Announcement

In this section, we deal with the case of public announcement action models and a fragment of $\mathcal{L}_{\mathsf{DEL}}$ for public announcement, referred to as $\mathcal{L}_{\mathsf{PAL}}$. Given a formula φ in $\mathcal{L}_{\mathsf{PAL}}$, and a multi-agent state model (M, w), we want to simulate checking the truth of φ in (M, w) by checking the truth of a corresponding temporal epistemic formula in a corresponding interpreted system. The interpreted system is based on (M, w) but should also encode the dynamics that is implicitly present in φ in the form of public announcement operators. It is therefore relative to both φ and (M, w). In other words, we are looking for a *semantic transformation* SEM and a *syntactic translation* SYN (with type: $\mathcal{L}_{\mathsf{PAL}} \to \mathcal{L}_{\mathsf{NTEL}}$) such that:

$$M, w \models_{\mathsf{sd}} \varphi \quad \text{iff} \quad \text{SEM}((M, w), \varphi) \models_{\mathsf{it}} \text{SYN}(\varphi).$$

The image of the actual world w under SEM (a global state s_w) is entirely determined by the role of w in M. It is therefore sufficient to determine $\text{SEM}(M, \varphi)$:

$$M, w \models_{\mathsf{sd}} \varphi \quad \text{iff} \quad \text{SEM}(M, \varphi), s_w \models_{\mathsf{it}} \text{SYN}(\varphi).$$

5.3.1 Syntactic translation

The PAL formulas translate to NTEL formulas in the following way.

Definition 5.8 ($\mathcal{L}_{\mathsf{PAL}}$ to $\mathcal{L}_{\mathsf{NTEL}}$)**.** *Suppose that every action corresponds to a different announcement modality, we define a translation* SYN *from* $\mathcal{L}_{\mathsf{PAL}}$ *to* $\mathcal{L}_{\mathsf{NTEL}}$ *as follows:*

$$\begin{array}{lcl}
\text{SYN}(q) & ::= & q \\
\text{SYN}(\varphi \wedge \psi) & ::= & \text{SYN}(\varphi) \wedge \text{SYN}(\psi) \\
\text{SYN}(\neg\varphi) & ::= & \neg\text{SYN}(\varphi) \\
\text{SYN}(K_i\varphi) & ::= & K_i\text{SYN}(\varphi) \\
\text{SYN}(C_B\varphi) & ::= & C_B\text{SYN}(\varphi) \\
\text{SYN}([\varphi]\psi) & ::= & \text{SYN}(\varphi) \to \bigcirc_{\mathsf{w}}\text{SYN}(\psi)
\end{array}$$

where action w *corresponds to* $[\varphi]$.

We assume that every announcement in different positions corresponds to a different action, so even when two announcement modalities have the same formula, they still get different names. For example, $[\varphi][\varphi]q$ cannot be translated to $\bigcirc_{\mathsf{w}} \bigcirc_{\mathsf{w}} q$, as the first $[\varphi]$ and second $[\varphi]$ are in different positions and have

different dynamic effects. For this reason, we introduce a simple procedure to mark the announcement modalities so that they get a unique name. We mark the n announcements occurring in a formula with indexes from 1 to n in the order of occurrence of their left '[' bracket, when reading the formula from left to right. Then we associate each modality with index i with action w_i. Note that this is not the only way to assign different names to different announcements.

Here is an example of the above translation method.

Example 5.2. *Suppose we have a formula* $[q \wedge [r]K_2 r]C_{12}q \wedge [\top]\neg K_1 q$ *with three announcements. The left bracket '*[*' in '*$[q \wedge \ldots$*' comes first when reading from left to right. Then comes the left bracket of the announcement* $[r]$ *that is a subformula of* $[q \wedge [r]K_2 r]$*. Finally we reach the announcement* $[\top]$ *in the right-hand side of the conjunction. We add indexes to the modalities as follows* $[^1 q \wedge [^2 r]K_2 r]C_{12}q \wedge [^3\top]\neg K_1 q$*, then associate them with three announcement variables as follows*

$$\begin{array}{ll} \mathsf{w}^1 & [^1 q \wedge [r]K_2 r] \\ \mathsf{w}^2 & [^2 r] \\ \mathsf{w}^3 & [^3\top] \end{array}$$

The translation $\textsc{syn}([q \wedge [r]K_2 r]C_{12}q \wedge [\top]\neg K_1 q)$ *then is*

$$((q \wedge (r \to \bigcirc_{\mathsf{w}^2} K_2 r)) \to \bigcirc_{\mathsf{w}^1} C_{12}q) \wedge (\top \to \bigcirc_{\mathsf{w}^3} \neg K_1 q).$$

PAL Protocols The dynamics implicitly present in PAL formula φ can be *identified* with the set of all sequences of public announcements that may need to be evaluated in order to determine the truth of φ. As this is known as a protocol [55], we call this the *protocol of a formula* φ. It can be determined from φ and is therefore another syntactic feature that we can address before applying it in the semantic transformation $\textsc{sem}((M, w), \varphi)$.

Definition 5.9 (Protocol of PAL formula)**.** *The* protocol of a PAL formula *is defined by induction on the formula structure. In the last clause,* w *is the name for the announcement of* φ *in* $[\varphi]\psi$*, and* $\mathsf{w}\textsc{prot}(\psi) = \{\mathsf{w}\mathsf{w}^1 \ldots \mathsf{w}^m \mid \mathsf{w}^1 \ldots \mathsf{w}^m \in$

$\text{PROT}(\psi)\}$, *i.e. the concatenation of* w *to all sequences in the set of* $\text{PROT}(\psi)$.

$$\begin{array}{lcl}
\text{PROT}(q) & ::= & \emptyset \\
\text{PROT}(\neg\varphi) & ::= & \text{PROT}(\varphi) \\
\text{PROT}(\varphi \wedge \psi) & ::= & \text{PROT}(\varphi) \cup \text{PROT}(\psi) \\
\text{PROT}(K_i\varphi) & ::= & \text{PROT}(\varphi) \\
\text{PROT}(C_B\varphi) & ::= & \text{PROT}(\varphi) \\
\text{PROT}([\varphi]\psi) & ::= & \text{PROT}(\varphi) \cup \mathsf{w}\text{PROT}(\psi)
\end{array}$$

This notion of protocol is similar to the one in [55]. The difference is that in our case $\text{PROT}(\psi)$ is not subsequence closed. The protocol of a formula would be subsequence closed if the last clause is changed to $\text{PROT}([\varphi]\psi) ::= \text{PROT}(\varphi) \cup \mathsf{w}\text{PROT}(\psi) \cup \{\mathsf{w}\}$. For a protocol variable we use T.

Example 5.3. *We have that* $\text{PROT}([q][r](q \wedge r) \wedge [r]K_1 r) = \{\mathsf{w}^1\mathsf{w}^2, \mathsf{w}^3\}$, *and that* $\text{PROT}([q \wedge [r]K_2 r]C_{12}q \wedge [\top]\neg K_1 q) = \{\mathsf{w}^1, \mathsf{w}^2, \mathsf{w}^3\}$. *(See previous example.)*

5.3.2 Semantic transformation

The required semantic transformation SEM in $\text{SEM}(M, \varphi)$ is determined in two steps. First, we construct the *forest model*, $\text{F}(M, \text{PROT}(\varphi))$ from the state model M and the protocol $\text{PROT}(\varphi)$ of the public announcement formula φ in a similar way as in [75, 79]. Then we determine an interpreted system $\text{IS}(M')$ corresponding to a forest model M'. We then simply define $\text{SEM}(M, \varphi)$ as $\text{IS}(\text{F}(M, \text{PROT}(\varphi)))$.

Definition 5.10 (Generated Forest Models)**.** *Given a state model* $M = \langle W, \sim_1, ..., \sim_n, \pi\rangle$, $w \in W$, *and a protocol* $\mathsf{T} = \text{PROT}(\varphi)$ *generated from* PAL *formula* φ. *The forest model* $\text{F}(M, \mathsf{T})$ *is defined in three steps.*

(1) *Let* $\mathsf{w}^1 \cdots \mathsf{w}^m$ *be a sequence of actions in protocol* T, *and suppose that these actions belong to public announcement models* $\mathsf{M}_1, \mathsf{M}_2, \cdots, \mathsf{M}_m$ *respectively. Let* M_i *be a state model* $M \otimes \mathsf{M}_1 \cdots \otimes \mathsf{M}_i$, *which is the result of announcing* w^1 *to* w^i *subsequently on* M. *Then* $g(M, \mathsf{w}^1 \cdots \mathsf{w}^m)$ *is a forest model* $M' = \langle W', \sim'_1, \ldots, \sim'_n, \{\rightarrow'_{\mathsf{w}}\}, \pi'\rangle$ *such that*

- $W' = W_M \cup W_{M_1} \cup \cdots \cup W_{M_m}$ *i.e. the set of states obtained from subsequent updates by announcements;*
- $\sim'_i = \bigcup_{j \in [1..m]} \sim_i^j$, *where* $\sim_i^j$ *is the epistemic relation of agent* i *in model* M_j;

- $w \rightarrow_{\mathsf{w}} (w, \mathsf{w})$ *for any* $w, (w, \mathsf{w}) \in W'$*;*

- $\pi'(w) = \pi(w)$ *s.t.* $\exists M_k (w \in W_{M_k}$ & π *belongs to* $M_k)$*.*

(2) *We define a union* $\uplus$ *of two forest models. Given forest model* $M' = \langle W', \sim'_1, \ldots, \sim'_n, \{\rightarrow'_{\mathsf{w}}\}, \pi' \rangle$*, and* $M'' = \langle W'', \sim''_1, \ldots, \sim''_n, \{\rightarrow''_{\mathsf{w}}\}, \pi'' \rangle$*,*

$$M' \uplus M'' ::= \langle W''', \sim'''_1, \ldots, \sim'''_n, \{\rightarrow'''_{\mathsf{w}}\}, \pi''' \rangle$$

where $W''' = W' \cup W''$*,* $\sim'''_i = \sim'_i \cup \sim''_i$ *for all* $i \in Ag$*,* $\rightarrow'''_{\mathsf{w}} = \rightarrow'_{\mathsf{w}} \cup \rightarrow''_{\mathsf{w}}$ *for all* w*, and* $\pi'''(w) = \pi'(w) \cup \pi''(w)$ *for all* $w \in W' \cap W''$*,* $\pi'''(w) = \pi'(w)$ *for all* $w \in W' \setminus W''$*,* $\pi'''(w) = \pi''(w)$ *for all* $w \in W'' \setminus W'$*.*

(3) *We have,*

$$\mathrm{F}(M, \mathrm{PROT}(\varphi)) ::= \uplus_{\tau \in \mathrm{PROT}(\varphi)} g(M, \tau).$$

The construction can be seen as repeatedly merging a model and the modal product of that model and a singleton 'action model' corresponding to an announcement. We refer to the next section for an example illustrating this procedure.

Next, from such a forest model we determine an action-based interpreted system. This is based on a fairly intuitive idea. For each *world* in a forest model we associate it with a *global state* of an interpreted system. This can be achieved by keeping that world as the value of the *environmental state* and for each agent the set of indistinguishable worlds as the value of that agent's *local state*. The valuation π remains as it was. Now for a world w in a state model $M = \langle W, \sim_1, \ldots, \sim_n \rangle$ this recipe delivers a corresponding global state $s = (w, w^{\sim_1}, \ldots, w^{\sim_n})$, where $w^{\sim_i}$ is the i-equivalence class containing w, i.e. $\{w' \in W \mid w' \sim_i w\}$. The same recipe applies, in principle, to worlds $(w, \mathsf{w}^1, \ldots, \mathsf{w}^m)$ in the forest model $\mathrm{F}(M, \mathrm{PROT}(\varphi))$, but here we can somewhat simplify matters by observing that (i) the environment is fully determined by the w in $(w, \mathsf{w}^1, \ldots, \mathsf{w}^m)$ because all events (such as announcements) are defined relative to their combined effect on the agents *only*, and by observing that (ii) public announcements are fully observable by all agents so we can represent them as global parameters. In the following we use $(w, w^{\sim_1}, \ldots, w^{\sim_n}, \mathsf{w}^1, \ldots, \mathsf{w}^m)$ to denote the global state $((w, \mathsf{w}^1, \ldots, \mathsf{w}^m), (w, \mathsf{w}^1, \ldots, \mathsf{w}^m)^{\sim_1}, \ldots, (w, \mathsf{w}^1, \ldots, \mathsf{w}^m)^{\sim_n})$.

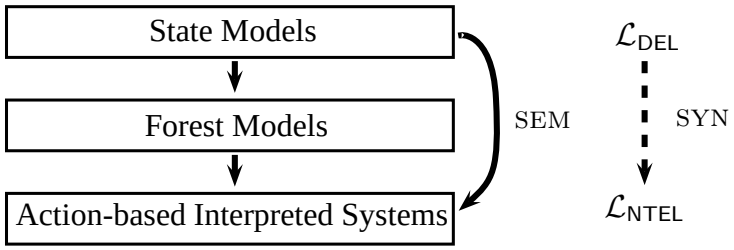
Definition 5.11 (Generated Action-based Interpreted System)**.** *Given a forest model* $M = \langle W, \sim_1, ..., \sim_n, \{\rightarrow_{\mathsf{w}}\}, \pi\rangle$*, we associate* M *with an action-based interpreted system* $\mathcal{I}$*, also written as* $\text{IS}(M)$*, which is a structure* $(\mathcal{R}, \{\rightarrow_{\mathsf{w}}\}, \pi)$ *defined as follows.*

Every state $(w, \mathsf{w}^1, \ldots, \mathsf{w}^m)$ *in the forest model* M *corresponds to a global state* $(w, w^{\sim_1}, \ldots, w^{\sim_n}, \mathsf{w}^1, \ldots, \mathsf{w}^m)$*, where the local state of agent* i *is* $(w^{\sim_i}, \mathsf{w}^1, \ldots, \mathsf{w}^m)$*. We pick every state in forest model* M *with no* $\rightarrow_{\mathsf{w}}$ *successors for any action* w*. A run* $r \in \mathcal{R}$ *is defined for each of such states. Suppose* $(w, \mathsf{w}^1, \ldots, \mathsf{w}^k)$ *is a state in* M*, its associated run* r*, together with* $\rightarrow_{\mathsf{w}}$ *and* π*, is defined as follows:*

- $r(0) = (w, w^{\sim_1}, \ldots, w^{\sim_n})$*;*
- $r(i) = (w, w^{\sim_1}, \ldots, w^{\sim_n}, \mathsf{w}^1, \ldots, \mathsf{w}^i)$ *for all* $1 \leq i \leq k$ *;* $r(i) = r(i-1)$*, otherwise;*
- $r(i-1) \rightarrow_{\mathsf{w}^i} r(i)$ *for all* $1 \leq i < k$*;*
- *The valuations correspond:* $\pi(r(i)) = \pi(w)$*, i.e. all the states in a run have the same valuation.*

It is easy to see that each run is essentially a branch in the corresponding forest model. We have an example in the next section to illustrate this.

The following diagram summarizes the syntactic translation and the semantic transitions we made.

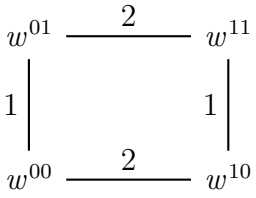


5.3.3 Example

We illustrate the transformations defined in the previous section through the following example.

Consider two agents 1 and 2 and two facts q and r. Agent 1 knows whether q but is uncertain about the truth of r, whereas agent 2 knows whether r but is uncertain about the truth of q. The agents are commonly aware of each other's

factual knowledge and ignorance. In fact, both q and r are true. This is modelled by the following state model (call it M_{init}).

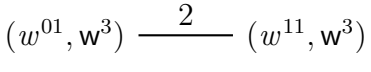


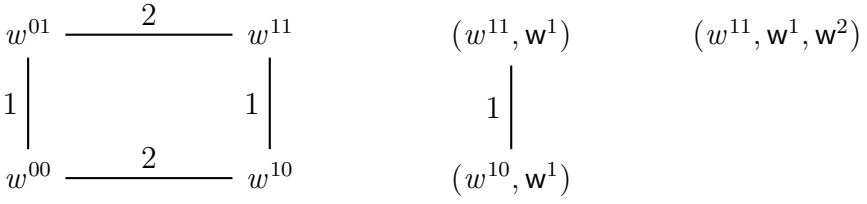
We have named the four states of the model w^{00}, w^{10}, w^{01}, and w^{11}, where the index reveals the valuation of atoms q and r (in that order), e.g. to state w^{01}, we give $\pi(w^{01}) = \{r\}$. Agent 1 cannot distinguish states w^{00}, w^{01}, therefore they are linked and the link is labelled with a '1', i.e. $w^{00} \sim_1 w^{01}$. Similar for agent 2.

Suppose we want to check the truth of formula $[q][r](q \wedge r) \wedge [r]K_1 r$ in state w^{11} of the above model. One could associate this formula with an indexed version $[^1q][^2r](q \wedge r) \wedge [^3r]K_1 r$ we proposed earlier, and the action variables $\mathsf{w}^1, \mathsf{w}^2$, and w^3 represent the three different announcements in this formula. Note that the first and second announcement r are named differently. The protocol $\textsc{prot}([q][r](q \wedge r) \wedge [r]K_1 r)$ is $\{\mathsf{w}^1\mathsf{w}^2, \mathsf{w}^3\}$.

We now apply the procedure introduced in Definition 5.10, and construct the forest model $\textsc{f}(M_{init}, \textsc{prot}([q][r](q \wedge r) \wedge [r]K_1 r))$ as follows.

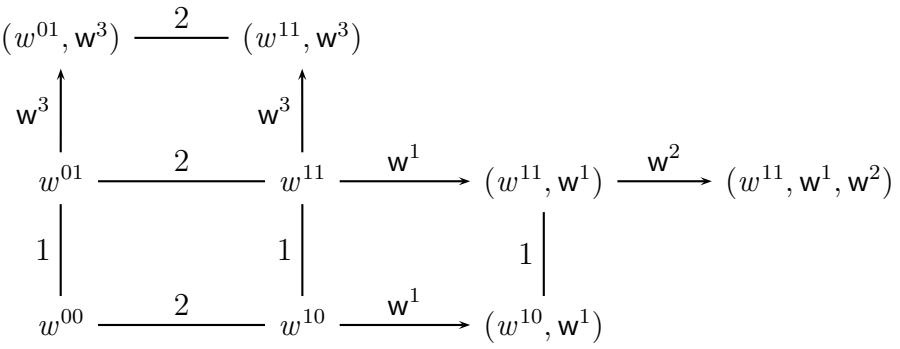
First, consider q. An announcement of q results in a new model, M_1 with two states (w^{11}, w^1) and (w^{10}, w^1). In the resulting model, agent 1 is still uncertain about r, but agent 2 now knows the value of q. After the announcement of q, given $[q][r](q \wedge r)$, atom r is subsequently announced, resulting in another state model M_2, which consists a single state $(w^{11}, \mathsf{w}^1, \mathsf{w}^2)$. In this model, both agents know that q and r are true. Now we consider the second r. It is announced in the initial model and results in a third model M_3 with two states (w^{01}, w^3) and (w^{11}, w^3). In this model, agent 2 is still uncertain about q, but agent 1 knows whether q. Depicting all three announcements at the same time, we get





where M_1 is in column 3, and M_2 is in column 4 and M_3 is in row 1 (counting from top to bottom).

$\mathrm{F}(M_{init}, \mathsf{w}^1\mathsf{w}^2)$ is depicted in the row 2 and 3 with seven states in total, and $\mathrm{F}(M_{init}, \mathsf{w}^3)$ is depicted in the column 1 and 2 with six states in total. Now $\mathrm{F}(M_{init}, \mathsf{w}^1\mathsf{w}^2) \uplus \mathrm{F}(M_{init}, \mathsf{w}^3)$ is the merge of above two forests which have common states exactly the same as the states in the initial model M_{init}.

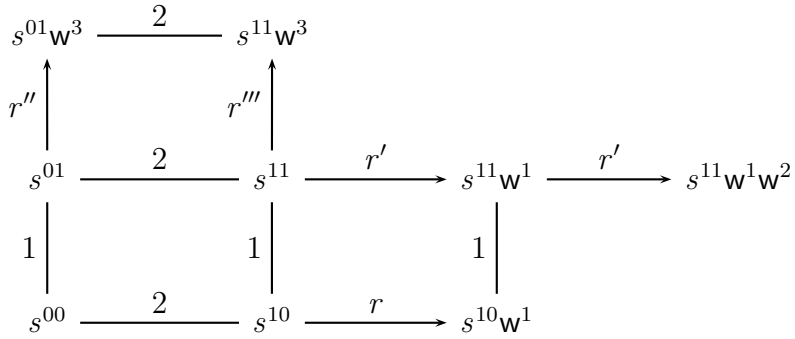


We now associate an action-based interpreted system with the forest model just given, following Definition 5.11. The above forest model consists of four trees with the roots w^{11}, w^{00}, w^{01}, and w^{10}, and five states that have no action successors: (w^{10}, w^1), $(w^{11}, \mathsf{w}^1, \mathsf{w}^2)$, (w^{01}, w^3), and (w^{11}, w^3) and w^{00}.

The global state $(w^{10}, \{w^{10}, w^{11}\}, \{w^{00}, w^{10}\})$ is associated with the state w^{10}, and the global state $(w^{10}, \{w^{10}, w^{11}\}, \{w^{10}\}, \mathsf{w}^1)$, (in other words: $(w^{10}, \{(w^{10}, \mathsf{w}^1), (w^{11}, \mathsf{w}^1)\}, \{(w^{10}, \mathsf{w}^1)\})$), is associated with the state (w^{10}, w^1) in the forest model, etc. Write s^{10} for the former global state and $s^{10}\mathsf{w}^1$ for the latter. The accessibility relations for agent 1 and 2 remain the same. Instead of action-labelled transitions we now have runs connecting the global states. There are five runs, (arbitrarily) named as

$$
\begin{array}{ll}
r & (s^{10}, s^{10}\mathsf{w}^1) \\
r' & (s^{11}, s^{11}\mathsf{w}^1, s^{11}\mathsf{w}^1\mathsf{w}^2) \\
r'' & (s^{01}, s^{01}\mathsf{w}^3) \\
r''' & (s^{11}, s^{11}\mathsf{w}^3) \\
r'''' & (s^{00})
\end{array}
$$

This interpreted system can now be depicted as



The translation SYN of the formula $[q][r](q \wedge r) \wedge [r]K_1 r$, was, as we have already seen, $(q \rightarrow \bigcirc_{\mathsf{w}^1}(r \rightarrow \bigcirc_{\mathsf{w}^2}(q \wedge r))) \wedge (r \rightarrow \bigcirc_{\mathsf{w}^3} K_1 r)$. It is easy to verify that $M_{init}, w^{11} \models_{\mathsf{sd}} [q][r](q \wedge r) \wedge [r]K_1 r$, as well as, $\textsc{is}(M_{init}, \textsc{prot}([q][r](q \wedge r) \wedge [r]K_1 r)), s^{11} \models_{\mathsf{it}} (q \rightarrow \bigcirc_{\mathsf{w}^1}(r \rightarrow \bigcirc_{\mathsf{w}^2}(q \wedge r))) \wedge (r \rightarrow \bigcirc_{\mathsf{w}^3} K_1 r)$.

5.3.4 Theoretical results

We now show, in three steps, the equivalence

$$M, w \models_{\mathsf{sd}} \varphi \text{ iff } \textsc{sem}(M, \varphi), s_w \models_{\mathsf{it}} \textsc{syn}(\varphi).$$

The *first step* is to show that given a state model M and a PAL formula φ, the interpretation of φ over (M, w) is equivalent to its interpretation over $\textsc{f}(M, \textsc{prot}(\varphi))$ which is the forest model built from M and φ. The *second step* is to show that φ and its syntactic transformation $\textsc{syn}(\varphi)$ are equivalent when they are both interpreted over the forest model $\textsc{f}(M, \textsc{prot}(\varphi))$. The *third, last, step* is to show that the interpretation of $\textsc{syn}(\varphi)$ over an arbitrary forest model and its corresponding interpreted system are equivalent. We explain these steps in three propositions: Proposition 5.1, Proposition 5.2, and Proposition 5.3. Before doing that, we first prove a lemma about some important features of the forest models.

Lemma 5.1. *Given a state model M and* PAL *formulas φ, ψ, the following equivalences hold:*

i. $\textsc{f}(M, \textsc{prot}(\varphi \wedge \psi)), w \models_{\mathsf{fd}} \varphi \Leftrightarrow \textsc{f}(M, \textsc{prot}(\varphi)), w \models_{\mathsf{fd}} \varphi$;

ii. $\textsc{f}(M, \textsc{prot}([\varphi]\psi)), w \models_{\mathsf{fd}} \varphi \Leftrightarrow \textsc{f}(M, \textsc{prot}(\varphi)), w \models_{\mathsf{fd}} \varphi$;

iii. $\textsc{f}(M, \textsc{prot}(\varphi \wedge \psi)), w \models_{\mathsf{ft}} \textsc{syn}(\varphi) \Leftrightarrow \textsc{f}(M, \textsc{prot}(\varphi)), w \models_{\mathsf{ft}} \textsc{syn}(\varphi)$;

iv. $\mathrm{F}(M, \mathrm{PROT}([\varphi]\psi)), w \models_{\mathsf{ft}} \mathrm{SYN}(\varphi) \Leftrightarrow \mathrm{F}(M, \mathrm{PROT}(\varphi)), w \models_{\mathsf{ft}} \mathrm{SYN}(\varphi)$;

Proof. Let a state model M and PAL formulas φ, ψ be given.

Case i:

We first prove the direction $\Rightarrow$. Suppose $\mathrm{F}(M, \mathrm{PROT}(\varphi \wedge \psi)), w \models_{\mathsf{fd}} \varphi$, we have $\mathrm{F}(M, \mathrm{PROT}(\varphi) \cup \mathrm{PROT}(\psi)), w \models_{\mathsf{fd}} \varphi$, since $\mathrm{PROT}(\varphi \wedge \psi) = \mathrm{PROT}(\varphi) \cup \mathrm{PROT}(\psi)$. It follows that $\mathrm{F}(M, \mathrm{PROT}(\varphi)) \uplus \mathrm{F}(M, \mathrm{PROT}(\psi)), w \models_{\mathsf{fd}} \varphi$. Suppose the domain of $\mathrm{F}(M, \mathrm{PROT}(\varphi))$ is W_1, and that of $\mathrm{F}(M, \mathrm{PROT}(\psi))$ is W_2, then according to the forest model construction in Definition 5.10, we have $W_1 \cap W_2 = W_M$, which means that the set of common states between these two forests is exactly the set of states in model M.

There are two cases for formula φ, either it contains no action modalities, then its truth value can be solely decided by the states in M, or it contains action modalities that correspond only to the actions in forest model $\mathrm{F}(M, \mathrm{PROT}(\varphi))$, therefore its truth value can be decided solely by $\mathrm{F}(M, \mathrm{PROT}(\varphi))$. In both cases, the truth value of φ can be solely decided in $\mathrm{F}(M, \mathrm{PROT}(\varphi))$, so we have $\mathrm{F}(M, \mathrm{PROT}(\varphi)), w \models_{\mathsf{fd}} \varphi$. The direction $\Leftarrow$ follows from a reverse reasoning.

Case ii: (assume the action corresponding to $[\varphi]$ is w)

We first prove the direction $\Rightarrow$. Suppose $\mathrm{F}(M, \mathrm{PROT}([\varphi]\psi)), w \models_{\mathsf{fd}} \varphi$, we have $\mathrm{F}(M, \mathrm{PROT}(\varphi) \cup \mathsf{w}\mathrm{PROT}(\psi)), w \models_{\mathsf{fd}} \varphi$, since $\mathrm{PROT}([\varphi]\psi) = \mathrm{PROT}(\varphi) \cup \mathsf{w}\mathrm{PROT}(\psi)$. Therefore $\mathrm{F}(M, \mathrm{PROT}(\varphi)) \uplus \mathrm{F}(M, \mathsf{w}\mathrm{PROT}(\psi)), w \models_{\mathsf{fd}} \varphi$. Again we distinguish two cases of φ, and conclude that the truth value of φ is solely decided by the forest $\mathrm{F}(M, \mathrm{PROT}(\varphi))$, so $\mathrm{F}(M, \mathrm{PROT}(\varphi)), w \models_{\mathsf{fd}} \varphi$. The direction $\Leftarrow$ follows from a reverse reasoning.

Case iii:

We first show the direction $\Rightarrow$. Suppose $\mathrm{F}(M, \mathrm{PROT}(\varphi \wedge \psi)), w \models_{\mathsf{ft}} \mathrm{SYN}(\varphi)$, it follows that $\mathrm{F}(M, \mathrm{PROT}(\varphi) \cup \mathrm{PROT}(\psi)), w \models_{\mathsf{ft}} \mathrm{SYN}(\varphi)$, and therefore $\mathrm{F}(M, \mathrm{PROT}(\varphi)) \uplus \mathrm{F}(M, \mathrm{PROT}(\psi)), w \models_{\mathsf{ft}} \mathrm{SYN}(\varphi)$. We distinguish two cases of formula $\mathrm{SYN}(\varphi)$, either it contains no temporal modalities, then its truth value can be solely decided by M, or it contains temporal modalities parameterized by the actions only in forest model $\mathrm{F}(M, \mathrm{PROT}(\varphi))$, hence its value can be decided by $\mathrm{F}(M, \mathrm{PROT}(\varphi))$. In both case, the truth value of $\mathrm{SYN}(\varphi)$ can be decided by forest model $\mathrm{F}(M, \mathrm{PROT}(\varphi))$, so we have $\mathrm{F}(M, \mathrm{PROT}(\varphi)) \models_{\mathsf{ft}} \mathrm{SYN}(\varphi)$. The direction $\Leftarrow$ follows from a reverse reasoning.

Case iv:

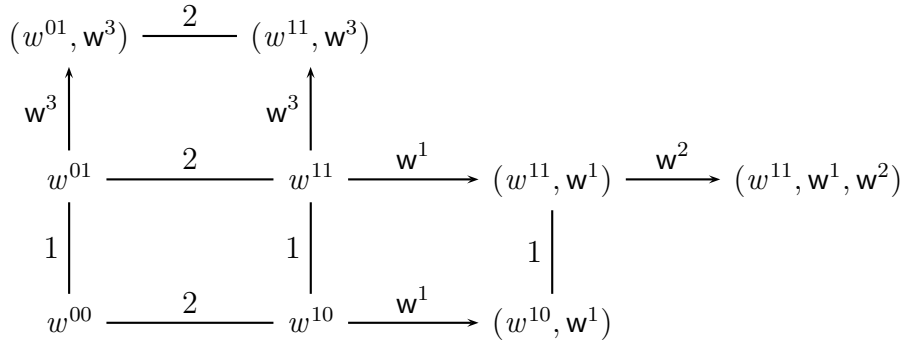
It follows from a similar reasoning as presented in *Case ii* and *Case iii*.

□

This lemma shows a special property of the forest models built from a state model and a PAL formula. In the case of formula $\varphi \wedge \psi$ and $[\varphi]\psi$, the truth value of φ is solely decided by the forest model $\mathrm{F}(M, \mathrm{PROT}(\varphi))$, which is a submodel of both $\mathrm{F}(M, \mathrm{PROT}(\varphi \wedge \psi))$ and $\mathrm{F}(M, \mathrm{PROT}([\varphi]\psi))$.

We give the following example to explain the idea more intuitively.

Example 5.4. *As we show in the previous example,* $\mathrm{F}(M_{init}, \mathrm{PROT}([q][r](q \wedge r) \wedge [r]K_1 r))$ *is as follows,*



The forest model $\mathrm{F}(M_{init}, \mathrm{PROT}([q][r](q \wedge r))$ *consists of all the states in the lower two rows, and the forest model* $\mathrm{F}(M_{init}, \mathrm{PROT}([r]K_1 r))$ *consists of all the states in the first and second columns. Clearly, the common states of these two forest models are* w^{01}, w^{11}, w^{00} *and* w^{10}*, which are exactly those states in model* M_{init}*. We evaluate the second conjunct of* $[q][r](q \wedge r) \wedge [r]K_1 r$*, namely* $[r]K_1 r$*, in the state* w^{11} *of the model* $\mathrm{F}(M_{init}, \mathrm{PROT}([q][r](q \wedge r)) \wedge K_1 r)$*. It is easy to verify that* r *is true in* w^{11} *and there is a* w^3*-successor* (w^{11}, w^3) *in which* $K_1 r$ *is true. Since all* w^3*-successors can only be included in the forest* $\mathrm{F}(M_{init}, \mathrm{PROT}([r]K_1 r))$ *and there are no epistemic links to the rest of the states, we conclude that the evaluation of* $[r]K_1 r$ *in the state* w^{11} *of the model* $\mathrm{F}(M_{init}, \mathrm{PROT}([r]K_1 r))$ *is the same.*

We can do a similar analysis for the evaluation of $\mathrm{SYN}([r]K_1 r))$*, i.e.* $r \to \bigcirc_{\mathsf{w}^3} K_1 r$*, in model* $\mathrm{F}(M_{init}, \mathrm{PROT}([q][r](q \wedge r))$.

Here is our first result.

Proposition 5.1. *Let* M *be a state model and* $\varphi \in \mathcal{L}_{\mathsf{PAL}}$.

$$M, w \models_{\mathsf{sd}} \varphi \quad \textit{iff} \quad \mathrm{F}(M, \mathrm{PROT}(\varphi)), w \models_{\mathsf{fd}} \varphi$$

Proof. Given a state model M, and formula φ, we follow the procedure in Definition 5.10 and build a forest model $\textsc{f}(M, \textsc{prot}(\varphi))$. Suppose that the interpretation function of M is π_1 and that of $\textsc{f}(M, \textsc{prot}(\varphi))$ is π_2. We do an induction on the structure of φ.

Case q:

$M, w \models_{\mathsf{sd}} q$

$\Leftrightarrow$

$q \in \pi_1(w)$

$\Leftrightarrow$ $\pi_1(w) = \pi_2(w)$ by the construction of $\textsc{f}(M, \textsc{prot}(\varphi))$

$q \in \pi_2(w)$

$\Leftrightarrow$

$\textsc{f}(M, \textsc{prot}(q)), w \models_{\mathsf{fd}} q$

Case $\neg\psi$:

$M, w \models_{\mathsf{sd}} \neg\psi$

$\Leftrightarrow$

$M, w \not\models_{\mathsf{sd}} \psi$

$\Leftrightarrow$ By induction

$\textsc{f}(M, \textsc{prot}(\psi)), w \not\models_{\mathsf{fd}} \psi$

$\Leftrightarrow$ As $\textsc{prot}(\psi) = \textsc{prot}(\neg\psi)$

$\textsc{f}(M, \textsc{prot}(\neg\psi)), w \not\models_{\mathsf{fd}} \psi$

$\Leftrightarrow$

$\textsc{f}(M, \textsc{prot}(\neg\psi)), w \models_{\mathsf{fd}} \neg\psi$

Case $\psi_1 \wedge \psi_2$:

$M, w \models_{\mathsf{sd}} \psi_1 \wedge \psi_2$

$\Leftrightarrow$

$M, w \models_{\mathsf{sd}} \psi_1$ and $M, w \models_{\mathsf{sd}} \psi_2$

$\Leftrightarrow$ By induction

$\textsc{f}(M, \textsc{prot}(\psi_1)), w \models_{\mathsf{fd}} \psi_1$ and $\textsc{f}(M, \textsc{prot}(\psi_2)), w \models_{\mathsf{fd}} \psi_2$

$\Leftrightarrow$ By Lemma 5.1

$\textsc{f}(M, \textsc{prot}(\psi_1 \wedge \psi_2)), w \models_{\mathsf{fd}} \psi_1$ and $\textsc{f}(M, \textsc{prot}(\psi_1 \wedge \psi_2)), w \models_{\mathsf{fd}} \psi_2$

$\Leftrightarrow$

$\textsc{f}(M, \textsc{prot}(\psi_1 \wedge \psi_2)), w \models_{\mathsf{fd}} \psi_1 \wedge \psi$

Case $K_i\psi$:

$M, w \models_{\mathsf{sd}} K_i\psi$
$\Leftrightarrow$
$\forall v(w \sim_i v \Rightarrow M, v \models_{\mathsf{sd}} \psi)$
$\Leftrightarrow$ By induction
$\forall v(w \sim_i v \Rightarrow \mathrm{F}(M, \mathrm{PROT}(\psi)), v \models_{\mathsf{fd}} \psi)$
$\Leftrightarrow$ As $\mathrm{PROT}(\psi) = \mathrm{PROT}(K_i\psi)$
$\forall v(w \sim_i v \Rightarrow \mathrm{F}(M, \mathrm{PROT}(K_i\psi)), v \models_{\mathsf{fd}} \psi)$
$\Leftrightarrow$
$\mathrm{F}(M, \mathrm{PROT}(K_i\psi)), w \models_{\mathsf{fd}} K_i\psi$

Case $C_B\psi$:
$M, w \models_{\mathsf{sd}} C_B\psi$
$\Leftrightarrow$
$\forall v(w \sim_B^* v \Rightarrow M, v \models_{\mathsf{sd}} \psi)$
$\Leftrightarrow$ By induction
$\forall v(w \sim_B^* v \Rightarrow \mathrm{F}(M, \mathrm{PROT}(\psi)), v \models_{\mathsf{fd}} \psi)$
$\Leftrightarrow$ As $\mathrm{PROT}(\psi) = \mathrm{PROT}(C_B\psi)$
$\forall v(w \sim_B^* v \Rightarrow \mathrm{F}(M, \mathrm{PROT}(C_B\psi)), v \models_{\mathsf{fd}} \psi)$
$\Leftrightarrow$
$\mathrm{F}(M, \mathrm{PROT}(C_B\psi)), w \models_{\mathsf{fd}} C_B\psi$

Case $[\psi_1]\psi_2$: assume that $[\psi_1]$ corresponds to the only action w in public announcement model M,
$M, w \models_{\mathsf{sd}} [\psi_1]\psi_2$
$\Leftrightarrow$
$M, w \models_{\mathsf{sd}} \psi_1 \Rightarrow M \otimes \mathsf{M}, (w, \mathsf{w}) \models_{\mathsf{sd}} \psi_2$
$\Leftrightarrow$ By induction
$\mathrm{F}(M, \mathrm{PROT}(\psi_1)), w \models_{\mathsf{fd}} \psi_1 \Rightarrow \mathrm{F}(M \otimes \mathsf{M}, \mathrm{PROT}(\psi_2)), (w, \mathsf{w}) \models_{\mathsf{fd}} \psi_2$
$\Leftrightarrow$ By Lemma 5.1
$\mathrm{F}(M, \mathrm{PROT}([\psi_1]\psi_2)), w \models_{\mathsf{fd}} \psi_1 \Rightarrow \mathrm{F}(M \otimes \mathsf{M}, \mathrm{PROT}(\psi_2)), (w, \mathsf{w}) \models_{\mathsf{fd}} \psi_2$
$\Leftrightarrow$ By forest model construction
$\mathrm{F}(M, \mathrm{PROT}([\psi_1]\psi_2)), w \models_{\mathsf{fd}} \psi_1 \Rightarrow \mathrm{F}(M, \mathsf{w}\mathrm{PROT}(\psi_2)), (w, \mathsf{w}) \models_{\mathsf{fd}} \psi_2$
$\Leftrightarrow$
$\mathrm{F}(M, \mathrm{PROT}([\psi_1]\psi_2)), w \models_{\mathsf{fd}} \psi_1 \Rightarrow$
$\mathrm{F}(M, \mathsf{w}\mathrm{PROT}(\psi_2) \cup \mathrm{PROT}(\psi_1)), (w, \mathsf{w}) \models_{\mathsf{fd}} \psi_2$
$\Leftrightarrow$ As $\mathsf{w}\mathrm{PROT}(\psi_2) \cup \mathrm{PROT}(\psi_1) = \mathrm{PROT}([\psi_1]\psi_2)$

$\text{F}(M, \text{PROT}([\psi_1]\psi_2)), w \models_{\mathsf{fd}} \psi_1 \Rightarrow \text{F}(M, \text{PROT}([\psi_1]\psi_2)), (w, \mathsf{w}) \models_{\mathsf{fd}} \psi_2$
$\Leftrightarrow$
$\text{F}(M, \text{PROT}([\psi_1]\psi_2)), w \models_{\mathsf{fd}} \psi_1 \Rightarrow$ there is (w, w) such that $w \rightarrow_{\mathsf{w}} (w, \mathsf{w})$
and $\text{F}(M, \text{PROT}([\psi_1]\psi_2)), (w, \mathsf{w}) \models_{\mathsf{fd}} \psi_2$
$\Leftrightarrow$
$\text{F}(M, \text{PROT}([\psi_1]\psi_2)), w \models_{\mathsf{fd}} [\psi_1]\psi_2$

□

This result shows that we can either evaluate a PAL formula in a state model, or alternatively construct a 'supermodel' that already contains all future dynamic developments labelled by actions. Our formulation, relative to a formula φ to be evaluated, is slightly different from the standard purely semantic form. In Venema's chapter 'Dynamic Models in their Logical Surroundings' in [78, page 122], he presented a model construction by way of a ternary accessibility operator, based on ordinary binary epistemic accessibilities. For a description of the technique see [78], or [75, 79].

The next result says that a formula $\varphi \in \mathcal{L}_{\mathsf{PAL}}$ and its translation $\text{SYN}(\varphi) \in \mathcal{L}_{\mathsf{NTEL}}$ are equivalent when they are interpreted over the same forest model.

Proposition 5.2. *Given a state model M and a* PAL *formula φ:*

$$\text{F}(M, \text{PROT}(\varphi)), w \models_{\mathsf{fd}} \varphi \quad \textit{iff} \quad \text{F}(M, \text{PROT}(\varphi)), w \models_{\mathsf{ft}} \text{SYN}(\varphi)$$

Proof. Given a state model M, and formula φ, we follow the procedure in Definition 5.10 and build a forest model $\text{F}(M, \text{PROT}(\varphi))$. Suppose that the interpretation function of $\text{F}(M, \text{PROT}(\varphi))$ is π in both semantics $\models_{\mathsf{fd}}$ and $\models_{\mathsf{ft}}$. We do an induction on the structure of φ. The cases of q, $\neg\psi$, $K_i\psi$ and $C_B\psi$ are trivial, so we just show the following two cases.

Case $\psi_1 \wedge \psi_2$:
$\text{F}(M, \text{PROT}(\psi_1 \wedge \psi_2)), w \models_{\mathsf{fd}} \psi_1 \wedge \psi_2$
$\Leftrightarrow$
$\text{F}(M, \text{PROT}(\psi_1 \wedge \psi_2)), w \models_{\mathsf{fd}} \psi_1$ and $\text{F}(M, \text{PROT}(\psi_1 \wedge \psi_2)), w \models_{\mathsf{fd}} \psi_2$
$\Leftrightarrow$ By Lemma 5.1
$\text{F}(M, \text{PROT}(\psi_1)), w \models_{\mathsf{fd}} \psi_1$ and $\text{F}(M, \text{PROT}(\psi_2)), w \models_{\mathsf{fd}} \psi_2$
$\Leftrightarrow$ By induction
$\text{F}(M, \text{PROT}(\psi_1)), w \models_{\mathsf{ft}} \text{SYN}(\psi_1)$ and $\text{F}(M, \text{PROT}(\psi_2)), w \models_{\mathsf{ft}} \text{SYN}(\psi_2)$
$\Leftrightarrow$ By Lemma 5.1

$\textsc{f}(M, \textsc{prot}(\psi_1 \wedge \psi_2)), w \models_{\mathsf{ft}} \textsc{syn}(\psi_1)$ and
$\textsc{f}(M, \textsc{prot}(\psi_1 \wedge \psi_2)), w \models_{\mathsf{ft}} \textsc{syn}(\psi_2)$
$\Leftrightarrow$
$\textsc{f}(M, \textsc{prot}(\psi_1 \wedge \psi_2)), w \models_{\mathsf{ft}} \textsc{syn}(\psi_1) \wedge \textsc{syn}(\psi_2)$
$\Leftrightarrow$
$\textsc{f}(M, \textsc{prot}(\psi_1 \wedge \psi_2)), w \models_{\mathsf{ft}} \textsc{syn}(\psi_1 \wedge \psi_2)$

Case $[\psi_1]\psi_2$: (assume that $[\psi_1]$ corresponds to the only action w in public announcement model M)

We have to show that

$$\textsc{f}(M, \textsc{prot}([\psi_1]\psi_2)), w \models_{\mathsf{fd}} [\psi_1]\psi_2 \text{ iff } \textsc{f}(M, \textsc{prot}([\psi_1]\psi_2)), w \models_{\mathsf{ft}} \textsc{syn}([\psi_1]\psi_2).$$

In other words:

$$\textsc{f}(M, \textsc{prot}([\psi_1]\psi_2)), w \models_{\mathsf{fd}} \psi_1 \Rightarrow \textsc{f}(M, \textsc{prot}([\psi_1]\psi_2)), (w, \mathsf{w}) \models_{\mathsf{fd}} \psi_2$$
$$\Leftrightarrow$$
$$\textsc{f}(M, \textsc{prot}([\psi_1]\psi_2)), w \models_{\mathsf{ft}} \textsc{syn}(\psi_1) \Rightarrow \textsc{f}(M, \textsc{prot}([\psi_1]\psi_2)), w \models_{\mathsf{ft}} \bigcirc_{\mathsf{w}} \textsc{syn}(\psi_2)$$

First we show that both conditional parts are equivalent (i). Then we show that on the condition, both consequential parts are equivalent (ii). In the proof we use various times that $\textsc{prot}([\psi_1]\psi_2)) = \textsc{prot}(\psi_1) \cup \mathsf{w}\textsc{prot}(\psi_2)$.

(i) We show that

$$\textsc{f}(M, \textsc{prot}([\psi_1]\psi_2)), w \models_{\mathsf{fd}} \psi_1 \text{ iff } \textsc{f}(M, \textsc{prot}([\psi_1]\psi_2)), w \models_{\mathsf{ft}} \textsc{syn}(\psi_1)$$

by the following equivalence:
$\textsc{f}(M, \textsc{prot}([\psi_1]\psi_2)), w \models_{\mathsf{fd}} \psi_1$
$\Leftrightarrow$ By Lemma 5.1
$\textsc{f}(M, \textsc{prot}(\psi_1)), w \models_{\mathsf{fd}} \psi_1$
$\Leftrightarrow$ By induction
$\textsc{f}(M, \textsc{prot}(\psi_1)), w \models_{\mathsf{ft}} \textsc{syn}(\psi_1)$
$\Leftrightarrow$ By Lemma 5.1
$\textsc{f}(M, \textsc{prot}([\psi_1]\psi_2)), w \models_{\mathsf{ft}} \textsc{syn}(\psi_1)$

(*ii*) Next, we show that on condition of $\mathrm{F}(M, \mathrm{PROT}(\psi_1)), w \models_{\mathsf{fd}} \psi_1$:

$\mathrm{F}(M, \mathrm{PROT}([\psi_1]\psi_2)), (w, \mathsf{w}) \models_{\mathsf{fd}} \psi_2$ iff $\mathrm{F}(M, \mathrm{PROT}([\psi_1]\psi_2)), w \models_{\mathsf{ft}} \bigcirc_{\mathsf{w}} \mathrm{SYN}(\psi_2)$.

$\mathrm{F}(M, \mathrm{PROT}([\psi_1]\psi_2)), (w, \mathsf{w}) \models_{\mathsf{fd}} \psi_2$
$\Leftrightarrow$
$\mathrm{F}(M, \mathrm{PROT}(\psi_1) \cup \mathsf{w}\mathrm{PROT}(\psi_2)), (w, \mathsf{w}) \models_{\mathsf{fd}} \psi_2$
$\Leftrightarrow$ $(w, \mathsf{w}) \notin \mathrm{F}(M, \mathrm{PROT}(\psi_1))$
$\mathrm{F}(M, \mathsf{w}\mathrm{PROT}(\psi_2)), (w, \mathsf{w}) \models_{\mathsf{fd}} \psi_2$
$\Leftrightarrow$ By forest model construction
$\mathrm{F}(M \otimes \mathsf{M}, \mathrm{PROT}(\psi_2)), (w, \mathsf{w}) \models_{\mathsf{fd}} \psi_2$
$\Leftrightarrow$ By induction
$\mathrm{F}(M \otimes \mathsf{M}, \mathrm{PROT}(\psi_2)), (w, \mathsf{w}) \models_{\mathsf{ft}} \mathrm{SYN}(\psi_2)$
$\Leftrightarrow$ By forest model construction
$\mathrm{F}(M, \mathsf{w}\mathrm{PROT}(\psi_2)), (w, \mathsf{w}) \models_{\mathsf{ft}} \mathrm{SYN}(\psi_2)$
$\Leftrightarrow$ $(w, \mathsf{w}) \notin \mathrm{F}(M, \mathrm{PROT}(\psi_1))$
$\mathrm{F}(M, \mathrm{PROT}(\psi_1) \cup \mathsf{w}\mathrm{PROT}(\psi_2)), (w, \mathsf{w}) \models_{\mathsf{ft}} \mathrm{SYN}(\psi_2)$
$\Leftrightarrow$
$\mathrm{F}(M, \mathrm{PROT}([\psi_1]\psi_2)), (w, \mathsf{w}) \models_{\mathsf{ft}} \mathrm{SYN}(\psi_2)$
$\Leftrightarrow$
$\mathrm{F}(M, \mathrm{PROT}([\psi_1]\psi_2)), w \models_{\mathsf{ft}} \bigcirc_{\mathsf{w}} \mathrm{SYN}(\psi_2)$

□

We now turn to the third result.

Proposition 5.3. *For every* executable $\varphi \in \mathcal{L}_{\mathsf{NTEL}}$ *(i.e., a formula of the form* $\mathrm{SYN}(\psi)$ *with* $\psi \in \mathcal{L}_{\mathsf{PAL}}$*), and forest model* M*:*

$$M, w \models_{\mathsf{ft}} \varphi \quad \textit{iff} \quad \mathrm{IS}(M), (w, w^{\sim_1}, \ldots, w^{\sim_n}) \models_{\mathsf{it}} \varphi$$

Proof. Let a forest model M be given. We construct an interpreted system $\mathrm{IS}(M)$ according to the procedure in Definition 5.11. Let s_w stand for $(w, w^{\sim_1}, \ldots, w^{\sim_n})$, the interpretation function of M be π_1, and the interpretation function of $\mathrm{IS}(M)$ be π_2. We do an induction on φ. All the cases are trivial but the following case:

Case $\psi_1 \to \bigcirc_{\mathsf{w}}\psi_2$:
$M, w \models_{\mathsf{ft}} \psi_1 \to \bigcirc_{\mathsf{w}}\psi_2$
$\Leftrightarrow$

$M, w \models_{\mathsf{ft}} \psi_1 \Rightarrow M, w \models_{\mathsf{ft}} \bigcirc_{\mathsf{w}} \psi_2$
$\Leftrightarrow$ ($\sharp$) on condition of $M, w \models \psi_1$ a run exists
$M, w \models_{\mathsf{ft}} \psi_1 \Rightarrow M, (w, \mathsf{w}) \models_{\mathsf{ft}} \psi_2$
$\Leftrightarrow$ By induction
$\textsc{is}(M), s_w \models_{\mathsf{it}} \psi_1 \Rightarrow \textsc{is}(M), (s_w, \mathsf{w}) \models_{\mathsf{it}} \psi_2$
$\Leftrightarrow$ (@) a run always exists
$\textsc{is}(M), s_w \models_{\mathsf{it}} \psi_1 \Rightarrow \textsc{is}(M), s_w \models_{\mathsf{it}} \bigcirc_{\mathsf{w}} \psi_2$
$\Leftrightarrow$
$\textsc{is}(M), s_w \models_{\mathsf{it}} \psi_1 \rightarrow \bigcirc_{\mathsf{w}} \psi_2$

In step $\sharp$ of the proof, this is guaranteed by the condition $M, w \models_{\mathsf{ft}} \psi_1$: as the announcement is true, it can be executed and there is an $\rightarrow_{\mathsf{w}}$ accessible state from w. This is not guaranteed if ψ_1 is false.

In step @ of the proof the required path always exists, as runs in interpreted systems are infinite. In particular, if $s_w = (r_w, i)$, the selected (s_w, w) (i.e. $(w, (w^{\sim_1}, \mathsf{w}), \ldots, (w^{\sim_n}, \mathsf{w}))$ is of the form $(r'_w, i+1)$ where r' is equivalent to r to time i. □

We emphasise that Proposition 5.3 does not hold for arbitrary formulas in our temporal epistemic fragment, because of the observed slight but essential difference between forest models, where action sequences are finite, and corresponding interpreted systems, with infinite runs. More precisely: in case $\varphi \rightarrow \bigcirc_{\mathsf{w}} \psi$ of the proof of Proposition 5.3 the precondition φ is essential! States in forest models do not necessarily have an action successor, so that in such states all formulas of form $\bigcirc_{\mathsf{w}} \psi$ are false, whereas runs in interpreted systems keep looping after a finite meaningful prefix, e.g. $\bigcirc_{\mathsf{w}} q$ will always remain true if q is true.

We now have the main result from Propositions 5.1, 5.2, and 5.3. Note that $\textsc{sem}(M, \varphi)$ is by definition $\textsc{is}(\textsc{f}(M, \textsc{prot}(\varphi)))$.

Theorem 5.1. *Given a state model M, and a* PAL *formula φ,*

$$M, w \models_{\mathsf{sd}} \varphi \quad \textit{iff} \quad \textsc{sem}(M, \varphi), s_w \models_{\mathsf{it}} \textsc{syn}(\varphi)$$

Proof. $M, w \models_{\mathsf{sd}} \varphi$
$\Leftrightarrow$ Proposition 5.1
$\textsc{f}(M, \textsc{prot}(\varphi)), w \models_{\mathsf{fd}} \varphi$
$\Leftrightarrow$ Proposition 5.2

$\text{F}(M, \text{PROT}(\varphi)), w \models_{\mathsf{ft}} \text{SYN}(\varphi)$
$\Leftrightarrow$ Proposition 5.3
$\text{IS}(\text{F}(M, \text{PROT}(\varphi))), s_w \models_{\mathsf{it}} \text{SYN}(\varphi)$

□

5.4 Generalization

We now generalize the approach in the previous section from public announcements as in $[\varphi]\psi$ to action models as in $[\mathsf{M}, \mathsf{w}]\psi$.

Definition 5.12 ($\mathcal{L}_{\mathsf{DEL}}$ to $\mathcal{L}_{\mathsf{NTEL}}$)**.** *We define a translation* SYN *from* $\mathcal{L}_{\mathsf{DEL}}$ *to* $\mathcal{L}_{\mathsf{NTEL}}$ *as follows:*

$$\begin{array}{lcl}
\text{SYN}(q) & ::= & q \\
\text{SYN}(\varphi \wedge \psi) & ::= & \text{SYN}(\varphi) \wedge \text{SYN}(\psi) \\
\text{SYN}(\neg\varphi) & ::= & \neg\text{SYN}(\varphi) \\
\text{SYN}(K_i\varphi) & ::= & K_i\text{SYN}(\varphi) \\
\text{SYN}(C_B\varphi) & ::= & C_B\text{SYN}(\varphi) \\
\text{SYN}([\mathsf{M}, \mathsf{w}]\psi) & ::= & \text{SYN}(\mathsf{pre}(\mathsf{w})) \rightarrow \bigcirc_{\mathsf{w}}\text{SYN}(\psi)
\end{array}$$

It is easy to see that the clause for public announcement (see Definition 5.8) is a special case.

We then define the protocol of a DEL formula in a similar way as in Definition 5.9.

Definition 5.13 (Protocol of DEL formula)**.** *The* protocol of a DEL formula *is defined by induction on the formula structure. In the last clause,* $\mathsf{v}\text{PROT}(\psi) = \{\mathsf{v}\mathsf{w}^1 \dots \mathsf{w}^m \mid \mathsf{w}^1 \dots \mathsf{w}^m \in \text{PROT}(\psi)\}$*, i.e. the concatenation of* v *to all sequences in the set of* $\text{PROT}(\psi)$*.*

$$\begin{array}{lcl}
\text{PROT}(q) & ::= & \emptyset \\
\text{PROT}(\neg\varphi) & ::= & \text{PROT}(\varphi) \\
\text{PROT}(\varphi \wedge \psi) & ::= & \text{PROT}(\varphi) \cup \text{PROT}(\psi) \\
\text{PROT}(K_i\varphi) & ::= & \text{PROT}(\varphi) \\
\text{PROT}(C_B\varphi) & ::= & \text{PROT}(\varphi) \\
\text{PROT}(\text{PROT}([\mathsf{M}, \mathsf{w}]\psi)) & ::= & \bigcup_{\mathsf{v} \in \mathcal{D}(\mathsf{M})}(\text{PROT}(\mathsf{pre}(\mathsf{v}) \cup \mathsf{v}\text{PROT}(\psi))
\end{array}$$

where $\mathcal{D}(\mathsf{M})$ *is the domain of the action model, which includes the point* w*.*

The main change in this definition compared to Definition 5.9 is the last clause. Here we chose to take a union for all $\mathsf{v} \in \mathcal{D}(\mathsf{M})$ because the truth of φ might be evaluated in the result of the current state model updated with action model M. And of course Definition 5.9 is still a special case of this one, as the public announcement model has a singleton domain.

Next, we generalize Definition 5.10 in a straight-forward way.

Definition 5.14 (Generated Forest Models: the General Case)**.** *Given a state model $M = \langle W, \sim_1, ..., \sim_n, \pi\rangle$, $w \in W$, and a protocol $\mathsf{T} = \text{PROT}(\varphi)$ generated from* DEL *formula φ. The forest model $\text{F}(M, \mathsf{T})$ is defined in three steps.*

(1) *Let $\mathsf{w}^1 \cdots \mathsf{w}^m$ be a sequence of actions in protocol T, and suppose that these actions belongs to action models $\mathsf{M}_1, \mathsf{M}_2, \cdots, \mathsf{M}_m$ respectively. Let M_i be a state model $M \otimes \mathsf{M}_1 \cdots \otimes \mathsf{M}_i$, which is the result of updating M_1 to M_i subsequently on M. Then $g(M, \mathsf{w}^1 \cdots \mathsf{w}^m)$ is a forest model $M' = \langle W', \sim'_1, \ldots, \sim'_n, \{\rightarrow'_{\mathsf{w}}\}, \pi'\rangle$ such that*

- *$W' = W_M \cup W_{M_1} \cup \cdots \cup W_{M_m}$ i.e. the set of states from subsequent updates;*
- *$w \sim'_i w'$ iff $\exists M_k(w \sim_i w'$ & $\sim_i$ is the relation of agent i in $M_k)$;*
- *$w \rightarrow_{\mathsf{w}} (w, \mathsf{w})$ for any $w, (w, \mathsf{w}) \in W'$;*
- *$\pi'(w) = \pi(w)$ s.t. $\exists M_k(w \in W_{M_k}$ & π belongs to $M_k)$.*

(2) *We define a union $\uplus$ of two forest models. Given forest model $M' = \langle W', \sim'_1, \ldots, \sim'_n, \{\rightarrow'_{\mathsf{w}}\}, \pi'\rangle$, and $M'' = \langle W'', \sim''_1, \ldots, \sim''_n, \{\rightarrow''_{\mathsf{w}}\}, \pi''\rangle$,*

$$M' \uplus M'' ::= \langle W''', \sim'''_1, \ldots, \sim'''_n, \{\rightarrow'''_{\mathsf{w}}\}, \pi'''\rangle$$

where $W''' = W' \cup W''$, $\sim'''_i = \sim'_i \cup \sim''_i$ for all $i \in Ag$, $\rightarrow'''_{\mathsf{w}} = \rightarrow'_{\mathsf{w}} \cup \rightarrow''_{\mathsf{w}}$ for all w, and $\pi'''(w) = \pi'(w) \cup \pi''(w)$ for all $w \in W' \cap W''$, $\pi'''(w) = \pi'(w)$ for all $w \in W' \setminus W''$, $\pi'''(w) = \pi''(w)$ for all $w \in W'' \setminus W'$.

(3) *We have,*

$$\text{F}(M, \text{PROT}(\varphi)) ::= \uplus_{\tau \in \text{PROT}(\varphi)} g(M, \tau).$$

The transformation from the forest models to the action-based interpreted systems is the same as in Definition 5.11, as we do not put any special restrictions of forest model in that definition. The idea again is to associate every state

$(w, \mathsf{w}^1, \ldots, \mathsf{w}^m)$ in the forest model with a global state $(w, w^{\sim_1}, \ldots, w^{\sim_n}, \mathsf{w}^1, \ldots, \mathsf{w}^m)$ in $\mathcal{G}$, where the local state of agent i is $(w, \mathsf{w}^1, \ldots, \mathsf{w}^m)^{\sim_i}$, and to compile runs out of branches of the forest model.

5.4.1 Theoretical results

We now proceed to generalize the results in Section 5.3.4.

Lemma 5.2. *Given a state model M and* DEL *formulas φ, ψ, the following equivalences hold:*

i. $\mathrm{F}(M, \mathrm{PROT}(\varphi \wedge \psi)), w \models_{\mathsf{fd}} \varphi \Leftrightarrow \mathrm{F}(M, \mathrm{PROT}(\varphi)), w \models_{\mathsf{fd}} \varphi$;

ii. $\mathrm{F}(M, \mathrm{PROT}([\mathsf{M}, \mathsf{w}]\psi)), w \models_{\mathsf{fd}} \mathsf{pre}(\mathsf{w}) \Leftrightarrow \mathrm{F}(M, \mathrm{PROT}(\mathsf{pre}(\mathsf{w}))), w \models_{\mathsf{fd}} \mathsf{pre}(\mathsf{w})$;

iii. $\mathrm{F}(M, \mathrm{PROT}(\varphi \wedge \psi)), w \models_{\mathsf{ft}} \mathrm{SYN}(\mathsf{pre}(\mathsf{w})) \Leftrightarrow \mathrm{F}(M, \mathrm{PROT}(\varphi)), w \models_{\mathsf{ft}} \mathrm{SYN}(\mathsf{pre}(\mathsf{w}))$;

iv. $\mathrm{F}(M, \mathrm{PROT}([\mathsf{M}, \mathsf{w}]\psi)), w \models_{\mathsf{ft}} \mathrm{SYN}(\mathsf{pre}(\mathsf{w})) \Leftrightarrow \mathrm{F}(M, \mathrm{PROT}(\mathsf{pre}(\mathsf{w}))), w \models_{\mathsf{ft}} \mathrm{SYN}(\mathsf{pre}(\mathsf{w}))$;

Proof. Let a state model M and DEL formulas φ, ψ be given.

Case i: This follows from the same reasoning as in Lemma 5.1.

Case ii: For the direction $\Rightarrow$, suppose $\mathrm{F}(M, \mathrm{PROT}([\mathsf{M}, \mathsf{w}]\psi)), w \models_{\mathsf{fd}} \mathsf{pre}(\mathsf{w})$. We have $\mathrm{F}(M, \bigcup_{\mathsf{v} \in \mathcal{D}(\mathsf{M})}(\mathrm{PROT}(\mathsf{pre}(\mathsf{v}) \cup \mathsf{v}\mathrm{PROT}(\psi)))), w \models_{\mathsf{fd}} \mathsf{pre}(\mathsf{w})$. It is easy to see that $\mathrm{F}(M, \mathrm{PROT}(\mathsf{pre}(\mathsf{w})))$ is a sub-model of $\mathrm{F}(M, \bigcup_{\mathsf{v} \in \mathcal{D}(\mathsf{M})}(\mathrm{PROT}(\mathsf{pre}(\mathsf{v}) \cup \mathsf{v}\mathrm{PROT}(\psi))))$.

We distinguish two cases of $\mathsf{pre}(\mathsf{w})$, i.e. either $\mathsf{pre}(\mathsf{w})$ contains no action modalities, or it contains action modalities that correspond only to the actions in forest $\mathrm{F}(M, \mathrm{PROT}(\mathsf{pre}(\mathsf{w})))$. In both cases, the truth value of $\mathsf{pre}(\mathsf{w})$ is solely decided by the forest $\mathrm{F}(M, \mathrm{PROT}(\mathsf{pre}(\mathsf{w})))$, therefore we have $\mathrm{F}(M, \mathrm{PROT}(\mathsf{pre}(\mathsf{w}))), w \models_{\mathsf{fd}} \mathsf{pre}(\mathsf{w})$. The direction $\Leftarrow$ follows from a reverse reasoning.

Case iii: This follows from the same reasoning as in Lemma 5.1.

Case iv: This follows from the similar reasoning as in *Case ii.* □

Proposition 5.4. *Let M be a state model and $\varphi \in \mathcal{L}_{\mathsf{DEL}}$.*

$$M, w \models_{\mathsf{sd}} \varphi \quad \textit{iff} \quad \mathrm{F}(M, \mathrm{PROT}(\varphi)), w \models_{\mathsf{fd}} \varphi$$

Proof. Given a state model M, and formula φ, we follow the procedure in Definition 5.14 and build a forest model $\mathrm{F}(M, \mathrm{PROT}(\varphi))$. We do an induction on

the structure of φ. The cases of atomic proposition, negation, knowledge and common knowledge are essentially the same as in the proof of Proposition 5.1. We only show the case with action modality.

Case $[\mathsf{M}, \mathsf{w}]\psi$:

$M, w \models_{\mathsf{sd}} [\mathsf{M}, \mathsf{w}]\psi$

$\Leftrightarrow$

$M, w \models_{\mathsf{sd}} \mathsf{pre}(\mathsf{w}) \Rightarrow M \otimes \mathsf{M}, (w, \mathsf{w}) \models_{\mathsf{sd}} \psi$

$\Leftrightarrow$ By induction

$\mathrm{F}(M, \mathrm{PROT}(\mathsf{pre}(\mathsf{w}))), w \models_{\mathsf{fd}} \mathsf{pre}(\mathsf{w}) \Rightarrow \mathrm{F}(M \otimes \mathsf{M}, \mathrm{PROT}(\psi)), (w, \mathsf{w}) \models_{\mathsf{fd}} \psi$

$\Leftrightarrow$ By Lemma 5.2

$\mathrm{F}(M, \mathrm{PROT}([\mathsf{M}, \mathsf{w}]\psi)), w \models_{\mathsf{fd}} \mathsf{pre}(\mathsf{w}) \Rightarrow \mathrm{F}(M \otimes \mathsf{M}, \mathrm{PROT}(\psi)), (w, \mathsf{w}) \models_{\mathsf{fd}} \psi$

$\Leftrightarrow$ By forest model construction

$\mathrm{F}(M, \mathrm{PROT}([\mathsf{M}, \mathsf{w}]\psi)), w \models_{\mathsf{fd}} \mathsf{pre}(\mathsf{w}) \Rightarrow$
$\mathrm{F}(M, \cup_{\mathsf{v} \in \mathcal{D}(\mathsf{M})} \mathsf{v}\mathrm{PROT}(\psi)), (w, \mathsf{w}) \models_{\mathsf{fd}} \psi$

$\Leftrightarrow$

$\mathrm{F}(M, \mathrm{PROT}([\mathsf{M}, \mathsf{w}]\psi)), w \models_{\mathsf{fd}} \mathsf{pre}(\mathsf{w}) \Rightarrow$
$\mathrm{F}(M, \cup_{\mathsf{v} \in \mathcal{D}(\mathsf{M})} \mathsf{v}\mathrm{PROT}(\psi) \cup \mathrm{PROT}(\mathsf{pre}(\mathsf{w}))), (w, \mathsf{w}) \models_{\mathsf{fd}} \psi$

$\Leftrightarrow$ As $\cup_{\mathsf{v} \in \mathcal{D}(\mathsf{M})} \mathsf{v}\mathrm{PROT}(\psi) \cup \mathrm{PROT}(\mathsf{pre}(\mathsf{w})) = \mathrm{PROT}([\mathsf{M}, \mathsf{w}]\psi)$

$\mathrm{F}(M, \mathrm{PROT}([\mathsf{M}, \mathsf{w}]\psi)), w \models_{\mathsf{fd}} \mathsf{pre}(\mathsf{w}) \Rightarrow$
$\mathrm{F}(M, \mathrm{PROT}([\mathsf{M}, \mathsf{w}]\psi)), (w, \mathsf{w}) \models_{\mathsf{fd}} \psi$

$\Leftrightarrow$

$\mathrm{F}(M, \mathrm{PROT}([\mathsf{M}, \mathsf{w}]\psi)), w \models_{\mathsf{fd}} \mathsf{pre}(\mathsf{w}) \Rightarrow \exists (w, \mathsf{w})$ s.t. $w \rightarrow_{\mathsf{w}} (w, \mathsf{w})$
and $\mathrm{F}(M, \mathrm{PROT}([\mathsf{M}, \mathsf{w}]\psi)), (w, \mathsf{w}) \models_{\mathsf{fd}} \psi$

$\Leftrightarrow$

$\mathrm{F}(M, \mathrm{PROT}([\mathsf{M}, \mathsf{w}]\psi)), w \models_{\mathsf{fd}} [\mathsf{M}, \mathsf{w}]\psi$

□

Proposition 5.5. *Given a state model M and a* DEL *formula φ:*

$$\mathrm{F}(M, \mathrm{PROT}(\varphi)), w \models_{\mathsf{fd}} \varphi \quad \textit{iff} \quad \mathrm{F}(M, \mathrm{PROT}(\varphi)), w \models_{\mathsf{ft}} \mathrm{SYN}(\varphi)$$

Proof. Given a state model M and a DEL formula φ, we follow the procedure in Definition 5.14 and build a forest model $\mathrm{F}(M, \mathrm{PROT}(\varphi))$. We do an induction on the structure of φ. The cases of atomic proposition, negation, knowledge and common knowledge are essentially the same as in the proof of Proposition 5.2. We only show the case with action modality.

Case $[\mathsf{M},\mathsf{w}]\psi$:

We have to show that

$$\textsc{f}(M,\textsc{prot}([\mathsf{M},\mathsf{w}]\psi)),w \models_{\mathsf{fd}} [\mathsf{M},\mathsf{w}]\psi \text{ iff } \textsc{f}(M,\textsc{prot}([\mathsf{M},\mathsf{w}]\psi)),w \models_{\mathsf{ft}} \textsc{syn}([\mathsf{M},\mathsf{w}]\psi).$$

In other words:

$$\textsc{f}(M,\textsc{prot}([\mathsf{M},\mathsf{w}]\psi)),w \models_{\mathsf{fd}} \mathsf{pre}(\mathsf{w}) \Rightarrow \textsc{f}(M,\textsc{prot}([\mathsf{M},\mathsf{w}]\psi)),(w,\mathsf{w}) \models_{\mathsf{fd}} \psi$$
$$\Leftrightarrow$$
$$\textsc{f}(M,\textsc{prot}([\mathsf{M},\mathsf{w}]\psi)),w \models_{\mathsf{ft}} \textsc{syn}(\mathsf{pre}(\mathsf{w})) \Rightarrow \textsc{f}(M,\textsc{prot}([\mathsf{M},\mathsf{w}]\psi)),w \models_{\mathsf{ft}} \bigcirc_{\mathsf{w}} \textsc{syn}(\psi)$$

First we show that both conditional parts are equivalent (i). Then we show that on the condition, both consequential parts are equivalent (ii). In the proof we use various times that $\textsc{prot}([\mathsf{M},\mathsf{w}]\psi)) = \cup_{\mathsf{v}\in\mathcal{D}(\mathsf{M})}\mathsf{v}\textsc{prot}(\psi) \cup \textsc{prot}(\mathsf{pre}(\mathsf{w}))$.

(i) We show that

$$\textsc{f}(M,\textsc{prot}([\mathsf{M},\mathsf{w}]\psi)),w \models_{\mathsf{fd}} \mathsf{pre}(\mathsf{w}) \text{ iff } \textsc{f}(M,\textsc{prot}([\mathsf{M},\mathsf{w}]\psi)),w \models_{\mathsf{ft}} \textsc{syn}(\mathsf{pre}(\mathsf{w}))$$

by the following equivalence:

$\textsc{f}(M,\textsc{prot}([\mathsf{M},\mathsf{w}]\psi)),w \models_{\mathsf{fd}} \mathsf{pre}(\mathsf{w})$
$\Leftrightarrow$ By Lemma 5.2
$\textsc{f}(M,\textsc{prot}(\mathsf{pre}(\mathsf{w}))),w \models_{\mathsf{fd}} \mathsf{pre}(\mathsf{w})$
$\Leftrightarrow$ By induction
$\textsc{f}(M,\textsc{prot}(\mathsf{pre}(\mathsf{w}))),w \models_{\mathsf{ft}} \textsc{syn}(\mathsf{pre}(\mathsf{w}))$
$\Leftrightarrow$ By Lemma 5.2
$\textsc{f}(M,\textsc{prot}([\mathsf{M},\mathsf{w}]\psi)),w \models_{\mathsf{ft}} \textsc{syn}(\mathsf{pre}(\mathsf{w}))$

(ii) Next, we show that on condition of $\textsc{f}(M,\textsc{prot}(\mathsf{pre}(\mathsf{w}))),w \models_{\mathsf{fd}} \mathsf{pre}(\mathsf{w})$:

$$\textsc{f}(M,\textsc{prot}([\mathsf{M},\mathsf{w}]\psi)),(w,\mathsf{w}) \models_{\mathsf{fd}} \psi \text{ iff } \textsc{f}(M,\textsc{prot}([\mathsf{M},\mathsf{w}]\psi)),w \models_{\mathsf{ft}} \bigcirc_{\mathsf{w}}\textsc{syn}(\psi).$$

$\textsc{f}(M,\textsc{prot}([\mathsf{M},\mathsf{w}]\psi)),(w,\mathsf{w}) \models_{\mathsf{fd}} \psi$
$\Leftrightarrow$
$\textsc{f}(M,\cup_{\mathsf{v}\in\mathcal{D}(\mathsf{M})}\mathsf{v}\textsc{prot}(\psi) \cup \textsc{prot}(\mathsf{pre}(\mathsf{w}))),(w,\mathsf{w}) \models_{\mathsf{fd}} \psi$
$\Leftrightarrow$ $(w,\mathsf{w}) \notin \textsc{f}(M,\textsc{prot}(\mathsf{pre}(\mathsf{w})))$
$\textsc{f}(M,\cup_{\mathsf{v}\in\mathcal{D}(\mathsf{M})}\mathsf{v}\textsc{prot}(\psi)),(w,\mathsf{w}) \models_{\mathsf{fd}} \psi$
$\Leftrightarrow$ By forest model construction

$\mathrm{F}(M \otimes \mathsf{M}, \mathrm{PROT}(\psi)), (w, \mathsf{w}) \models_{\mathsf{fd}} \psi$
$\Leftrightarrow$ By induction
$\mathrm{F}(M \otimes \mathsf{M}, \mathrm{PROT}(\psi)), (w, \mathsf{w}) \models_{\mathsf{ft}} \mathrm{SYN}(\psi)$
$\Leftrightarrow$ By forest model construction
$\mathrm{F}(M, \cup_{\mathsf{v} \in \mathcal{D}(\mathsf{M})} \mathsf{v}\mathrm{PROT}(\psi)), (w, \mathsf{w}) \models_{\mathsf{ft}} \mathrm{SYN}(\psi)$
$\Leftrightarrow$ $(w, \mathsf{w}) \notin \mathrm{F}(M, \mathrm{PROT}(\mathsf{pre}(\mathsf{w})))$
$\mathrm{F}(M, \cup_{\mathsf{v} \in \mathcal{D}(\mathsf{M})} \mathsf{v}\mathrm{PROT}(\psi) \cup \mathrm{PROT}(\mathsf{pre}(\mathsf{w}))), (w, \mathsf{w}) \models_{\mathsf{ft}} \mathrm{SYN}(\psi)$
$\Leftrightarrow$
$\mathrm{F}(M, \mathrm{PROT}([\mathsf{M}, \mathsf{w}]\psi)), (w, \mathsf{w}) \models_{\mathsf{ft}} \mathrm{SYN}(\psi)$
$\Leftrightarrow$
$\mathrm{F}(M, \mathrm{PROT}([\mathsf{M}, \mathsf{w}]\psi)), w \models_{\mathsf{ft}} \bigcirc_{\mathsf{w}} \mathrm{SYN}(\psi)$

□

Proposition 5.6. *For every* executable $\varphi \in \mathcal{L}_{\mathsf{NTEL}}$ *(i.e., a formula of the form* $\mathrm{SYN}(\psi)$ *with* $\psi \in \mathcal{L}_{\mathsf{DEL}}$*), and forest model* M*:*

$$M, w \models_{\mathsf{ft}} \varphi \quad \textit{iff} \quad \mathrm{IS}(M), (w, w^{\sim_1}, \ldots, w^{\sim_n}) \models_{\mathsf{it}} \varphi$$

Proof. Let a forest model M be given. We construction an interpreted system $\mathrm{IS}(M)$ according to the procedure in Definition 5.11. Let s_w stand for $(w, w^{\sim_1}, \ldots, w^{\sim_n})$. We do an induction on φ. We only show the case with temporal modality as the other cases are essentially the same as in the proof of Proposition 5.3.

Case $\mathsf{pre}(\mathsf{w}) \to \bigcirc_{\mathsf{w}}\psi$:
$M, w \models_{\mathsf{ft}} \mathsf{pre}(\mathsf{w}) \to \bigcirc_{\mathsf{w}}\psi$
$\Leftrightarrow$
$M, w \models_{\mathsf{ft}} \mathsf{pre}(\mathsf{w}) \Rightarrow M, w \models_{\mathsf{ft}} \bigcirc_{\mathsf{w}} \psi$
$\Leftrightarrow$ ($\sharp$) on condition of $M, w \models \mathsf{pre}(\mathsf{w})$ a run exists
$M, w \models_{\mathsf{ft}} \mathsf{pre}(\mathsf{w}) \Rightarrow M, (w, \mathsf{w}) \models_{\mathsf{ft}} \psi$
$\Leftrightarrow$ By induction
$\mathrm{IS}(M), s_w \models_{\mathsf{it}} \mathsf{pre}(\mathsf{w}) \Rightarrow \mathrm{IS}(M), (s_w, \mathsf{w}) \models_{\mathsf{it}} \psi$
$\Leftrightarrow$ (@) a run always exists
$\mathrm{IS}(M), s_w \models_{\mathsf{it}} \mathsf{pre}(\mathsf{w}) \Rightarrow \mathrm{IS}(M), s_w \models_{\mathsf{it}} \bigcirc_{\mathsf{w}} \psi$
$\Leftrightarrow$
$\mathrm{IS}(M), s_w \models_{\mathsf{it}} \mathsf{pre}(\mathsf{w}) \to \bigcirc_{\mathsf{w}}\psi$

In step ♯ of the proof, this is guaranteed by the condition $M, w \models_{\mathsf{ft}} \mathsf{pre}(\mathsf{w})$: as the precondition of w is true, it can be executed and there is an $\rightarrow_{\mathsf{w}}$ accessible state from w. This is not guaranteed if $\mathsf{pre}(\mathsf{w})$ is false.

In step @ of the proof the required path always exists, as runs in interpreted systems are infinite. In particular, if $s_w = (r_w, i)$, the selected (s_w, w) (i.e. $(w, (w, \mathsf{w})^{\sim 1}, \ldots, (w, \mathsf{w})^{\sim n})$ is of the form $(r'_w, i+1)$ where r' is equivalent to r to time i.

□

Now the generalization of Theorem 5.1 also holds.

Theorem 5.2. *Given a state model M, and a* DEL *formula φ,*

$$M, w \models_{\mathsf{sd}} \varphi \quad \textit{iff} \quad \mathrm{SEM}(M, \varphi), s_w \models_{\mathsf{it}} \mathrm{SYN}(\varphi)$$

Proof. It directly follows from Proposition 5.4, Proposition 5.5 and 5.6. □

5.5 Summary

Theorem 5.1 and Themorem 5.2 provide a correspondence on dynamic epistemic framework and temporal epistemic framework. Given an epistemic state and a formula in dynamic epistemic logic, we constructed an action-based interpreted system relative to that epistemic state and that formula, that satisfied the translation of the formula into temporal epistemic logic, and vice versa. The construction involved the protocol implicitly present in the dynamic epistemic formula, i.e. the set of sequences of actions being executed to evaluate the formula.

The temporal epistemic model checkers mentioned [67, 25, 73] have all in common that their semantics is on interpreted systems, and that they optimize search by implementing ordered binary decision diagrams (OBDDs, see [42] for an introduction). This requires *boolean* local state variables. Our presentation so far has been on a rather 'abstract' level where local states took structured forms such as $(w, \mathsf{w}, \mathsf{v})$, thus providing a direct representation for the comparison with dynamic epistemic logics. Such abstract presentation has its value in theoretical investigations, but it is still open how it can be used in the context of MAS programs. Although the current temporal epistemic checkers do not yet support action-based interpreted systems, we believe that it is a possible direction to go.

Chapter 6

Model Checking Knowledge Dynamics

6.1 Introduction

In Chapter 4, we considered the verification of playability conditions for complete information games. In Chapter 5, we built a connection between two closely-related formalisms, Dynamic Epistemic Logic (DEL) and Temporal Epistemic Logic (TEL), in which one can represent and reason about the dynamics of the agents' knowledge. The modelling of agents' knowledge and ignorance captures the essence of incomplete information. Now we turn to the verification of incomplete information scenarios more practically, with model checking tools (also referred to as "model checkers") using DEL or TEL as underlying theories.

This chapter first discusses the role of protocols in building multi-agent systems during the model checking process (section 6.2), then gives an introduction (section 6.3) to three state-of-the-art model checkers that are going to be used: DEMO (Dynamic Epistemic MOdelling), MCK (Model Checking Knowledge) and MCMAS (Model Checking Multi-Agent Systems). The differences of modelling multi-agent systems in DEL and TEL with three model checkers are studied and compared through two detailed case studies: the *Russian Cards Problem* (section 6.4) and the *Sum And Product Problem* (section 6.5). In the first case study, we formulate and study the properties of a communication protocol that solves the Russian Cards Problem, and then verify these properties in three model checkers. In the second case study, we investigate a protocol involved with agents' knowledge and ignorance. It is specified and verified in DEMO. Then the com-

plexity of this problem in DEMO is discussed. And we also explain the inherent difficulties to use MCK and MCMAS for this case.

6.2 Model Checking and Protocols

As introduced in Chapter 1 and shown in Chapter 4, there are mainly three steps in the model checking process: (1) modelling the problem in a logical formalism that is accepted by a model checker, (2) translating the specification in natural language to its formal correspondence in the specification language of the model checker, and (3) running the actual verification with the model produced in the first step and the specification produced in the second step. The new aspect in this chapter is that we are dealing with models for incomplete information and with knowledge specifications.

In the case of DEL modelling, there are two types of models: *state models*, which represent all possible states of a multi-agent system in a particular time point and agents' uncertainties about these states, and *action models*, which represent all possible actions with preconditions and agents' uncertainties about these actions. State models will be drawn from (1), and action models will be derived in (2). In the case of TEL modelling, there is only one type of models, namely interpreted systems, which represent all possible states of a multi-agent system, agents' uncertainties of these states, and the temporal precedence relations among these states.

Depending on the specific model checker at hand, the models and specifications in steps (1) and (2) are obtained differently. The DEL model checker DEMO specifies state models and action models directly in the form of mathematical structures, while the TEL model checkers MCK and MCMAS specify interpreted systems in the form of programs, hence the dynamics of interpreted systems are generated by such programs. To be more specific, an interpreted system is built from a set of initial states and a set of protocols for agents. The protocols basically specify the set of actions that each agent can choose in each state. A protocol for an agent typically consists of a set of guarded commands: if φ then a, where φ is a propositional formula on the local states, and a is an action of that agent. In each round, the preconditions of these commands are checked, and if the precondition of a command is true, then the agent will be able to choose the corresponding action. When all agents have selected an action in a state, then a unique outcome state will be determined. Thus a transition of the

current state to the next state represents a change in the system. The restriction of φ to be propositional formulas makes these two TEL model checkers unable to build interpreted systems from protocols with knowledge preconditions.

Let us go back to DEL. There is no explicit representation of an agent's protocol in DEL, but the actions are represented in a way very close to the guarded commands. Suppose w is an atomic action in DEL, and it has a precondition pre(w). The execution of the action w in a state w of a model M is as follows: if $(M, w) \models \mathsf{pre}(\mathsf{w})$ then there is a new state (w, w). This is close to the execution of a guarded command: if φ then a. Yet, there are two main differences: (1) φ can only be a propositional formula in MCK and MCMAS, but a precondition pre(w) can be any DEL formulas in DEMO; (2) the atomic actions in DEL are either joint actions by the agents or some events happened in the environment.

So, the protocols of interpreted systems in TEL and action models in DEL are similar in that actions may have preconditions, but they also have differences. On the one hand the protocols used in TEL model checkers can have individual actions for agents, but restriction on the propositional preconditions, and on the other hand, the action models in DEL have more flexibility in preconditions but have more restrictions on agents' actions, in the sense that these actions have to be unified as one. This will be demonstrated more concretely in the second case study.

6.3 Three Model Checkers

Recently, epistemic model checkers with dynamic facilities have been developed to verify properties of various multi-agent systems. In this section, we introduce three state-of-the-art model checkers in this category: DEMO, MCK and MCMAS. This is by no means the complete list the model checkers for multi-agent systems. There are other model checkers such as VerICS [53], but they will not be discussed in this chapter.

6.3.1 Model Checker DEMO

DEMO is short for Dynamic Epistemic MOdelling, developed by J. van Eijck [94]. DEMO implements the Dynamic Epistemic Logic of [9] (see Section 2.2.4). It allows modelling of epistemic updates, formula evaluation, graphical display of models, and so on.

DEMO is based on DEL, so it must provide data structures for state models, action models and DEL formulas. Written in the functional programming language Haskell, DEMO allows users to call the pre-defined functions in Haskell. The state models and action models are represented in a very natural way. Here is a generalized definition over these two types of models in DEMO.

```
Mo [state][(state,formula)][(Agent,state,state)]
```

`Mo` is an indicator of model; [`state`] represents a list of states; [(`state`, `formula`)] represents a valuation or precondition function depending on the type of `formula`; and finally [(`Agent`, `state`, `state`)] represents accessibility relations of the agents. Note that '[]' indicates a list, which is a standard data structure in Haskell. The elements in a list is ordered, and the elements in a set is not; but in the cases of state or action models, we could just treat a list as a set.

Typically pointed models, i.e. models of the form (M, w), are used in model checking. DEMO uses a more general version, multiple pointed models, adding a set of points instead of one:

```
Pmod [state][(state,formula)][(Agent,state,state)][state]
```

The only difference between `Mo` and `Pmod` is clearly the last [`state`].

Depending on the type of '`formula`', there are two kinds of pointed models:

```
EpistM = Pmod [state][(state,[Prop])][(Agent,state,state)][state]
PoAM = Pmod [state][(state, Form)][(Agent,state,state)][state]
```

Here `EpistM` represents pointed (epistemic) state models; `PoAM` represents pointed action models; [`Prop`] is a list of atomic propositions, and `Form` is a DEL formula.

We proceed with a relevant part of the definition of formulas in DEMO.

```
Form = Top | Prop Prop | Neg Form | Conj [Form] | Disj [Form]
          | K Agent Form | CK [Agent] Form | Up PoAM Form
```

Formula `Top` stands for $\top$, `Prop Prop` for atomic propositional letters (the first occurrence of `Prop` means that the datatype is 'propositional atom', whereas the second occurrence of `Prop` is the placeholder for an actual proposition letter, such as `P 3`), `Neg` for negation, `Conj [Form]` stands for the conjunction of a list of formulas of type `Form`, similarly for `Disj`, `K Agent` stands for the individual knowledge operator for agent `Agent`, and `CK [Agent]` for the common knowledge

operator for the group of agents listed in `[Agent]`; and finally `Up PoAM Form` stands for the formula with action modal operator.

The state changes are via *update product*: this is a mechanism to produce a new state model from two given models (the current state model and the current action model). We have explained this mechanism in Section 2.2.4. In DEMO, the update operation is specified as:

```
upd :: EpistM -> PoAM -> EpistM
```

Here, `EpistM` is a pointed state and `PoAM` is a pointed action model, and the update generates a new pointed state model. We can also update with a list of pointed action models:

```
upds :: EpistM -> [PoAM] -> EpistM
```

In this chapter, the case studies can all be treated in PAL, so we just need to use public announcement models. The pointed and singleton action model for a public announcement is created by a function `public` with a precondition (the announced formula) as argument.

Finally, the following function is truth evaluation function, which takes a pointed state model and a DEL formula and return a Boolean value.

```
isTrue :: EpistM -> Form -> Bool
```

So far we have introduced the modelling and specification languages in DEMO, and we will see the use of them shortly.

6.3.2 Model Checker MCK

MCK, for 'Model Checking Knowledge', is a model checker for temporal and knowledge specifications, developed by P. Gammie and R. van der Meyden [25]. The system is written in Haskell. But different from DEMO, the input language of MCK is a declarative language, so no Haskell functions are supported in the input. To address the state explosion problem, MCK uses OBDD-based techniques.

MCK is based on TEL (see Section 2.2.3). The overall setup supposes a number of agents acting in an environment. This is modelled by an interpreted system where agents perform actions according to a protocol i.e. a set of rules. The effect of actions is to change the state of the system. Actions and the

environment may be only partially observable to agents at each instant in time. Knowledge may be based on current observations only, on current observations and clock value, and on the history of all observations and clock value. The last corresponds to synchronous perfect recall. We will use the later approach. In the temporal dimension, the specification formulas may describe the evolution of the system along a single computation, i.e. using linear-time temporal logic, or they may describe the branching structure of all possible computations, i.e using branching time or computation tree logic.

An input file consists of the following parts [25]:

- **The environment in which the agents operate**, including: the possible states of the environment, the initial states of the environment, the names of agents operating in this environment, how the agents' actions change the state of the environment, (optionally) a set of fairness conditions;

- **The protocol by which each of the named agents chooses their sequence of actions**, including: the structure of local states maintained by the agent to make such choices and record other useful information, the possible initial values of the agent's local states, and a description of what parts of the state are observable to the agent;

- **A number of specification formulas**, expressing some property of the way that the agent's knowledge evolves over time.

The Figure 6.1 shows an example of input file. The scenario is that a single agent called Bob (running the protocol "robot") operates in an environment of 8 possible positions. The environment provides noisy readings of the position to the robot via the `sensor` variable, which is declared to be an observable input to the agent. Bob does not have the complete information of the system as he could not access the `position` variable. The transitions section represents the effects of the robot's actions (Halt and skip) on the state, by means of a program using non-deterministic 'if' statements. This program is considered to run atomically in a single tick of the clock.

The specification formula `spec_spr_xn` indicates that the knowledge modality should be interpreted using synchronous perfect recall, and that the formula has the form `X n` φ, expressing that φ holds in precisely n steps after an initial state (n = 2 in this example). This specification expresses that Bob knows he is in the

```
type Pos = {0..7}
position : Pos
sensor : Pos
halted : Bool

init_cond = position == 0 /\ sensor == 0 /\ neg halted

agent Bob "robot" ( sensor )

protocol "robot" (sensor : observable Pos)
begin
    do neg (sensor >= 3) -> skip
    [] break -> <<Halt>>
    od
end

transitions
begin
    if Bob.Halt -> halted := True fi;
    if neg halted -> position := position + 1 fi;
    if True -> sensor := position - 1
    [] True -> sensor := position
    [] True -> sensor := position + 1
    fi
end

spec_spr_xn = X 2 (sensor >= 3 <=> Knows Bob position in {2..4})
```

Figure 6.1: An example of MCK input, adapted from [25]

region {2..4} exactly when the sensor reading is larger or equal than 3. Then it is up MCK to verify whether this specification is true or not.

6.3.3 Model Checker MCMAS

MCMAS, for 'Model Checking Multi-Agent Systems', is a model checker developed by F. Raimondi and A. Lomuscio [67, 65]. Similar to MCK, MCMAS uses interpreted systems as the models for multi-agent systems, and uses OBDD techniques as well; but in the specification language, MCMAS allows ATL formulas in addition to LTL and CTL formulas. MCMAS is implemented in the programming language C++. Our investigation is based on the version 0.6.0. Currently

```
Agent Smith
Lstate = {s0,s1,s2,s3};
Lgreen = {s0,s1,s2};
Action = {a1,a2,a3};
Protocol:
  s0:{a1};
  s1:{a2};
  s2:{a1,a3};
  s3:{a2,a3};
endProtocol
Ev:
  s2 if ((AnotherAgent.Action=a7);
  s3 if Lstate=s2;
endEv
endAgent
```

Figure 6.2: An example of ISPL for MCMAS

there is a new version 0.9.6[1] with improved functionalities.

The input language of MCMAS is called Interpreted Systems Programming Language (ISPL). The basic unit is an agent, starting from `Agent` and ending at `endAgent`. An example is given in Figure 6.2. Each agent description consists of the following parts: a list of local states, and a list of green local states: `Lstate` and `Lgreen`; a list of actions: `Action = {...}`; a protocol for the agent: '`Protocol ... endProtocol`', which specifies a set of actions available for the agent; and an evolution function for the agent: '`Ev ... endEv`', which specifies the change of the local state of the agent.

The initial condition of the system is specified in '`InitStates ... end InitStates`', and the execution of agents generates temporal evolutions of the system. The interpretation function is then defined in a construct '`Evaluation ... end Evaluation`' with entries like '`p if Smith.Lstate = s0`'. And the properties to be verified are specified in '`Formulae ... end Formulae`' with entries like `EG (p -> K(Smith, p))`, which means there exists a run (or computation) such that if `p` is true then agent `Smith` knows it.

[1] http://dfn.dl.sourceforge.net/sourceforge/ist-contract/mcmas-0.9.6.tar.gz

6.4 Case Study: Russian Cards Problem

This section studies the Russian Cards Problem. We first give an analysis of this problem and present a communication protocol that solves this problem in Section 6.4.1. Then we show how to model this problem and specify the safety conditions that should hold after any communications, and specify them in Public Announcement Logic (PAL), a special case of DEL, in Section 6.4.2 and in TEL in Section 6.4.4 respectively. Finally, we verify the safety conditions for the protocol given earlier, in all three model checkers, and compare the differences.

6.4.1 The Problem

> *From a pack of seven known cards two players each draw three cards and a third player gets the remaining card. How can the players with three cards openly inform each other about their cards, without the third player learning from any of their cards who holds it?*

This Russian Cards Problem originated at the Moscow Math Olympiad 2000. Call the players Anne, Bill and Cath, and the cards $0, ..., 6$, and suppose Anne holds $\{0, 1, 2\}$, Bill $\{3, 4, 5\}$, and Cath card 6. For the hand of cards $\{0, 1, 2\}$, write 012 instead, for the card deal, write 012.345.6, etc. Assume from now on that 012.345.6 is the actual card deal. All announcements must be public and truthful. There are not many things Anne can safely say. Obviously, she cannot say "I have 0 or 6," because then Cath learns that Anne has 0. But Anne can also not say "I have 0 or 3," because Anne does not know if Cath has 3 or another card, and *if* Cath had card 3, she would have learnt that Anne has card 0. But Anne can also not say "I have 0 or 1." Even though Anne holds both 0 and 1, so that she does not appear to risk that Cath eliminates either card and thus gains knowledge about single card ownership (weaker knowledge, about alternatives, is allowed), Cath knows that Anne will not say anything from which Cath may learn her cards. And thus Cath can conclude that Anne will only say "I have 0 or 1" if she actually holds both 0 and 1. And in that way Cath learns two cards at once! The apparent contradiction between Cath not knowing and Cath knowing is not really there, because these observations are about different information states: it is merely the case that announcements may induce further updates that contain yet other information.

Whenever after Anne's announcement it is (at least) not common knowledge to Anne, Bill, and Cath, that Cath remains ignorant of any of Anne's or Bill's cards, this may be informative to Cath after all. A typical example is when *Anne says that she either holds 012 or not any of those cards, after which Bill says that Cath holds card 6.* For details, see [88]. Indeed, a solution requirement is that Cath's ignorance remains public knowledge after any announcement. Such announcements are called *safe*. In next section, a safety condition will be defined precisely.

A Solution A solution to the Russian Cards Problem is a sequence of safe announcements after which it is commonly known to Anne and Bill (not necessarily including Cath) that Anne knows Bill's hand and Bill knows Anne's hand. This (instance of a) five hands protocol is a solution [88]:

> *Anne says "My hand of cards is one of 012, 034, 056, 135, 246," after which Bill says "Cath has card 6."*

Note that Bill's announcement is equivalent to "My hand of cards is one of 345, 125, 024." After this sequence, it is even publicly known that Anne knows Bill's hand and Bill knows Anne's hand. If we extend Anne's announcement with one more hand, namely 245, and if it is public knowledge that the protocols used by Anne and Bill are of finite length (so may consist of more than two announcements), then it is 'merely' common knowledge to Anne and Bill that they know each other's hand, but (disregarding further analysis) Cath considers it possible that they do not know each other's hand of cards. This is a useful security feature for Anne and Bill, as Cath plays the role of the eavesdropper. A further postcondition is that all safe announcements by Anne ensure at least one safe response from Bill, and vice versa. This recursive requirement results in a more complex condition. See [89]. We will verify this five hands protocol indeed meets safety conditions.

6.4.2 Modelling in Public Announcement Logic

We now model the Russian Cards Problem in public announcement logic. Given a stack of known cards and some players, the players blindly draw some cards from the stack. In a state where cards are dealt in that way, but where no game actions of whatever kind have been done, it is commonly known what the cards

are, that they are all different, how many cards each player holds, and that players only know their own cards. From the last it follows that two deals are *the same for an agent*, if she holds the same cards in both, and if all players hold the same number of cards in both. This induces an equivalence relation on deals.

A (epistemic) state model M_{rus} encodes the knowledge of the players Anne, Bill and Cath (a, b, c). It consists of $\binom{7}{3}\binom{4}{3}\binom{1}{1} = 140$ deals. Each deal is represented as $aaa.bbb.c$, where aaa represents three cards for Anne, bbb three cards for Bill, and c one card for Cath. For each player, access between states is induced by the equivalence above, for example, $012.345.6 \sim_a 012.346.5$ says that Anne cannot tell these two card deals apart as her hand is 012 in both. Facts about card ownership written as q_i, for 'card q is held by player i'. The valuation V_{0_a} of fact 0_a (Anne holds card 0) consists of all 60 deals where 0 occurs in Anne's hand, etc. We select 012.345.6 as the actual deal, thus the initial condition is captured by this pointed state model, $(M_{rus}, 012.345.6)$.

After a sequence of announcements that is a solution of the Russian Cards Problem, it should hold that Anne knows Bill's cards, and that Bill knows Anne's cards:

$$\begin{array}{rcl} \textbf{a\_knows\_bs} & ::= & \bigwedge_{q=0..6}(K_a q_b \vee K_a \neg q_b) \\ \textbf{b\_knows\_as} & ::= & \bigwedge_{q=0..6}(K_b q_a \vee K_b \neg q_a) \end{array}$$

Moreover, it needs to satisfy that Cath does not know any of Anne's or Bill's cards:

$$\textbf{c\_ignorant} \quad ::= \quad \bigwedge_{q=0..6}(\neg K_c q_a \wedge \neg K_c q_b)$$

We suggested in the previous section that these conditions are too weak. This can be exemplified by the observation that, e.g.,

$$M_{rus}, 012.345.6 \models [(012_a \vee (\neg 0_a \wedge \neg 1_a \wedge \neg 2_a))][\textbf{c\_ignorant}]\neg \textbf{c\_ignorant}$$

After Anne says that her hand is 012 or that she does not hold any of those cards, **c_ignorant** is true, but a further update with that (in other words: when Cath can assume that this is true) makes Cath learn some of Anne's cards, so that **c_ignorant** is false. The actually required postconditions avoiding such complications are: after every announcement of an executed protocol, it is publicly known that Cath is ignorant, and after the execution of the entire protocol it is commonly known to Anne and Bill that: Anne knows that Bill knows her hand of cards, and Bill knows that Anne knows his hand of cards. Also using that $C_{ab}(K_b\textbf{a\_knows\_bs} \wedge K_a\textbf{b\_knows\_as})$ is equivalent to

$C_{ab}(\mathbf{a\_knows\_bs} \wedge \mathbf{b\_knows\_as})$. The *correct* solution condition and the safety condition are formalised as

$$C_{ab}(\mathbf{a\_knows\_bs} \wedge \mathbf{b\_knows\_as})$$
$$C_{abc}\mathbf{c\_ignorant}$$

The solution given in Section 6.4.1 consists of the successive announcements

$$\begin{aligned} \mathbf{a\_announce} &::= 012_a \vee 034_a \vee 056_a \vee 135_a \vee 246_a \\ \mathbf{b\_announce} &::= 6_c \end{aligned}$$

To verify whether this is indeed a solution, it amounts to the DEL model checking problems in Figure 6.4.2. Note that the safety condition should hold after any announcements.

i. $M_{rus}, 012.345.6 \models [\mathbf{a\_announce}][\mathbf{b\_announce}]C_{ab}\mathbf{a\_knows\_bs}$

ii. $M_{rus}, 012.345.6 \models [\mathbf{a\_announce}][\mathbf{b\_announce}]C_{ab}\mathbf{b\_knows\_as}$

iii. $M_{rus}, 012.345.6 \models [\mathbf{a\_announce}]C_{abc}\mathbf{c\_ignorant}$

iv. $M_{rus}, 012.345.6 \models [\mathbf{a\_announce}][\mathbf{b\_announce}]C_{abc}\mathbf{c\_ignorant}$

Figure 6.3: DEL model checking for solutions of Russian Cards Problem

6.4.3 Russian Cards in DEMO

In DEMO, one is restricted to propositional letters starting with lower case p, q and r, so we cannot write, for example, 0_a for the atomic proposition that Anne holds card 0, as in Section 6.4.1. Instead, we use atoms $\{\mathtt{p}, \ldots, \mathtt{p6}, \mathtt{q}, \ldots, \mathtt{q6}, \mathtt{r}, \ldots, \mathtt{r6}\}$ to represent such atomic propositions, for example proposition `p1` stands for 'Anne holds card 1'.

An implementation of Russian Cards Problem is given in DEMO in Figure 6.4. We explain this implementation as follows.

The Models The initial state model `mrus` can be specified in a natural way. The set of card deals, i.e. `deals` in Figure 6.4, is a collection of elements in the form of $(\mathtt{d0}, \mathtt{d1}, \mathtt{d2}, \mathtt{d3}, \mathtt{d4}, \mathtt{d5}, \mathtt{d6})$ where $\mathtt{d0}, \mathtt{d1}, \mathtt{d2}$ represents three cards for agent Anne, $\mathtt{d3}, \mathtt{d4}, \mathtt{d5}$ for agent Bill and `d6` for Cath. The elements in `deals` are then associated with numbers in $[\mathtt{0..139}]$ by function `zip`,

```
module RUS
where
import DEMO

deals  = [(d0,d1,d2,d3,d4,d5,d6)|
          d0<-[0..6], d1<-[d0+1..6],d2<-[d1+1..6],
          d3<-[0..6], d4<-[d3+1..6],d5<-[d4+1..6],d6<-[0..6],
          d0/=d3, d0/=d4,d0/=d5,d0/=d6,
          d1/=d3, d1/=d4,d1/=d5,d1/=d6,
          d2/=d3, d2/=d4,d2/=d5,d2/=d6,
          d3/=d6, d4/=d6,d5/=d6]

ideals = zip [0..139] deals

mrus:: EpistM
mrus = (Pmod [0..139] val acc [0])
     where
      val=[(w,[P d0, P d1, P d2, Q d3, Q d4, Q d5, R d6])|
           (w, (d0,d1,d2,d3,d4,d5,d6))<-ideals]
      acc=[(a,w,v)|(w,(d0,d1,d2,d3,d4,d5,d6))<-ideals,
            (v,(d0',d1',d2',d3',d4',d5',d6'))<-ideals,
            d0==d0',d1==d1',d2==d2']++
          [(b,w,v)|(w,(d0,d1,d2,d3,d4,d5,d6))<-ideals,
            (v,(d0',d1',d2',d3',d4',d5',d6'))<-ideals,
            d3==d3',d4==d4',d5==d5']++
          [(c,w,v)|(w,(d0,d1,d2,d3,d4,d5,d6))<-ideals,
            (v,(d0',d1',d2',d3',d4',d5',d6'))<-ideals,
            d6==d6']

a_announce =public( K a (Disj[
            Conj[Prop (P 0), Prop (P 1), Prop (P 2)],
            Conj[Prop (P 0), Prop (P 3), Prop (P 4)],
            Conj[Prop (P 0), Prop (P 5), Prop (P 6)],
            Conj[Prop (P 1), Prop (P 3), Prop (P 5)],
            Conj[Prop (P 2), Prop (P 4), Prop (P 6)]]))

b_announce= public (K b (Prop (R 6)))

a_knows_bs = Conj[Disj[K a (Prop (Q i)), K a (Neg(Prop (Q i)))]| i<-[0..6]]

b_knows_as = Conj[Disj[K b (Prop (P i)), K b (Neg(Prop (P i)))]| i<-[0..6]]

c_ignorant = Conj[Conj[Neg (K c (Prop (P i))), Neg (K c (Prop (Q i)))]| i<-[0..6]]

checki   = isTrue mrus (Up a_announce (Up b_announce (CK [a,b] a_knows_bs)))
checkii  = isTrue mrus (Up a_announce (Up b_announce (CK [a,b] b_knows_as)))
checkiii = isTrue mrus (Up a_announce (CK [a,b,c] c_ignorant))
checkiv  = isTrue mrus (Up a_announce (Up b_announce (CK [a,b,c] c_ignorant)))

check1 = isTrue mrus (Up a_announce  b_knows_as)
check2 = isTrue mrus (Up a_announce (K b (Prop (R 6))))
check3 = isTrue mrus (Up a_announce (CK [a,b] b_knows_as))
```

Figure 6.4: Russian Cards in DEMO

```
ideals = zip [0..139] deals
```

The `ideals` represents the 140 different deals. Each deals is associated with seven propositional letters – the valuation of facts in that state. The first two deals correspond to the valuations

```
(0,[P 0,P 1,P 2,Q 3,Q 4,Q 5,R 6]),
(1,[P 0,P 1,P 2,Q 3,Q 4,Q 6,R 5])
```

The deal numbered 0 stands for actual deal 012.345.6. A pair of two integers is in the accessibility relation for an agent i, if that agent holds the same cards in both deals. Two such pairs for Anne are $(\mathtt{a},0,0),(\mathtt{a},0,1)$ where `a` is short for Anne.

Anne's public announcement **a_announce** corresponds to the following singleton action model named **a_announce**, which is produced by the function `public`.

```
public( K a (Disj[Conj[p,p1,p2],Conj[p,p3,p4],Conj[p,p5,p6],
             Conj[p1,p3,p5],Conj[p2,p4,p6]]))
```

Similarly, we have an action model **b_announce** for Bill's announcement **b_announce**.

The Specifications The postcondition that Anne knows Bill's hand of cards, **a_knows_bs**, is represented as

```
aknowsbs = Conj[ Disj[K a q,  K a (Neg q)  ],
                 Disj[K a q1, K a (Neg q1) ],
                 Disj[K a q2, K a (Neg q2) ],
                 Disj[K a q3, K a (Neg q3) ],
                 Disj[K a q4, K a (Neg q4) ],
                 Disj[K a q5, K a (Neg q5) ],
                 Disj[K a q6, K a (Neg q6) ]  ]
```

Similarly for **b_knows_as** and **c_ignorant**.

In the last part, we present the model checking problems. The first one is:

```
checki=isTrue mrus (Up a_announce (Up b_announce (CK [a,b] a_knows_bs)))
```

Here `checki` is just the name that refers to this problem, serving as a shortcut; `mrus` corresponds to the initial pointed model $(M_{rus}, 012.345.6)$; and the rest part corresponds to $[\mathbf{a\_announce}][\mathbf{b\_announce}]C_{ab}\mathbf{a\_knows\_bs}$. The `checki, checkii, checkiii` and `checkiv` correspond to four model checking problems (i), (ii), (iii) and (iv) in Figure 6.4.2 respectively. The last three, `check1, check2` and `check3`, check that "After Anne's announcement Bill knows Anne's card", "After Anne's announcement Bill knows that Cath's card is 6", and "After Bill's announcement it becomes common knowledge among Anne and Bill that Bill knows Anne's card" respectively.

These problems are verified in DEMO, and we get all confirmative results, which are desired. The performance of DEMO is compared with other model checker in Section 6.4.7.

6.4.4 Modelling in Temporal Epistemic Logic

The Russian Cards Problem is now studied in TEL. We have two main tasks: (1) model this problem as an interpreted system, and (2) specify solution conditions and safety conditions in temporal epistemic formulas.

We use the interpreted system introduced in Chapter 2, not the action-based interpreted system in Chapter 5, because the action-based interpreted system is not yet supported by MCK and MCMAS. We need to specify an interpreted system $\mathcal{I}_{rus} = (\mathcal{R}, \pi)$ with $\mathcal{R}$ a set of runs and π an interpretation function. A run consists a sequence of global states, and each global state consists of an environment state and a local state for each agent. The representation of global states and local states is different in MCK and MCMAS. In MCK, a system has a set of variables, and agents have access to a subset of these variables; an instance of these variables represents a global state, and a local state of an agent is then an instance of the variables that it can access. In MCMAS, agents' local states are atomic objects, and a global state is simply a tuple of all agents' local states. The runs are generated by agents' protocols and the system transition program in MCK, but in MCMAS runs are generated by agents' protocols and their own evolution programs. The interpretation function is specified naturally.

The solution conditions and safety conditions should be expressed as temporal epistemic formulas. As we have shown in Chapter 5, an action modality in DEL is associated with an action-labelled next-time modality in NTEL. We make a similar analogy without action labels, so $[\mathbf{a\_announce}][\mathbf{b\_announce}]C_{ab}\mathbf{a\_knows\_bs}$

is associated with $\bigcirc\bigcirc C_{ab}$**a_knows_bs**. Of course, they are not the same, as the modality [**a_announce**] certainly holds more information than just $\bigcirc$; but under certain interpreted systems, they can express the same meanings. We will discuss it in the next section.

We now proceed to specify and verify the Russian Cards Problem in MCK and MCMAS in the next two sections respectively.

6.4.5 Russian Cards in MCK

In MCK, we have to reinterpret the dynamic epistemics of Section 6.4.1 in temporal epistemic terms. An implementation is presented in `rus.mck`[2]. We now explain this implementation.

We introduce environmental variables and initialize those; we create three agents `A`, `B`, and `C` with corresponding protocols `"anne"`, `"bill"` and `"cath"`; the main part of the program specifies the (temporal) transitions, induced by card dealing and the announcements, that relate different information states for these players; finally `rus.mck` contains various properties to be verified.

A hand of cards of an agent is encoded by a list of seven booleans, for example `a_hand : Bool[7]` specifies for all of the cards 0, ..., 6 whether they are held by Anne or not, such that `anne_cards[0]` is true when Anne holds card 0, etc. Initially, such variables are set to false.

Agent A, for Anne, is created by

```
agent A "anne" (a_hand, a_announce, b_announce, stage)
```

The name of the agent is `A`. It uses protocol `"anne"`. It can interact with, and potentially observe the variables between parentheses. The first of those is, obviously, only observable by Anne, the others will reappear in the other agent definitions, as they are publicly observable. The variable `stage` is the 'clock tick'.

The `transitions` part of `rus.mck` specify what happens in different stages of the execution of the protocol. We distinguish stages (clock ticks) 0, 1, 2, and 3. In stage 0 the cards are dealt to the players, in the order 0, ..., 6. We show it up to the dealing of card 0.

```
stage == 0  ->
```

[2]Available to download: http://ac.jiruan.net/thesis

```
begin   if
          na < 3  -> begin a_hand[0]:=True; na:= na+1 end []
          nb < 3  -> begin b_hand[0]:=True; nb:= nb+1 end []
          nc == 0 -> begin c_hand[0]:=True; nc:= 1 end
        fi;
```

Variables `na`, `nb`, and `nc` are counters to record how many cards agents have, and `[]` means nondeterministic choice. In this part of the transitions, 140 different deals are created, represented as 140 different timelines.

In stage 1, Anne announces that her hands is one of 012, 034, 056, 135, and 246. This is done indirectly by executing the protocol `"anne"`, that contains a condition corresponding to these five deals, which causes the action `Announce` to be executed. This then results in the atom `a_announce` becoming true.

```
stage == 1 /\ A.Announce -> a_announce := True
```

In stage 2, Bill announces that Cath holds card 6. Alternatively, one can model that Bill announces Cath's card – whatever it is. Bill's announcement is by way of an action `B.Announce`, and results in the variable `b_announce` to become true. This is the transition to stage 3, the final stage. We can imagine that the whole system consists of 140 different runs. Whether variables `a_announce` and `b_announce` are true in stage 2 and stage 3, respectively, depends on the deal in that run.

The protocol for Anne is:

```
protocol "anne" (cards: observable Bool[7],
  a_announce: observable Bool, b_announce: observable Bool,
  stage: observable Counter)
begin
  skip; if
   ( (cards[0] /\ cards[1] /\ cards[2]) \/
     (cards[0] /\ cards[3] /\ cards[4]) \/
     (cards[0] /\ cards[5] /\ cards[6]) \/
     (cards[1] /\ cards[3] /\ cards[5]) \/
     (cards[2] /\ cards[4] /\ cards[6]) )
    -> <<Announce>>
  fi
end
```

The 'begin-end' part of this protocol specifies for each of the stages 0, 1, and 2 what happens in that stage. In stage 0 nothing happens: `skip`. In stage 1, the action `Announce` – that is, whatever is found between `<<` and `>>` – is executed. Actually, the value or instance of `cards` for Anne is `a_cards`; see above, where Anne is created. Alternatively to five actual hands, a much longer protocol creates five arbitrary hands of cards based on Anne's actual hand. Nothing is specified for stage 2: this is therefore `skip` again by default. Bill has a similar protocol but his protocol starts with `skip ; skip`, as his announcement is in stage 2. And Cath does not act at all, which carries the protocol `skip ; skip ; skip`.

The knowledge of the agents evolves with every stage, via the agents' limited access to the environment. Initially, they only observe their own hand of cards, and Anne's and Bill's public announcement is accessed by all agents. Anne cannot distinguish two states *iff* her observations are the same in those states. For example, in stage, 1 Anne cannot distinguish the timelines for deals 012.345.6 and 012.346.5, because: both have the same `a_hand` values (for all seven variables), `a_announce` is true in both cases, and `b_announce` is false is both cases. But in stage 3, Anne can distinguish these timelines, since `b_announce` is true for the former and false for the latter.

A final part of `rus.mck` lists various temporal epistemic properties to be checked. They are translated from the solution and safety conditions in Figure 6.4.2. For example, we want to translate and verify that

$$M_{rus}, 012.345.6 \models [\mathbf{a\_announce}][\mathbf{b\_announce}]\, C_{ab}\mathbf{a\_knows\_bs}.$$

The latest version of MCK does not support common knowledge operators for specifications in the perfect recall module. Therefore we verify instead that in stage 3, **a_knows_bs** is valid in the model. This corresponds to the following in DEL semantics

$$M_{rus} \otimes \mathbf{a\_announce} \otimes \mathbf{b\_announce} \models \mathbf{a\_knows\_bs}$$

which ensures that

$$M_{rus} \otimes \mathbf{a\_announce} \otimes \mathbf{b\_announce}, 012.345.6 \models C_{abc}\mathbf{a\_knows\_bs}.$$

And in this specific model

$$C_{ab}\textbf{a\_knows\_bs} \leftrightarrow C_{abc}\textbf{a\_knows\_bs}$$

is also valid.

```
spec_spr_xn = X 3 ( (a_announce /\ b_announce) =>
  (  (((Knows A b_hand[0]) \/ (Knows A neg b_hand[0]))) /\
             (...)
     (((Knows A b_hand[6]) \/ (Knows A neg b_hand[6]))) ))
```

The part `spec_spr_xn` means that we are using the perfect recall module of MCK, and `X 3` is the triple 'next state' temporal operator, counting from stage 0. Therefore, the formula bound by the operator is checked in stage 3. Similarly, other conditions for the five hands protocol are verified.

6.4.6 Russian Cards in MCMAS

We implement the Russian Cards Problem in MCMAS (see rus.mcmas[3]). In MCMAS, the global state is represented as a tuple of the local states of the agents. Let agents `Anne`, `Bill`, and `Cath` represent players, and let an agent `Env` (the environment) represent the card deal. The local state of agent Anne requires five components, that can be seen as variables; three represent her hand of cards, and two the status quo and outcome of the two announcements. The version 0.6 of MCMAS, which is being used here, does not support variables in the description of agents' local states, but the newer version 0.9.6 has addressed this issue and does support variables. Therefore we encode the variable parts in a single string. For example, one local state for Anne is `a012tf`. This means that Anne holds cards 0,1, and 2, that Anne's announcement **a_announce** has been (truthfully) made in the global state of which this local state is a component, and that Bill's announcement **b_announce** could not be made (was false) in that global state. Similarly, we have five variables for Bill, and three variables for Cath. The local state of the agent `Env` has seven variables, because it represents a card deal. An example is `e0123456`. This stands for the actual deal 012.345.6.

The information changes take the usual steps: (1) the cards are revealed to the agents, (2) Anne announces **a_announce**, and (3) Bill announces **b_announce**.

[3]Available to download: http://ac.jiruan.net/thesis

All reachable global states will be included in the next stage. An example initial global state is `(annnnn, bnnnnn, cnnn, e0123456)`; an 'n' essentially means that the agent has no information on the value of corresponding variable, modelled by giving the variable that value `n`. So, `bnnnnn` means that Bill's local state is that he does not know his cards yet (the first three n's), that Anne has not made her announcement yet (the fourth n) and that Bill has not made his announcement yet (the last n). The above global state `(annnnn, bnnnnn, cnnn, e0123456)` then transits to `(a012nn, b345nn, c6nn, e0123456)`, where each agent knows what cards it holds. Anne's **a_announce** is then made, causing the transition to `(a012tn, b345tn,c6tn, e0123456)` and **b_announce** finally results in `(a012tt, b345tt,c6tt, e0123456)` – this time, Bill's announcement is successful. These state transitions are specified in the program. For example, for agent Anne, the transition for step one is as follows; `Lstate` is the local state of (current) agent `Anne`, and `Env.Lstate` is the local state of `Env`.

```
a012nn if (Lstate=annnnn and
    ( Env.Lstate=e0123456 or Env.Lstate=e0123465  or
      Env.Lstate=e0123564 or Env.Lstate=e0124563 ));
```

The environment `Env` does not change during transitions, but this has to be made explicit as

```
e0123456 if Lstate=e0123456;
```

In the 'valuation' part of an MCMAS program we define what can be seen as (the denotation of) atomic propositions. For example, the following

```
ab_d0123456 if (Anne.Lstate=a012tt and Bill.Lstate=b345tt and
               Cath.Lstate=c6tt and Env.Lstate=e0123456);
```

is an atom that is (uniquely) true in the global state `(a012tt, b345tt,c6tt, e0123456)`. Similarly, atoms expressing card ownership such as 0_a for 'Anne holds card 0' are defined by enormous expressions starting as (and consisting of 60 alternative card deals)

```
a0 if (Env.Lstate=e0123456 or Env.Lstate=e0123465 or ...
```

Groups of agents can be named too. This is useful when checking common knowledge. For example, expression "`ABC={Anne, Bill, Cath;}`" gives the group consisting of Anne, Bill, and Cath the label `ABC`. The common knowledge formula $C_{abc}(0_a \rightarrow K_a 0_a)$ is then represented as `GCK(ABC,a0->K(Anne,a0))`. We conclude this short exposition with the postcondition C_{abc}**c_ignorant** that verifies that Cath remains ignorant after both announcements have been made – '!' stands for negation.

```
ab_d0123456 -> GCK(ABC,(
  ( !K(Cath,a0) and !K(Cath,b0) ) and
  ( !K(Cath,a1) and !K(Cath,b1) ) and
  ( !K(Cath,a2) and !K(Cath,b2) ) and
  ( !K(Cath,a3) and !K(Cath,b3) ) and
  ( !K(Cath,a4) and !K(Cath,b4) ) and
  ( !K(Cath,a5) and !K(Cath,b5) ) and
  ( !K(Cath,a6) and !K(Cath,b6) ) ));
```

6.4.7 Experimental Results and Comparison

The experiments were made with these three model checkers in two different system configurations. The first system, SYSA, is configured with GNU/Linux 2.4.30 i686, 800Mhz Pentium 4 CPU and 2GB RAM, and the second, SYSB, is configured with GNU/Linux 2.6.20 i686, 2.13Ghz Core 2 CPU and 3GB DDR RAM.

Model Checkers	SYSA	SYSB	Number of States
DEMO	9 seconds	5 seconds	163
MCK	109 seconds	50 seconds	420
MCMAS	117 seconds	52 seconds	420

Figure 6.5: Experimental Results for Russian Cards Problem

Rough performance for the implementations in previous sections, are presented in Figure 6.5. It takes around half of the time to finish the checking tasks for all three model checkers in the faster system, i.e. SYSB. This shows that the improvement of hardware does help.

These results cannot be straightforwardly interpreted as indicative of the relative performance of the model checkers, as they are based on rather different

modellings and model checking questions. One clear difference is the number of states in the models. The column "Number of States" indicates the number of states that will be produced by each model checker. The DEMO will have 163 different states, because the original state model has 140 states, and after Anne's announcement, there are 20 states left, and after Bill's announcement, there are only 3 states left. The MCK and MCMAS both have 420 states because they are based on an interpreted system which has 140 different runs, and in each run there are essentially three different time points.

Another difference is in the time measured by model checkers. The time measure for MCK and MCMAS is from the whole model checking process, i.e., both model construction and formula checking. DEMO operates on slightly different principles: first, the Haskell interpreter compiles RUS.hs and related modules DPLL and DEMO; only then, DEMO check individual formulas; we measured the combined autogeneration, compilation and checking steps.

6.5 Case Study: Sum And Product Problem

This section studies the Sum And Product Problem. We first model this problem in PAL and solve it using model checker DEMO. We then study different variations of this problem, and discuss the complexity issues involved. Finally, we discuss the inherent difficulties to specify this problem in MCK and MCMAS.

6.5.1 The Problem

The following problem, or riddle, was first stated in the Dutch-language mathematics journal *Nieuw Archief voor Wiskunde* in 1969 by H. Freudenthal [23] and subsequently solved in [24]. A translation of the original formulation is:

> A says to S and P: I have chosen two integers x, y such that $1 < x < y$ and $x + y \leq 100$. In a moment, I will inform S only of $s = x + y$, and P only of $p = xy$. These announcements remain private. You are required to determine the pair (x, y).
>
> They act as said. The following conversation now takes place:
>
> i. P says: "I do not know it."
>
> ii. S says: "I knew you didn't."

iii. P says: "I now know it."

iv. S says: "I now also know it."

Determine the pair (x, y).

The announcements by the agents appear to be about ignorance and knowledge only. But actually the agents learn numerical facts from each other's announcements. For example, the numbers cannot be 2 and 3, or any other pair of prime numbers, nor for example 2 and 4, because in all those cases Product would immediately have deduced the pair from their product. As a somewhat more complicated example, the numbers cannot be 14 and 16: if they were, their sum would be 30. This is also the sum of the prime numbers 7 and 23. But then, as in the previous example, Product (P) would have known the numbers, and therefore Sum (S)—if the sum had been 30—would have considered it possible that Product knew the numbers. But Sum said that he *knew* that Product didn't know the numbers. So the numbers cannot be 14 and 16. Sum and Product learn enough, by elimination of which we gave some examples, to be able to determine the pair of numbers: the unique solution of the problem is the pair $(4, 13)$.

The knowledge that agents have about mental states of other agents and, in particular, about the effect of communications, can be important for solving problems in multi-agent systems, both for cooperative and for competitive groups. Dynamic epistemic logic was developed to study the changes brought about by communication in such higher-order knowledge of other agent's and of group knowledge [10, 27]. The Sum And Product Problem presents a complex illustrative case of the strength of specifications in dynamic epistemic logic and of the possibilities of automated model checking, and both can also be used in real multi-agent system applications.

6.5.2 Sum And Product in Public Announcement Logic

We give a specification of the Sum And Product Problem in PAL. Modulo inessential differences, explicitly mentioned below, this specification was first suggested by Plaza in [61], and in this section we elaborate on his results.

First we need to determine the set of atomic propositions and the set of agents. In the formulation of the problem, x, y are two integers such that $1 < x < y$ and $x + y \leq 100$. Define $I ::= \{(x, y) \in \mathbb{N}^2 \mid 1 < x < y \;\; \& \;\; x + y \leq 100\}$. Consider the variable x. If its value is 3, we can represent this information as the

(truth of) the atomic proposition '$x = 3$'. Slightly more formally we can think of '$x = 3$' as a propositional letter x_3. Thus we create a (finite) set of atoms $\{x_i \mid (i,j) \in I\} \cup \{y_j \mid (i,j) \in I\}$.

Concerning the agents, the role of the announcer A is to guarantee that the background knowledge for solving the problem is commonly known among Sum and Product. The announcer need not be introduced as an agent in the logical modelling of the system. That leaves $\{S, P\}$ as the set of agents. Agents S and P will also be referred to as Sum and Product, respectively.

The proposition 'Sum knows that the numbers are 4 and 13' is represented as $K_S(x_4 \wedge y_{13})$. The proposition 'Sum knows the (pair of) numbers' is described as $K_S(x,y) ::= \bigvee_{(i,j)\in I} K_S(x_i \wedge y_j)$. Similarly, 'Product knows the numbers' is represented by $K_P(x,y) ::= \bigvee_{(i,j)\in I} K_P(x_i \wedge y_j)$. Furthermore, note that the 'knew' in announcement ii, by Sum, refers to the truth of $K_S\neg K_P(x,y)$ in the *initial* epistemic state, not in the epistemic state *resulting* from announcement i, by Product.

Because of the property that all known propositions are true ('$K_i\varphi \rightarrow \varphi$' is valid), announcement i is entailed by announcement ii. Because of that, and as Product's announcement iii is a response to Sum's ii, and Sum's iv to Product's iii, the initial announcement i by Product is superfluous in the subsequent analysis. [4] This is sufficient to formalise the announcements made towards a solution of the problem:

i. P says: "I do not know it": $\neg K_P(x,y)$

ii. S says: "I knew you didn't": $K_S\neg K_P(x,y)$

iii. P says: "I now know it": $K_P(x,y)$

iv. S says: "I now also know it": $K_S(x,y)$

We can interpret these statements on a state model $\mathcal{SP}_{(x,y)} ::= \langle I, \sim, V\rangle$ consisting of a domain of all pairs $(x,y) \in I$ (as above), with accessibility relations $\sim_S$ and $\sim_P$ such that for Sum: $(x,y) \sim_S (x',y')$ iff $x + y = x' + y'$, and for Product: $(x,y) \sim_P (x',y')$ iff $xy = x'y'$; and with valuation V such that $V_{x_i} = \{(x,y) \in I \mid x = i\}$ and $V_{y_j} = \{(x,y) \in I \mid y = j\}$.

[4] Additional to this justification that announcement i is superfluous, we cannot formalise that announcement ii *follows* announcement i in our logical language, as we cannot refer to the past. In dynamic epistemic logic *with assignment* one can indirectly model such past tense epistemic statements [101].

We can describe the solution of the problem as the truth of the statement[5]

$$\mathcal{SP}_{(x,y)}, (4, 13) \models \langle K_S \neg K_P(x, y)\rangle\langle K_P(x, y)\rangle\langle K_S(x, y)\rangle\top$$

This expresses that, if $(4, 13)$ is the initial state, then it is possible to publicly announce *ii*, *iii*, and *iv*, in that order. This statement does not express that $(4, 13)$ is the *only* solution. We can express more properly that $(4, 13)$ is the only solution (and from here on our observations go beyond [61] again) as the model validity

$$\mathcal{SP}_{(x,y)} \models [K_S \neg K_P(x, y)][K_P(x, y)][K_S(x, y)](x_4 \wedge y_{13})$$

This expresses that in all *other points* of the model than $(4, 13)$ the sequence of three announcements cannot be truthfully made. Technically this relates to the well-known modal property that formulas of form $\Box\varphi$ are true for *all* φ in all worlds where there are no accessible worlds. If all announcements are true, three consecutive epistemic state transformations result in a final epistemic state (M, w) (namely $(\mathcal{SP}_{(x,y)}|ii|iii|iv, w)$) wherein $x_4 \wedge y_{13}$ should hold. Clearly, this is only the case when $w = (4, 13)$. In all other states of the domain I of model $\mathcal{SP}_{(x,y)}$, at least one announcement cannot be truthfully made; but that means that *any* postcondition of the dynamic 'necessity-type' modal operator corresponding to that announcement, even 'false', is true in that state.

For example, we observed that in state $(7, 23)$ Product would know the numbers (as they are both prime). Therefore, $\neg K_P(x, y)$ is false in $(\mathcal{SP}_{(x,y)}, (7, 23))$, and therefore $K_S \neg K_P(x, y)$ (announcement *ii*) is also false in $(\mathcal{SP}_{(x,y)}, (7, 23))$. The semantics gives us

$$\mathcal{SP}_{(x,y)}, (7, 23) \models [K_S \neg K_P(x, y)](\ [K_P(x, y)][K_S(x, y)](x_4 \wedge y_{13}) \ )$$

and $\mathcal{SP}_{(x,y)}, (7, 23) \models [K_S \neg K_P(x, y)]\bot$.

6.5.3 Sum And Product in DEMO

We implement the Sum And Product Problem in DEMO (see Figure 6.6) and show how the implementation finds the unique solution $(4, 13)$.

[5]Modality $\langle\varphi\rangle$ is the dual of $[\varphi]$.

```
module SNP
where
import DEMO

upb = 100
pairs = [(x,y)|x<-[2..100], y<-[2..100], x<y, x+y<=upb]
numpairs = llength(pairs)
llength [] =0
llength (x:xs) = 1+ llength xs
ipairs = zip [0..numpairs-1] pairs

msnp :: EpistM
msnp = (Pmod [0..numpairs-1] val acc [0..numpairs-1])
 where
 val = [(w,[P x, Q y]) | (w,(x,y))<- ipairs]
 acc = [(a,w,v)| (w,(x1,y1))<-ipairs, (v,(x2,y2))<-ipairs, x1+y1==x2+y2 ]++
       [(b,w,v)| (w,(x1,y1))<-ipairs, (v,(x2,y2))<-ipairs, x1*y1==x2*y2 ]

fmrs1e = K a (Conj [Disj[Neg (Conj [Prop (P x),Prop (Q y)]),
                Neg (K b (Conj [Prop (P x),Prop (Q y)]))]| (x,y)<-pairs])
amrs1e = public (fmrs1e)
fmrp2e = Conj [(Disj[Neg (Conj [Prop (P x),Prop (Q y)]),
             K b (Conj [Prop (P x),Prop (Q y)]) ] )|(x,y)<-pairs]
amrp2e = public (fmrp2e)
fmrs3e = Conj [(Disj[Neg (Conj [Prop (P x),Prop (Q y)]),
             K a (Conj [Prop (P x),Prop (Q y)]) ] )|(x,y)<-pairs]
amrs3e = public (fmrs3e)
solutione = showM (upds msnp [amrs1e, amrp2e, amrs3e])

fmrs1 = K a (Neg (Disj [ (K b (Conj [Prop (P x),Prop (Q y)]))| (x,y)<-pairs]))
amrs1 = public (fmrs1)
fmrp2 = Disj [K b (Conj [Prop (P x),Prop (Q y)])|(x,y)<-pairs]
amrp2 = public (fmrp2)
fmrs3 = Disj [K a (Conj [Prop (P x),Prop (Q y)])|(x,y)<-pairs]
amrs3 = public (fmrs3)
solution = showM (upds msnp [amrs1, amrp2, amrs3])
```

Figure 6.6: The DEMO program `SNP.hs`. The last part, starting from `fmrs1`, implements a less efficient variant.

Representing the state models

The set $I ::= \{(x, y) \in \mathbb{N}^2 \mid 1 < x < y \ \& \ x + y \leq 100\}$ is realized in DEMO as

```
upb = 100
pairs = [(x,y)| x<-[2..100], y<-[2..100], x<y, x+y<=upb]
```

`upb` is the maximal sum considered, in this case `upb=100`; `pairs` is a *list* of pairs: a list is a standard data structure in Haskell, unlike a set. Thus, { and } are replaced by [and], $\in$ is replaced by `<-`, and instead of I we name it `pairs`. A pair such as `(4,18)` is not a proper name for a domain element. In DEMO, natural numbers are such proper names. Therefore, we associate each element in `pairs` with a natural number and make a new list.

```
ipairs = zip [0..numpairs-1] pairs
```

Here, `numpairs` is the number of elements in `pairs`, and the function `zip` pairs the i-th element in `[0..numpairs-1]` with the i-th element in `pairs`, and makes that the i-th element of `ipairs`. For example, the first element in `ipairs` is `(0,(2,3))`. The initial model of the Sum And Product Problem is

```
msnp = (Pmod [0..numpairs-1] val acc [0..numpairs-1])
 where
 val = [(w,[P x, Q y]) | (w,(x,y))<- ipairs]
 acc = [(a,w,v)|(w,(x1,y1))<-ipairs, (v,(x2,y2))<-ipairs, x1+y1==x2+y2 ]++
       [(b,w,v)|(w,(x1,y1))<-ipairs, (v,(x2,y2))<-ipairs, x1*y1==x2*y2 ]
```

Here, `msnp` is a pointed state model, that consists of a domain `[0..numpairs-1]`, a valuation function `val`, an accessibility relation function `acc`, and `[0..numpairs-1]` points. As the points of the model are the entire domain, we may think of this initial epistemic state as the state model underlying it.

The valuation function `val` maps each state in the domain to the subset of atoms that are true in that state. This is different from Section 2.2.4, where the valuation V was defined as a function mapping each atom to the set of states where it is true. The correspondence $q \in \texttt{val}(w)$ iff $w \in V(q)$ is elementary. An element `(w,[P x, Q y])` in `val` means that in state `w`, atoms `P x` and `Q y` are true. For example, given that `(0,(2,3))` is in `ipairs`, `P 2` and `Q 3` are true in state `0`, where `P 2` stands for 'the smaller number is 2' and `Q 3` stands for 'the larger number is 3'. These same facts were described in Section 6.5.2 by x_2 and y_3, respectively, as that gave the closest match with the original problem formulation. In DEMO, names of atoms *must* start with capital P, Q, R, but the correspondence between names will be obvious.

The function `acc` specifies the accessibility relations. Agent `a` represents Sum and agent `b` represents Product. For `(w,(x1,y1))` and `(v,(x2,y2))` in `ipairs`, if their sum is the same: `x1+y1==x2+y2`, then they cannot be distinguished by Sum: `(a,w,v)` in `acc`; and if their product is the same: `x1*y1==x2*y2`, then they cannot be distinguished by Product: `(b,w,v)` in `acc`. Function `++` is an operation merging two lists.

Representing the announcements

Sum and Product's announcements are modelled as singleton action models, generated by the announced formula (precondition) φ and the operation `public`. Consider $K_S\neg\bigvee_{(i,j)\in I} K_P(x_i \wedge y_j)$, expressing that Sum says: "I knew you didn't."

This is equivalent to $K_S \bigwedge_{(i,j)\in I} \neg K_P(x_i \wedge y_j)$. A conjunct $\neg K_P(x_i \wedge y_j)$ in that expression, for 'Product does not know that the pair is (i,j)', is equivalent to $(x_i \wedge y_j) \rightarrow \neg K_P(x_i \wedge y_j)$.[6] The latter is computationally cheaper to check in the model, than the former: in all states but (i,j) of the model, the latter requires a check on two booleans only, whereas the former requires a check *in each of those states* of Product's ignorance, that relates to his equivalence class for that state, and that typically consists of several states.

This justifies why the check on $K_S \neg \bigvee_{(i,j)\in I} K_P(x_i \wedge y_j)$ can be replaced by one on $K_S \bigwedge_{(i,j)\in I}((x_i \wedge y_j) \rightarrow \neg K_P(x_i \wedge y_j))$. Similarly, using a model validity, the check on $\bigvee_{(i,j)\in I} K_P(x_i \wedge y_j)$ (Product knows the numbers) can also be replaced, namely by a check $\bigwedge_{(i,j)\in I}((x_i \wedge y_j) \rightarrow K_P(x_i \wedge y_j))$.[7] Using these observations, and writing an implication $\varphi \rightarrow \psi$ as $\neg\varphi \vee \psi$ (because DEMO does not support implication directly), we represent the three problem announcements *ii*, *iii*, and *iv* listed on page 146 as `fmrs1e`, `fmrp2e`, and `fmrs3e`, respectively, as listed in Figure 6.6. The corresponding singleton action models are obtained by applying the function `public`, e.g. `amrs1e = public (fmrs1e)`. This is also shown in the figure. The line with `solutione` abbreviates the computation of the successive model restrictions. In other words, `(upds msnp [amrs1e, amrp2e, amrs3e])` stands for state model $\mathcal{SP}|ii|iii|iv$. The final part of Figure 6.6 encodes the less efficient version of the public announcements discussed above, e.g., `fmrs1` stands for $K_S \neg \bigvee_{(i,j)\in I} K_P(x_i \wedge y_j)$. In Section 6.5.4 we will discuss the precise computational properties of the different versions.

DEMO's interaction with the implemented model

We continue by showing a relevant part of DEMO interaction with this implementation[8].

The riddle is solved by updating the initial model `msnp` with the action models corresponding to the three successive announcements. Below, `showM (upds msnp [amrs1e, amrp2e, amrs3e])` is user input and the lines from `==> [0]` is the system response to that input.

```
*SNP> showM (upds msnp [amrs1e, amrp2e, amrs3e])
```

[6] We use the T-validity $\neg K\varphi \leftrightarrow (\varphi \rightarrow \neg K\varphi)$, that can be shown as follows: $\neg K\varphi$ iff $(\varphi \vee \neg\varphi) \rightarrow \neg K\varphi$ iff $(\varphi \rightarrow \neg K\varphi) \wedge (\neg\varphi \rightarrow \neg K\varphi)$ iff $(\varphi \rightarrow \neg K\varphi) \wedge (K\varphi \rightarrow \varphi)$ iff (in T!) $(\varphi \rightarrow \neg K\varphi)$.

[7] We now use that $\varphi \bar{\vee} \psi$—where $\bar{\vee}$ is exclusive disjunction—entails that ($K\varphi \vee K\psi$ iff $(\varphi \rightarrow K\varphi) \wedge (\psi \rightarrow K\psi)$).

[8] The full (three-page) output of this interaction can be found on, http://ac.jiruan.net/thesis

```
==> [0]
[0]
(0,[p4,q13])
(a,[[0]])
(b,[[0]])
```

The function `showM` displays a pointed state model as:

```
==> [<points>]
[<domain>]
[<valuation>]
[<accessibility relations represented as equivalence classes>]
```

The list `[p4,q13]` represents the facts `P 4` and `Q 13`, i.e., the solution pair $(4, 13)$. Sum and Product have full knowledge (their access is the identity) on this singleton domain consisting of state `0`. That this state is named `0` is not a coincidence: after each update, states are renumbered starting from 0.

For another example, `(upds msnp [amrs1e,amrp2e])` represents the model that results from Product's announcement (*iii*) "Now I know it." Part of the `showM` results for that model are

```
*SNP> showM (upds msnp [amrs1e,amrp2e])
==> [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24
(...)
(0,[p2,q9])(1,[p2,q25])(2,[p2,q27])(3,[p3,q8])(4,[p3,q32])
(5,[p3,q38])(6,[p4,q7])(7,[p4,q13])(8,[p4,q19])(9,[p4,q23])
(...)
(a,[[0,3,6],[1,9,14,23,27,32,37,44,50],[2,10,17,24,28,38,45,46,51],[4
,11,18,29,33,39,47,55,60,65],[5,12,25,35,41,48,52,56,57,62,67,70,73],
[7],[8,22,36],[13,20,26,42,53,58,63,68,71,74,76,79,81],[15,19,30,34,4
0,61,66],[16,21,31,43,49,54,59,64,69,72,75,77,78,80,82,83,84,85]])
(b,[[0],[1],[2],[3],[4],[5],[6],[7],[8],[9],[10],[11],[12],[13],[14],
(...)
```

After two announcements, 86 pairs (x, y) remain possible. All remaining states are renumbered, from 0 to 85, of which part is shown. Product's (`b`) access consists of singleton sets only, of which part is shown. That should be obvious, as he just announced that he knew the number pair. Sum's (`b`) equivalence class `[0,3,6]` is that for sum 11: note that `(0,[p2,q9])`, `(3,[p3,q8])`, and

`(6,[p4,q7])` occur in the shown part of the valuation. Sum's access has one singleton equivalence class, namely `[7]`. That corresponds to the state for pair $(4, 13)$: see `(7,[p4,q13])` in the valuation. Therefore, Sum can now truthfully announce to know the pair of numbers, after which the singleton final epistemic state (that was already displayed) results.

Versions of the riddle in DEMO

How versatile the model checker DEMO is, may become clear by showing how easily the program `SNP.hs` in Figure 6.6 can be adapted to accommodate different versions of the riddle. The least upper bound for the Freudenthal version is 65. This we can check by replacing `upb = 100` in the program `SNP.hs` by `upb = 65`. More precisely we then also have to run the program for `upb = 64` after which it will appear that the model computed in `solutione = showM (upds msnp [amrs1e, amrp2e, amrs3e])` is now empty. The McCarthy version of the riddle can be checked by replacing

```
pairs = [(x,y)|x<-[2..100], y<-[2..100], x<y, x+y<=upb]
```

by

```
pairs = [(x,y)|x<-[2..100], y<-[2..100], x<=y, x<=upb, y<=upb]
```

Of course, the additions `x<=upb, y<=upb` are now superfluous, but, for another example, it is also easy to check that the least upper bound for which the McCarthy version has a solution (and now we can *not* remove `x<=upb, y<=upb`) is `upb = 62`. This we find when also changing the upper bound from 100 into 62 (and checking that `upb = 61` gives an empty model).

The interpreted system version of the model can be implemented by constructing the domain differently, namely as

```
pairs = [(v,w)|x<-[2..100],y<-[2..100],x<y,x+y<=upb,v=x+y,w=x*y]
```

and now, accessibility, later on, is simply defined as correspondence in the first argument for Sum and in the second argument for Product:

```
acc = [(a,w,v)| (w,(x1,y1))<-ipairs,(v,(x2,y2))<-ipairs,x1==x2]++
      [(b,w,v)| (w,(x1,y1))<-ipairs,(v,(x2,y2))<-ipairs,y1==y2]
```

In principle, one can also check that, when increasing the upper bound, more solutions or different solutions from $(4, 13)$ may emerge. One is then likely to run quickly into complexity problems; for that see the next section. The Plaza version with an unlimited model cannot be checked, as DEMO requires models to be finite. Also, as already mentioned, the epistemic formulas would be infinitary, which is not allowed either.[9]

Other epistemic riddles consisting of public announcements being made in some initial epistemic state, such as the Muddy Children problem, are similarly implemented by adapting the domain construction and the announcement formulas. For many examples, see [94].

6.5.4 Complexity

In this section, we analyse the complexity of finding solutions using our DEMO implementation. We first cover the theoretical boundaries, and then we present our experimental results. Computational complexity of epistemic model checking is currently a focus of the research community; for temporal epistemic model checking we refer to [84, 83, 46, 80]. These results are as such inapplicable to our setting, because even apart from the different logical (namely temporal) setting, they also focus on other modelling aspects, e.g. [46] is not based on actual epistemic models but on succinct descriptions of such models in concurrent programs, and [84] is more concerned with the complexities involved when reformulating planning problems in a model checking context.

Theoretical analysis

The Sum And Product Problem is solved by updating the initial model with a sequence of three public announcements, i.e. by `upds msnp [amrs1e, amrp2e, amrs3e]`. Each such update requires determining the set $\{w \in \mathcal{D}(M) \mid M, w \models \varphi\}$. Given a model M, a state w, and a formula φ, checking whether $M, w \models \varphi$ can be solved in time $O(|M| \times |\varphi|)$, where $|M|$ is the size of the model as measured in the size of its domain plus the number of pairs in its accessibility relations, and where $|\varphi|$ is the length of the formula φ. This result has been established by the well-known labelling method [34, 21]. This method is based on dividing φ into subformulas. One then orders all these subformulas, of which there are at

[9] Obviously, Plaza used some finite approximation of the problem, but although [61] mentions 'a program', it gives no details.

most $|\varphi|$, by increasing length. For each subformula, all states are labelled with either the formula or its negation, according to the valuation of the model and based on the results of previous steps. This is a bottom-up approach, in the sense that the labelling starts from the smallest subformulas. So it ensures that each subformula is checked only once in each state.

In DEMO v1.02, the algorithm for checking whether $M, w \models \varphi$ does not employ the bottom-up approach described above. Instead, it uses a top-down approach, starting with the formula φ and recursively checking its largest subformulas. For example, to check whether $M, w \models K_a\psi$, the algorithm checks whether $M, w' \models \psi$ for all w' such that $w \sim_a w'$, and then recursively checks the subformulas of ψ. This algorithm is $O(|M|^{|\varphi|})$, since each subformula may need to be checked $|M|$ times, and there are at most $|\varphi|$ subformulas of φ. So, in the worst case, DEMO's algorithm is quite expensive.

In practice it is less expensive, because the Haskell language and its compiler and interpreter support a cache mechanism: after evaluating a function, it caches some results in memory, for reuse. For a study on the cache mechanism in Haskell programs we refer to [54]. Since it is hard to predict what results will be cached and for how long, we cannot give an estimate how much the cache mechanism influences our experimental results. But we can still show some meaningful experimental results on the DEMO algorithm.

Experimental results

Our experimental results were based on a system configured with Windows XP, AMD CPU 3000+ (1.8Ghz), 1GB RAM. We used DEMO v1.02, and the Glasgow Haskell Compiler Interactive (GHCi) version 6.4.1, enabling the option ":set +s" to display information after evaluating each expression, including the elapsed time and number of bytes allocated.[10]

Formula efficiency In Section 6.5.3 we observed that checking a formula such as $K_S \bigwedge_{(i,j)\in I}((x_i \wedge y_j) \rightarrow \neg K_P(x_i \wedge y_j))$, `fmrs1e` in Figure 6.6, is computationally cheaper than checking its (in T) logically equivalent form $K_S \neg \bigvee_{(i,j)\in I} K_P(x_i \wedge y_j)$, `fmrs1` in Figure 6.6. Our experiments confirm this result. In Table 6.1

[10]The allocation figure is only accurate to the size of the storage manager's allocation area, because it is calculated at every garbage collection. The RAM occupation is normally 60 Mbytes for GHCi when loading the SNP.hs and DEMO modules. For evaluating a particular expression, the figure might be quite large, for example in table 1, in the case of upb=80 and fmrs1e, the result is around 3219 Mb, due to the repeated garbage collection.

	fmrs1e		**fmrs1**	
upb	time(secs)	space(bytes)	time(secs)	space(bytes)
20	0.66	13,597,832	3.13	14,430,216
40	75.30	141,967,304	954.41	172,666,424
60	714.80	858,501,572	21,389.60	1,074,424,452
80	7,232.69	3,374,852,696	not available	not available

Table 6.1: Experimental results on formula checking

upb	| msnp |	time(secs)	space(bytes)
65	23,367	1,609.03	1,325,890,048
80	43,674	7,288.08	3,150,903,744
86	54,298	10,636.03	3,905,964,300
94	70,936	20,123.02	6,048,639,068
100	85,406	34,962.38	9,047,930,216

Table 6.2: Experimental results on the trend of time-space consumption

we show the results for time and space consumption of function `upds msnp [public(fmrs1e)]` and `upds msnp [public(fmrs1)]` for different upper bounds `upb` in the initial model `msnp`: namely for `upb` 20, 40, 60 and 80. It is easy to see that checking with `fmrs1e` is substantially less costly than checking with `fmrs1` in terms of time, and slightly less costly in terms of space. We estimate that it may take more than a week to run the case `fmrs1` with `upb=80`. Cases 80 and 100 are only feasible for the more efficient form `fmrs1e`. Our next experiment is based on the more efficient `fmrs1e,fmrp2e,fmrs3e` only.

Trends for time and space consumption The smallest `upb` for which the Freudenthal version of the problem has a solution is **65**. We investigated the trend of time-space consumption between `upb`=65 and `upb`=100, as this relates to the size of the initial model `msnp`. The results for running `solutione` (see Figure 6.6) are shown in Table 6.2. In this table, | `msnp` | is the size of the model `msnp`, measured as the number of states in the domain plus the number of pairs in the accessibility relations (for Sum and for Product). The proportional increase of the figures in Table 6.2 is clearer in Table 6.3, wherein they are normalised to the case `upb=65`. Note that time consumption increases faster than space consumption when the size of the model increases.

Finally, we also investigated the computational differences between the stan-

upb	| msnp |/N	time/N	space/N
65	1	1	1
80	1.87	4.53	2.38
86	2.32	6.61	2.95
94	3.04	12.51	4.56
100	3.65	21.73	6.82

Table 6.3: Normalisation of the experimental results in Table 6.2.

dard 'smaller/larger number' modelling of the puzzle (Section 6.5.2) and the 'interpreted system' modelling of the puzzle. In this case we did not find significant results.

6.5.5 Other Model Checkers

As mentioned in the introduction of Section 6.3.1, other epistemic model checkers with dynamic features include MCK [25] and MCMAS [67]. The question is whether we could also implement this problem in those model checkers. For the latest versions of these model checkers in both case the answer appears to be 'no'.

In MCK, a state of the environment is an assignment to a set of variables declared in the environment section. These variables are usually assumed to be partially accessible to the individual agents, and agents could share some variables. The change of the state of a multi-agent system is either made by agents or the environment, in the form of changing these variables. There are two ways to make such changes. One is to send signals to the environment using the action construct by agents in conjunction with the transitions construct by the environment, which provides a way to describe how the environment variables are updated. The other is a specialised form for actions from the perspective that environment variables are shared variables, by providing read and write operations on those shared variables. In both cases, we need guarded statements to make the change. For example, a simple deterministic statement has the form:

$$\text{if } \mathit{cond} \rightarrow C \text{ [otherwise} \rightarrow C_o \text{] fi}$$

where command C is eligible for execution only if the corresponding condition *cond* evaluates to true in the current state. Otherwise, the command C_o will be executed. If we would like to model the Sum And Product Problem in MCK, the

effect of a public announcement should be recorded in a variable which is accessible to all agents. Suppose the effect of P's public announcement : "I now know it" ($K_P(x, y)$) is recorded in variable v. Then in a state just after this announcement, the variable v will be set to *True* if $K_P(x, y)$ holds in the previous state, and otherwise to *False*. Clearly, we need that statement in the above *epistemic* form, with *cond* involving knowledge checking. Unfortunately, even though in MCK we can check epistemic postconditions, the current version of MCK does not support checking epistemic formulas as preconditions, as in *cond*. Similarly to MCK, MCMAS also does not support actions with knowledge-based preconditions to transit from one global state to another global state. The difficulty of enabling knowledge-based preconditions in MCK and MCMAS might be related to the interpretation issues of knowledge-based programs in terms of interpreted systems (see [21]).

6.6 Summary

In this chapter, we first discussed the role of protocols in building multi-agent systems during the model checking process. In TEL model checking the protocols are specified in the construction of interpreted systems. In DEL model checking, the protocols are specified in DEL formulas. Then we introduced three state-of-the-art model checkers for multi-agent systems verification: DEMO, MCK and MCMAS. We studied and compared the differences of modelling multi-agent systems in DEL and TEL through two detailed case studies: the *Russian Cards Problem* and the *Sum And Product Problem*. In the first case study, we formulated the properties of a communication protocol that solves the Russian Cards Problem, and then verified these properties in three model checkers. We also showed that how dynamic epistemic requirements can be reformalised in temporal epistemic terms. In the second case study, we investigated a protocol involved with agents' knowledge and ignorance. It was specified and verified only in DEMO because only DEMO supports knowledge-based protocols. We studied the complexity of model checking with DEMO and the experimental results. Finally, we discussed the inherent difficulties to use MCK and MCMAS for this case.

Chapter 7

Looking Backward and Forward

We are coming to the end of this thesis. It is a good time to reflect upon what we have done, and give some thoughts on what we could do in the future, just like any reflexive agents would do.

7.1 Looking Backward

Looking backward, I have, with the collaboration of my colleagues, built two one-way bridges: one is from GDL to ATL, and the other is from DEL to TEL. Let me again start from the motivations and then give a summary of our main contributions.

Why logic? Why did I get interested in the logical formalisms in the first place? Apart from the reasons given in Section 1.2, I give a more personal account here. Through the years, I observed and learned that there were already many formalisms modelling multi-agent systems. I was particularly interested in logic-based approaches because I thought they provided an easy, yet profound way of understanding multi-agent systems. For instance, in the *Sum And Product Problem*, all possible combinations of two numbers are encoded in states; the agents' knowledge and ignorance can be naturally encoded in their accessibilities of these states, and expressed in succinct formulas (e.g. $K_S \neg K_p(x, y)$ for "Mr. Sum knew Mr. Product did not know the two numbers"); the solution can be computed by updating the initial state model with agents' announcements in a sequence. The actual computation might be tedious, but the logic formalisms, DEL in this case, really give us a high-level grasp of the key information flow

going through agents' interactions. Since we have powerful computers, we do not need to be bothered by the tedious computation anyway, except studying its complexity in order to avoid possible endless waiting.

Why building bridges? So why building the bridges for these particular frameworks, and why just one-way? First, we study GDL because it is a well-defined language for a serious multi-agent research area: the General Game Playing (GGP) Competition, which is supported by AAAI. Games have been an important part of different cultures. For instance, both Chess and Chinese Chess were invented to mimic the conflict scenarios to provide entertainment, as well as to develop players' strategic reasoning abilities, without them getting into bloody physical conflicts. The games also provide well-defined settings to study multi-agent interactions, as they typically consist of a set of rules, an initial configuration, and the goals for players. The GGP systems have to be more flexible than dedicated chess-playing systems like Deep Blue, and hence will have greater potential to be used in building more adaptable multi-agent systems. There are logic-based formalisms developed for reasoning about games. In particular, ATL can reason about agents' strategic abilities, and has a complete set of tools from logical theory to model checking tools. To enable these assets to be reused, we decided to investigate the possible connections between GDL and ATL, and subsequently built a concrete link between them. The link is only one-way because GDL is a language for representing games, not for reasoning about the powers of the coalitions.

Second, we study the connections between DEL and TEL because they both are capable of modelling multi-agent systems with incomplete information, yet in different ways. They are very similar in terms of modelling knowledge, but are different in dealing with the dynamics in the systems. In DEL the dynamics of the system is solely expressed in the logical language with actions, but in TEL the dynamics are realized in the semantic model and expressed in the language as well. Due to such differences, we created a variant of TEL, namely NTEL. The bridges we built, i.e. the syntactic translation SYN, and the semantic transformation SEM, are only from DEL to NTEL, and hence it can be seen as an embedding. The reason is that the temporal changes modelled in DEL correspond to a special class of temporal changes modelled in NTEL; hence in TEL.

Summary of Contributions Here we give a summary of the main contributions of this thesis made in separate chapters.

In Chapter 3, we studied the axiomatisation of games in GDL. We first formalised a propositional GDL and associated each GDL description with a game model. Then we provided a semantics of GDL rules over such game models. Next, GDL was understood as a specification language of ATL models, i.e. AATSs. Given a GDL description, on the one hand we built an AATS using a semantic transformation $\mathcal{T}_{sem}$, on the other hand, we translated the GDL description in to an axiom expressed in ATL using a syntactic translation $\mathcal{T}_{syn}$. We showed, in Theorem 3.4, that any AATS that satisfies the translated axiom is alternating-bisimilar to the AATS built from the GDL description semantically. This means that our axiomatic characterization of GDL descriptions did capture the agents' strategic powers completely. As a corollary, we proved that the complexity of model checking ATL formulas over GDL descriptions is EXPTIME-Complete (Theorem 3.5).

In Chapter 4, we studied the verification of games defined in GDL more practically. There were two main contributions. First, in Section 4.2, using ATL, we characterised a class of game properties that should be held in general: *coherence properties* ensure the game has a "sensible" interpretation; *fair playability conditions* characterise the strategic abilities of coalitions of agents. Then we proposed another class of properties that characterise different games. Second, in Section 4.3, we developed an automated tool that transforms a GDL description into an RML specification that is accepted by the ATL model checker, MOCHA. Then, in Section 4.4, we showed the feasibility of our approach by a detailed case study on the game Tic-Tac-Toe.

In Chapter 5, we studied a correspondence between DEL and NTEL, a variant of TEL. Given a state model and a formula in dynamic epistemic logic, we constructed an action-based interpreted system relative to that epistemic state and that formula by using a semantic transformation SEM, that satisfied a syntactic translation SYN of the formula into temporal epistemic logic, and vice versa. The SEM involved the protocol implicitly present in the dynamic epistemic formula, i.e. the set of sequences of actions being executed to evaluate the formula. We first showed this correspondence for PAL, a special case of DEL, in Theorem 5.1, and then generalized the case to full DEL in Theorem 5.2.

In Chapter 6, we studied model checking knowledge dynamics for multi-agent systems both in DEL and TEL. We started with a general discussion of protocol

and multi-agent system building, and introduced three model checkers DEMO, MCK and MCMAS. Next, we modelled the Russian Cards Problem both in DEL and TEL, specified the safety conditions needed for solutions, and verified such conditions in three model checkers for a specific solution of this problem. Then we modelled the Sum And Product Problem in DEL, specified it in DEMO and then found the correct solution; in addition, we studied the complexity related to DEMO model checking. In the end, we explained the difficulties of modelling the Sum And Product Problem in the current versions of MCK and MCMAS.

7.2 Looking Forward

Looking forward, I would like to give four possible directions for further research, some of which have been discussed in the relevant chapters.

- **Extending GDL for incomplete information games**
  The current GDL specification studied in Chapter 3 is only suitable for describing *complete information* games, in which agents are fully aware of the current state of the games. In games like Chess, for instance, the players know exactly what is on the game board. But there is another large class of games in which agents can only have incomplete information of the system. For example, in Poker, agents typically do not know what cards their opponents hold initially, as they can only observe their own cards. In reality, incomplete information access for agents is more ubiquitous, due to their limited observation powers or memories. So an extension of GDL for incomplete information games seems to be called for.

  Meanwhile, there has been already some progress on modelling incomplete information games using logics. In, [76] van Benthem gave a modelling of games in DEL. In [86, 43], van der Hoek, Wooldridge and Jamroga developed epistemic extensions of ATL. In [38], Herzig and Troquard took another approach based on STIT logic, which is another important temporal logic. So this topic has certainly attracted many logicians. And after we have the epistemic extension of GDL, it would be interesting to study its connections with the above logical frameworks.

- **Investigating DEL and Knowledge-based Protocols in TEL**
  In Chapter 5, we noticed that there is a relation between the protocol

generated from a DEL formula with the interpreted system built from a state model with this protocol. Such a protocol is similar to the knowledge-based protocols proposed in [21]. A further investigation on this relation will likely reveal more connections between DEL and TEL.

- **Improving Model Checkers**
  In Chapter 6, we compared three state-of-the-art model checkers. Despite the fact that DEMO had certain advantages when dealing with the Russian Cards Problem, its scalability is not very promising witnessed by the results in the Sum And Product Problem. It might be a good idea to explore the possibility to extend DEMO with OBDD techniques which are already employed in both MCK and MCMAS. As for MCK and MCMAS, we find that they have difficulties in dealing with the Sum And Product Problem because the model specification language does not support knowledge preconditions. This is related to knowledge-based protocols again, and I believe it is worthwhile to explore this connection.

- **Playing GDL Games via Model Checking**
  In Section 4.4.3, we briefly explained the possibility of playing Tic-Tac-Toe via model checking. This might be not very promising due to the EXPTIME-Complete complexity for such reasoning, but with the rapid advancement of computational powers it might be still possible to address games with not too many states. The tasks will likely be exploring the playability conditions to build agents' strategies, reducing the variables in the presentation, and improving the efficiency of the model checker MOCHA.

Bibliography

[1] C. Alchourrón, P. Gärdenfors, and D. Makinson. On the logic of theory change: partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50:510–530, 1985.

[2] R. Alur, L. de Alfaro, T. A. Henzinger, S. C. Krishnan, F. Y. C. Mang, S. Qadeer, S. K. Rajamani, and S. Taşiran. MOCHA user manual. University of Berkeley Report, 2000.

[3] R. Alur, R. Grosu, and B. yaw Wang. Automated refinement checking for asynchronous processes. In *In Proc. 3rd FMCAD, LNCS*. Springer-Verlag, 2000.

[4] R. Alur, T. Henzinger, O. Kupferman, and M. Vardi. Alternating refinement relations. In *International Conference on Concurrency Theory*, pages 163–178, 1998.

[5] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, pages 100–109, Florida, October 1997.

[6] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, Sept. 2002.

[7] R. Alur, T. A. Henzinger, F. Y. C. Mang, S. Qadeer, S. K. Rajamani, and S. Taşiran. Mocha: Modularity in model checking. In *CAV 1998: Tenth International Conference on Computer-aided Verification, (LNCS Volume 1427)*, pages 521–525. Springer-Verlag: Berlin, Germany, 1998.

[8] K. Apt. Introduction to logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 494–574. Elsevier, 1990.

[9] A. Baltag and L. Moss. Logics for epistemic programs. *Synthese*, 139:165–224, 2004. Knowledge, Rationality & Action 1–60.

[10] A. Baltag, L. Moss, and S. Solecki. The logic of public announcements, common knowledge, and private suspicions. In I. Gilboa, editor, *Proceedings of the 7th Conference on Theoretical Aspects of Rationality and Knowledge (TARK 98)*, pages 43–56, 1998.

[11] R. E. Bryant. Graph-based algorithms for boolean function manipulation. In *IEEE transctions on Computers*, pages C–35(8): 677–619, 1986.

[12] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 1–33, Washington, D.C., 1990.

[13] E. Clarke, O. Grumberg, and D. Long. Verification tools for finite-state concurrent systems. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *A Decade of Concurrency — Reflections and Perspectives (LNCS Volume 803)*, pages 124–175. Springer-Verlag: Berlin, Germany, 1994.

[14] E. Clarke and B.-H. Schlingloff. Model checking. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, pages 1635 – 1790. Elsevier Science Publishers B.V., 2001.

[15] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In D. Kozen, editor, *Logics of Programs — Proceedings 1981 (LNCS Volume 131)*, pages 52–71. Springer-Verlag: Berlin, Germany, 1981.

[16] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press: Cambridge, MA, 2000.

[17] W. F. Clocksin and C. S. Mellish. *Programming in Prolog*. Springer-Verlag: Berlin, Germany, 1981.

[18] A. Dixit and S. Skeath. *Games of Strategy (Second Edition)*. New York: W. W. Norton & Co., Inc., 2004.

[19] G. Drimmelen. Satisfiability in alternating-time temporal logic. In *Eighteenth Annual IEEE Symposium on Logic in Computer Science (LICS 2003)*, pages 208–217, Ottawa, Canada, 2003.

[20] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science Volume B: Formal Models and Semantics*, pages 996–1072. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1990.

[21] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge MA, 1995.

[22] N. Francez. *Fairness*. Springer-Verlag: Berlin, Germany, 1986.

[23] H. Freudenthal. (formulation of the sum-and-product problem). *Nieuw Archief voor Wiskunde*, 3(17):152, 1969.

[24] H. Freudenthal. (solution of the sum-and-product problem). *Nieuw Archief voor Wiskunde*, 3(18):102–106, 1970.

[25] P. Gammie and R. van der Meyden. MCK: Model checking the logic of knowledge. In R. Alur and D. Peled, editors, *Proceedings of the 16th International Conference on Computer Aided Verification (CAV 2004)*, pages 479–483. Springer, 2004.

[26] M. Genereseth and N. Love. General game playing: Overview of the AAAI competition. Technical report, Stanford University, Stanford, 2005.

[27] J. Gerbrandy. *Bisimulations on Planet Kripke*. PhD thesis, University of Amsterdam, 1999. ILLC Dissertation Series DS-1999-01.

[28] J. Gerbrandy and W. Groeneveld. Reasoning about information change. *Journal of Logic, Language, and Information*, 6:147–169, 1997.

[29] R. Goldblatt. *Logics of Time and Computation (CSLI Lecture Notes Number 7)*. Center for the Study of Language and Information, Ventura Hall, Stanford, CA 94305, 1987. (Distributed by Chicago University Press).

[30] V. Goranko. Coalition games and alternating temporal logics. In J. van Benthem, editor, *Proceeding of the Eighth Conference on Theoretical Aspects of Rationality and Knowledge (TARK VIII)*, pages 259–272, Siena, Italy, 2001.

[31] V. Goranko and G. van Drimmelen. Complete axiomatization and decidability of alternating-time temporal logic. *Theoretical Computer Science*, 353(1):93–117, 2006.

[32] J. Halpern, R. van der Meyden, and M. Vardi. Complete axiomatizations for reasoning about knowledge and time. *SIAM Journal on Computing*, 33(3):674–703, 2004.

[33] J. Halpern and M. Vardi. The complexity of reasoning about knowledge and time. In *Proceedings 18th ACM Symposium on Theory of Computing*, pages 304–315, 1986.

[34] J. Halpern and M. Vardi. Model checking vs. theorem proving: a manifesto. In V. Lifschitz, editor, *Artificial intelligence and mathematical theory of computation: papers in honor of John McCarthy*, pages 151–176, San Diego, CA, USA, 1991. Academic Press Professional, Inc.

[35] J. Y. Halpern and Y. Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54(3):319–379, 1992.

[36] T. A. Henzinger, X. Liu, S. Qadeer, and S. K. Rajamani. Formal specification and verification of a dataflow processor array. In *In Proceedings of the International Conference on Computer-aided Design*, pages 494–499. IEEE Computer Society Press, 1999.

[37] T. A. Henzinger, S. Qadeer, and S. K. Rajamani. Verifying sequential consistency on shared-memory multiprocessor systems. In *In Proc. 11th Int. Conf. on Computer Aided Veri (CAV'99), LNCS 1633*, pages 301–315. Springer-Verlag, 1999.

[38] A. Herzig and N. Troquard. Knowing how to play: Uniform choices in logics of agency. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2006)*, Hakodate, Japan, 2006.

[39] J. Hintikka. *Knowledge and Belief.* Cornell University Press, Ithaca, NY, 1962.

[40] G. Holzmann. The Spin model checker. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.

[41] G. J. Holzmann and R. Joshi. Model-driven software verification. In *In Proc. 2001 ACM SIGPLANSIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE01*, pages 80–89. ACM Press, 2004.

[42] M. Huth and M. Ryan. *Logic in Computer Science – Modelling and Reasoning about Systems*. Cambridge University Press, Cambridge, UK, 2000.

[43] W. Jamroga and W. van der Hoek. Agents that know how to play. *Fundamenta Informaticae*, 63:2-3:185–219, 2004.

[44] S. Kripke. Semantical analysis of modal logic. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 9:67–96, 1963.

[45] C. I. Lewis. *A survey of symbolic logic*. Berkeley University of California Press, 1918.

[46] A. Lomuscio and F. Raimondi. The complexity of model checking concurrent programs against CTLK specifications. The Post-proceedings of DALT'06, the fourth workshop on Declarative Agent Languages and Technologies, 2006.

[47] A. Lomuscio and M. Ryan. On the relation between interpreted systems and kripke models. In C. Z. M. Pagnucco, W.R. Wobcke, editor, *Proceedings of the AI97 workshop on the theoretical and practical foundations of intelligent agents and agent-oriented systems*, pages 46–59, Berlin, 1998. Springer. Lecture Notes in Artificial Intelligence 1441.

[48] C. Lutz. Complexity and succinctness of public announcement logic. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 137–143, New York, NY, USA, 2006. ACM.

[49] Z. Manna and A. Pnueli. Verification of concurrent programs: The temporal framework. In R. S. Boyer and J. S. Moore, editors, *The Correctness Problem in Computer Science*, pages 215–273. The Academic Press: London, England, 1981.

[50] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag: Berlin, Germany, 1992.

[51] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems — Safety*. Springer-Verlag: Berlin, Germany, 1995.

[52] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers: Dordrecht, The Netherlands, 1993.

[53] W. Nabialek, A. Niewiadomski, W. Penczek, A. Polrola, and M. Szreter. Verics 2004: A model checker for real time and multi-agent systems. In *In Proc. of the Int. Workshop on Concurrency, Specication and Programming (CS&P04)*, pages 88–99. Humboldt University, 2004.

[54] N. Nethercote and A. Mycroft. The cache behaviour of large lazy functional programs on stock hardware. *SIGPLAN Notices*, 38(2 supplement):44–55, 2003.

[55] E. Pacuit. Some comments on history based structures. *Journal of Applied Logic*, 5(4):613 – 624, 2007. Selected papers from the 4th International Workshop on Computational Models of Scientific Reasoning and Applications, 4th International Workshop on Computational Models of Scientific Reasoning and Applications.

[56] R. Parikh and R. Ramanujam. Distributed processing and the logic of knowledge. In *Logic of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 256–268. Springer, 1985. A newer version appeared in *Journal of Logic, Language and Information*, vol. 12, 2003, pp. 453–467.

[57] M. Pauly. *Logic for Social Software*. PhD thesis, University of Amsterdam, 2001. ILLC Dissertation Series 2001-10.

[58] M. Pauly and M. Wooldridge. Logic for mechanism design — a manifesto. In *Proceedings of the 2003 Workshop on Game Theory and Decision Theory in Agent Systems (GTDT-2003)*, Melbourne, Australia, 2003.

[59] B. Pell. *Strategy Generation and Evaluation for Meta-Game Playing*. PhD thesis, Trinity College, University of Cambridge, 1993.

[60] R. Pfeifer. Building "fungus eaters": Design principles of autonomous agents. In *In Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior SAB96 (From Animals to Animats)*, pages 3–12. MIT Press, 1996.

[61] J. Plaza. Logics of public communications. In M. Emrich, M. Pfeifer, M. Hadzikadic, and Z. Ras, editors, *Proceedings of the 4th International Symposium on Methodologies for Intelligent Systems: Poster Session Program*, pages 201–216. Oak Ridge National Laboratory, 1989. ORNL/DSRD-24.

[62] A. Pnueli. The temporal logic of programs. In *Proceedings of the Eighteenth IEEE Symposium on the Foundations of Computer Science*, pages 46–57, 1977.

[63] A. Prior. *Time and Modality*. Oxford University Press: Oxford, England, 1957.

[64] J. P. Quielle and J. Sifakis. Specification and verification of concurrent systems in cesar. In *5th International Symposium on Programming*, pages 337–350, 1982.

[65] F. Raimondi and A. Lomuscio. Automatic verification of deontic properties of multi-agent systems. In *Proceedings of DEON04*, pages 228–242. Springer Verlag, 2004.

[66] F. Raimondi and A. Lomuscio. Symbolic model checking of multi-agent systems via OBDDs: an algorithm and its implementation. In *Proceedings of the Third International Conference on Autonomous Agents and Multiagent Systems (AAMAS-04)*, pages 630–637, New York, NY, 2004.

[67] F. Raimondi and A. Lomuscio. Verification of multiagent systems via ordered binary decision diagrams: An algorithm and its implementation. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 04)*, pages 630–637. IEEE Computer Society, 2004.

[68] S. Schiffel and M. Thielscher. Specifying multiagent environments in the game description language. In *Proceedings of ICAART 2009 - First Inter-*

national Conference on Agents and Artificial Intelligence. INSTICC Press, 2009.

[69] C. E. Shannon. Programming a computer for playing chess. *Computer chess compendium*, pages 2–13, 1988.

[70] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

[71] L. J. Stockmeyer and A. K. Chandra. Provably difficult combinatorial games. *SIAM Journal of Computing*, 8(2):151–174, 1979.

[72] P. Stone and M. Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, 2000.

[73] K. Su. Model checking temporal logics of knowledge in distributed systems. In D. McGuinness and G. Ferguson, editors, *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI 04)*, pages 98–103. AAAI Press / The MIT Press, 2004.

[74] J. van Benthem. Semantic parallels in natural language and computation. In *Logic Colloquium '87*, Amsterdam, 1989. North-Holland.

[75] J. van Benthem. Games in dynamic epistemic logic. *Bulletin of Economic Research*, 53(4):219–248, 2001.

[76] J. van Benthem. Ways of playing games: an analysis in dynamic-epistemic logic. In *Proceedings of The Fourth Conference on Logic and the Foundations of Game and Decision Theory (LOFT 2004)*, Torino, Italy, June 2004.

[77] J. van Benthem. *Logical Dynamics of Information and Interaction*. Manuscript, 2008.

[78] J. van Benthem, P. Dekker, J. van Eijck, M. de Rijke, and Y. Venema. *Logic in Action*. ILLC, Amsterdam, 2001.

[79] J. van Benthem, J. Gerbrandy, and E. Pacuit. Merging frameworks for interaction: Del and etl. In D. Samet, editor, *Proceedings of TARK 2007*, 2007.

[80] W. van der Hoek, A. Lomuscio, and M. Wooldridge. On the complexity of practical ATL model checking. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS 2006)*, pages 201–208. ACM, 2006.

[81] W. van der Hoek, M. Roberts, and M. Wooldridge. Social laws in alternating time: Effectiveness, feasibility, and synthesis. *Synthese*, 2007.

[82] W. van der Hoek, J. Ruan, and M. Wooldridge. Strategy logics and the game description language. In *A Meeting of the Minds: Proceedings of LORI Workshop on Logic, Rationality and Interaction*, Beijing, China, Aug. 2007.

[83] W. van der Hoek and M. Wooldridge. Model checking knowledge and time. In D. Bošnački and S. Leue, editors, *Model Checking Software, Proceedings of SPIN 2002 (LNCS Volume 2318)*, pages 95–111. Springer, 2002.

[84] W. van der Hoek and M. Wooldridge. Tractable multiagent planning for epistemic goals. In *Proceedings of the First International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS 2002)*, pages 1167–1174. ACM, 2002.

[85] W. van der Hoek and M. Wooldridge. Model checking cooperation, knowledge, and time — a case study. *Research in Economics*, 57(3):235–265, Sept. 2003.

[86] W. van der Hoek and M. Wooldridge. Time, knowledge, and cooperation: Alternating-time temporal epistemic logic and its applications. *Studia Logica*, 75(1):125–157, 2003.

[87] R. van der Meyden and K. Wong. Complete axiomatisations for reasoning about knowledge and branching time. *Studia Logica*, 75(1):93–123, 2003.

[88] H. van Ditmarsch. The russian cards problem. *Studia Logica*, 75:31–62, 2003.

[89] H. van Ditmarsch. The case of the hidden hand. In *Liber Amicorum Dick de Jongh*, 2004. `http://www.illc.uva.nl/D65/`.

[90] H. van Ditmarsch, J. Ruan, and W. van der Hoek. Model checking dynamic epistemics in branching time. In *Proceedings of Workshop on Formal Approaches to Multi-Agent Systems*, Durham, UK, Sept. 2007.

[91] H. van Ditmarsch, J. Ruan, and R. Verbrugge. Sum and product in dynamic epistemic logic. *Journal of Logic and Computation*, 18-4:563–588, 2006.

[92] H. van Ditmarsch, W. van der Hoek, and B. Kooi. *Dynamic Epistemic Logic*, volume 337 of *Synthese Library*. Springer, 2007.

[93] H. van Ditmarsch, W. van der Hoek, R. van der Meyden, and J. Ruan. Model checking russian cards. *Electronic Notes in Theoretical Computer Science*, 149:105–123, 2006. Presented at MoChArt 05 (Model Checking in Artificial Intelligence).

[94] J. van Eijck. Dynamic epistemic modelling. Technical report, Centrum voor Wiskunde en Informatica, Amsterdam, 2004. CWI Report SEN-E0424.

[95] M. Y. Vardi. Implementing knowledge-based programs. In *Proceedings of the Sixth Conference on Theoretical Aspects of Rationality and Knowledge (TARK 1996)*, pages 15–30, De Zeeuwse Stromen, The Netherlands, 1996.

[96] G. H. von Wright. *An Essay in Modal Logic*. North-Holland, Amsterdam, 1951.

[97] D. Walther, C. Lutz, F. Wolter, and M. Wooldridge. ATL satisfiability is indeed ExpTime-complete. *Journal of Logic and Computation*, 16:765–787, 2006.

[98] G. Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. The MIT Press: Cambridge, MA, Cambridge, MA, 1999.

[99] B. Wilsker. A study of multi-agent collaboration theories. In *ISI Research Report*, pages 96–449, 1996.

[100] M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, 2002.

[101] A. Yap. Product update and looking backward. Technical report, University of Amsterdam, 2006. ILLC Research Report PP-2006-39.