Behavioral Types in Programming Languages

Davide Ancona
Viviana Bono
Mario Bravetti
Joana Campos
Giuseppe Castagna
Pierre-Malo Deniélou
Simon J. Gay
Nils Gesbert
Elena Giachino
Raymond Hu
Einar Broch Johnsen
Francisco Martins
Viviana Mascardi
Fabrizio Montesi
Rumyana Neykova
Nicholas Ng
Luca Padovani
Vasco T. Vasconcelos
Nobuko Yoshida

now
the essence of knowledge
Boston — Delft

Foundations and Trends® in Programming Languages

Published, sold and distributed by:
now Publishers Inc.
PO Box 1024
Hanover, MA 02339
United States
Tel. +1-781-985-4510
www.nowpublishers.com
sales@nowpublishers.com

Outside North America:
now Publishers Inc.
PO Box 179
2600 AD Delft
The Netherlands
Tel. +31-6-51115274

The preferred citation for this publication is

D. Ancona et al.. *Behavioral Types in Programming Languages*. Foundations and Trends® in Programming Languages, vol. 3, no. 2-3, pp. 95–230, 2016.

This Foundations and Trends® issue was typeset in LaTeX using a class file designed by Neal Parikh. Printed on acid-free paper.

ISBN: 978-1-68083-135-1

Foundations and Trends® in Programming Languages
Volume 3, Issue 2-3, 2016

Editorial Scope

Topics

Foundations and Trends® in Programming Languages publishes survey and tutorial articles in the following topics:

- Abstract interpretation
- Compilation and interpretation techniques
- Domain specific languages
- Formal semantics, including lambda calculi, process calculi, and process algebra
- Language paradigms
- Mechanical proof checking
- Memory management
- Partial evaluation
- Program logic
- Programming language implementation
- Programming language security
- Programming languages for concurrency
- Programming languages for parallelism
- Program synthesis
- Program transformations and optimizations
- Program verification
- Runtime techniques for programming languages
- Software model checking
- Static and dynamic program analysis
- Type theory and type systems

Information for Librarians

Foundations and Trends® in Programming Languages, 2016, Volume 3, 4 issues. ISSN paper version 2325-1107. ISSN online version 2325-1131. Also available as a combined paper and online subscription.

Foundations and Trends® in Programming Languages
Vol. 3, No. 2-3 (2016) 95–230

DOI: 10.1561/2500000031

Behavioral Types in Programming Languages

Davide Ancona, DIBRIS, Università di Genova, Italy
Viviana Bono, Dipartimento di Informatica, Università di Torino, Italy
Mario Bravetti, Università di Bologna, Italy / INRIA, France
Joana Campos, LaSIGE, Faculdade de Ciências, Univ de Lisboa, Portugal
Giuseppe Castagna, CNRS, IRIF, Univ Paris Diderot, Sorbonne Paris Cité, France
Pierre-Malo Deniélou, Royal Holloway, University of London, UK
Simon J. Gay, School of Computing Science, University of Glasgow, UK
Nils Gesbert, Université Grenoble Alpes, France
Elena Giachino, Università di Bologna, Italy / INRIA, France
Raymond Hu, Department of Computing, Imperial College London, UK
Einar Broch Johnsen, Institutt for informatikk, Universitetet i Oslo, Norway
Francisco Martins, LaSIGE, Faculdade de Ciências, Univ de Lisboa, Portugal
Viviana Mascardi, DIBRIS, Università di Genova, Italy
Fabrizio Montesi, University of Southern Denmark
Rumyana Neykova, Department of Computing, Imperial College London, UK
Nicholas Ng, Department of Computing, Imperial College London, UK
Luca Padovani, Dipartimento di Informatica, Università di Torino, Italy
Vasco T. Vasconcelos, LaSIGE, Faculdade de Ciências, Univ de Lisboa, Portugal
Nobuko Yoshida, Department of Computing, Imperial College London, UK

Contents

Abstract

A recent trend in programming language research is to use behavioral type theory to ensure various correctness properties of large-scale, communication-intensive systems. Behavioral types encompass concepts such as interfaces, communication protocols, contracts, and choreography. The successful application of behavioral types requires a solid understanding of several practical aspects, from their representation in a concrete programming language, to their integration with other programming constructs such as methods and functions, to design and monitoring methodologies that take behaviors into account. This survey provides an overview of the state of the art of these aspects, which we summarize as the *pragmatics* of behavioral types.

D. Ancona et al.. *Behavioral Types in Programming Languages*. Foundations and Trends® in Programming Languages, vol. 3, no. 2-3, pp. 95–230, 2016.

DOI: 10.1561/2500000031.

1

Introduction

Modern society is increasingly dependent on large-scale software systems that are distributed, collaborative and communication-centered. Correctness and reliability of such systems depend on compatibility between components and services that are newly developed or may already exist. The consequences of failure are severe, including security breaches and unavailability of essential services. Current software development technology is not well suited to producing these large-scale systems, because of the lack of high-level structuring abstractions for complex communication behavior.

A recent trend in current research is to use *behavioral type theory* as the basis for new foundations, programming languages, and software development methods for communication-intensive distributed systems. Behavioral type theory encompasses concepts such as interfaces, communication protocols, contracts, and choreography. Roughly speaking, a *behavioral type* describes a software entity, such as an object, a communication channel, or a Web Service, in terms of the sequences of *operations* that allow for a *correct interaction* among the involved entities. The precise notions of "operations" and of "correct interaction" are very much context-dependent. Typical examples of op-

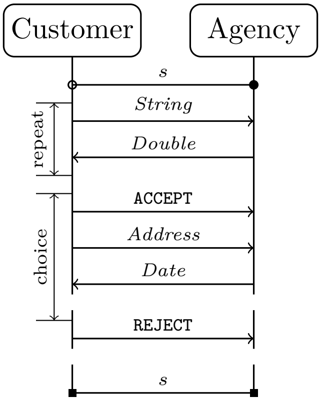

Figure 1.1: Graphical representation of the Customer-Agency protocol.

erations are invoking a method on an object, connecting a client with a Web Service in a distributed system, sending a message between cores in a parallel program. The notion of correct interaction may encompass both safety properties (such as the communication of valid method arguments, the absence of communication errors, the absence of deadlocks) as well as liveness properties (such as the eventual receipt of a message or the eventual termination of an interaction).

To illustrate some paradigmatic aspects of behavioral type theory more concretely, consider the diagram in Figure 1.1 depicting the interaction between two entities, named Customer and Agency. In this diagram, the horizontal lines represent interaction events between the two entities and the vertical lines represent their temporal ordering. The s-labeled line at the top of the diagram denotes the establishment of a connection between the two entities and the definition of an interaction scope that is often called *session*. The identifier s distinguishes this particular session from others (not depicted) in which Customer and Agency may be involved. We can think of s as the name of a communication channel that is known only to Customer and Agency. The proper interaction consists of two phases: the first one, marked as "repeat" in the figure, is made of an unbound number of queries issued by a Customer who is planning a trip through a travel Agency. Each query includes the journey details, abstracted as a message of type *String*, to which the Agency answers with the price of the journey, represented as a message of type *Double*. In the second phase, marked as "choice", Customer decides whether to book one of the journeys, which it signals by sending either an `ACCEPT` message followed by the *Address* to which the physical documents related to the journey should be delivered at some *Date* estimated by Agency, or a `REJECT` message that simply terminates the interaction. Arrows in the diagram denote the direction of messages. The discontinuity in the vertical development of the protocol suggests that the sub-protocols beginning with the `ACCEPT` and `REJECT` messages are mutually exclusive, the decision being taken by Customer. Eventually, the interaction between Customer and Agency terminates and the session that connects them is closed. This is denoted by the s-labeled line at the bottom of the diagram. In summary, the diagram describes a communication protocol between Customer and Agency as a set of valid sequences of interactions. Making sure that some piece of code modeling either Customer or Agency adheres to this protocol is among the purposes of behavioral type systems, and the technical instrument through which this is accomplished is behavioral types.

In the setting of typed programming languages, the challenge posed by describing a channel like s with a type is that the *same entity* s is used for exchanging messages of *different types* (labels such as `ACCEPT` and `REJECT`, integers, strings, floating-point numbers, dates, etc.) at *different times* and traveling in *different directions* (both Customer and Agency send and receive messages on s). Therefore, it is not obvious which *unique* type should be given to a channel like s or, equivalently, to the functions/methods for using it. The solution adopted in conventional (*i.e.*, non-behavioral) type systems, and that is found in virtually all mainstream programming languages used today, is to declare that communication channels like s can be used for exchanging "raw" messages in the form of *byte arrays* or *strings*. It is up to the programmer to appropriately *marshal* data into such raw messages before transmission and correspondingly *unmarshal* raw messages into data when they reach their destination. In Java, for instance, the `InputStream` and `OutputStream` interfaces and related ones provide `read` and `write` methods that respectively read data from a stream to a byte array and write data from a byte array to a stream. The main shortcoming of this approach is that it jeopardizes all the benefits and guarantees provided by the type system: such lax typing of channels and of the operations for using them provides no guarantee that the (un)marshalled data has the expected type, nor does it guarantee that messages flow along a channel in the direction intended by the protocol. Essentially, the approach corresponds to using *untyped* channels and establishes a border beyond which the type system of the programming language is no longer in effect. The resulting code is declared well typed by the compiler, but it may suffer from type-related errors (or other issues, such as deadlocks) at runtime.

The key idea of a behavioral type theory is to enrich the expressiveness of types so that it becomes possible to formally describe the sequences of messages (informally depicted in Figure 1.1) that are expected to be exchanged along the communication channel s that connects Customer and Agency. This type can then be used by a type checker to verify that the programs implementing Customer and Agency interact in accordance with the intended communication protocol. In fact, we can imagine *two* different types associated with channel s, depending on viewpoint we take, that of the Customer or that of the Agency. If we take the first viewpoint, we can describe s with a type T defined as

$$T = \bigoplus \left\{ \begin{array}{rcl} \texttt{QUERY} & : & !String.?Double.T \\ \texttt{ACCEPT} & : & !String.?Date.\texttt{end} \\ \texttt{REJECT} & : & \texttt{end} \end{array} \right\}$$

where:

- The symbol $\bigoplus$ denotes a choice of possible behaviors that Customer can attain to, each choice being represented by a symbolic label. In this case, the possible behaviors for Customer are querying the Agency (label `QUERY`), accepting an offer from the Agency (label `ACCEPT`), or quitting the interaction (label `REJECT`).

- The punctuation marks ! and ? respectively prefix the type of messages sent (*String*) and received (*Double* and *Date*) by Customer. With these annotations, we can specify the intended direction of messages.

- The punctuation marks : and . represent the sequentiality of actions described by the type. In this case, a Customer that queries an Agency must first send a message of type *String* and *then* wait for a message of type *Double*. With these annotations, we can specify how the capabilities of the channel change as the channel is used for input/output operations.

- The occurrence of T on the right hand side of the equation indicates that T is a *recursive* type, therefore allowing for an unbounded number of queries from Customer to Agency. This makes it possible to specify recursive protocols.

- `end` marks the points in which the interaction between Customer and Agency terminates and no more messages are supposed to be exchanged.

If we take the Agency viewpoint, it is reasonable to expect that the type of s should express complementary behaviors: the Agency offers choices when Customer selects one, the Agency receives a message when the Customer sends one, and vice versa. Customer and Agency should also agree on the moments in which the interaction terminates. This relation between the behaviors of Customer and Agency can be formalized as a notion of *duality* between the two types of s. In particular, the *dual* of T is the type S defined as

$$S = \sum \left\{ \begin{array}{rcl} \texttt{QUERY} & : & ?\mathit{String}.!\mathit{Double}.S \\ \texttt{ACCEPT} & : & ?\mathit{String}.!\mathit{Date}.\texttt{end} \\ \texttt{REJECT} & : & \texttt{end} \end{array} \right\}$$

obtained from T by swapping choices $\bigoplus$ with offers $\sum$, inputs ? with outputs !, and leaving `end` unchanged. Now, checking that Customer and Agency use the respective ends of s according to T and S makes sure that choices and offers match and messages of the right type are exchanged at the right time. In summary, that Customer and Agency interact correctly.

The successful application of behavioral types to the development of reliable, large-scale software requires both the study of formal type theories but also understanding and addressing more practical aspects, including the representation of behavioral types such as T and S in a concrete programming language, the integration of behavioral type checking with other programming constructs like methods and functions, and also design methodologies that take behaviors into account. The aim of this survey is to provide a first comprehensive overview of the state of the art of these aspects, which we may summarize as the *pragmatics* of behavioral types. The survey is structured as a series of chapters, each covering a particular programming paradigm or methodology. Below is an account of the content of each chapter:

- Chapter 2 is devoted to the integration of behavioral types into Object-Oriented languages. Object-oriented languages are relevant for their widespread adoption in the current development of software, for the wealth and popularity of tools that are available, and because objects nicely fit a distribution model to which behavioral types can be applied naturally. The integration can be achieved in different ways: either by *enriching* the languages with constructs (in particular, sessions) that call for a corresponding extension at the type level, or by *amalgamating* sessions and objects to the point that the objects themselves become the entities for which a behavioral description is required, for example to specify the order in which methods must/can be invoked. We also survey a parallel, but related line of research concerning *typestate*. This concept, originally introduced for discriminating the state of imperative variables (uninitialized, initialized, finalized), finds a natural application to describing object protocols and has been recently converging to behavioral typing.

- Chapter 3 explores the integration of behavioral types within functional languages. Functional languages are relevant for their qualities of being easily endowed with high-level type-theoretic and concurrent extensions, for their natural support to parallelism, and since they permit rapid prototyping. We survey three different approaches, one akin to an effect system, one based on explicit continuation passing, and one based on monads. Besides providing an out-of-the-box application of behavioral types to a concrete programming language, the continuation-based and monadic approaches can take advantage of the *type inference* engine of the language so that the programmer is not required to explicitly write (or annotate programs with) behavioral types, which can be automatically reconstructed from the source code of the program.

- High-performance computing often relies on parallel processes that synchronize by means of message passing. Chapter 4 describes the use of behavioral types in conjunction with Message Passing Interface (MPI) which is the *de facto* standard API for high-performance computing. Also in this case, behavioral types provide an effective means for making sure that communications occur without errors. We survey three alternative approaches making use of behavioral types in this context: one based on higher-level structuring abstractions, one based on source code verification, and one based on source code generation.

- Chapter 5 describes an application of behavioral types to multi-agent systems. The latter have been proved to be an industrial-strength technology for integrating and coordinating autonomous and heterogeneous entities in *open* systems. In this setting, the possibility of formally describing interaction protocols in the form of behavioral types enables forms of runtime monitoring for multi-agent systems.

- Chapter 6 provides an overview of the use of behavioral types in Singularity OS, a prototype Operating System developed by Microsoft that adopts communication as the fundamental and only synchronization mechanism between processes. Sing#, the programming language used for the implementation of Singularity OS, is an extension of C# that comprises both object-oriented and functional constructs and provides a native notion of *channel contract*, a form of behavioral type. The formal investigation of behavioral types in this setting has led to the discovery of unforseen system configurations that yield memory leaks and to the development of refined behavioral type theories preventing them.

- The WSDL and UDDI standards are technologies currently enabling the description of Web Service interfaces and the creation of Web Service repositories. Chapter 7 explores the potential of behavioral types, intended as abstract descriptions of Web Service behaviors, as natural generalizations of WSDL interfaces to realize sophisticated forms discovering, composition, and orchestration of Web Services.

- Chapter 8 illustrates the *design-by-contract* methodology for the development of possibly distributed, communicating systems. According to this methodology, behavioral types are used for describing, from a vantage point of view, the topology of the communication network, the communications that are supposed to occur, and in which order. Such global specifications serve multiple purposes: they are a valuable form of abstract specification of the overall behavior of a distributed system; they can be projected for describing the local behavior of the network participants to allow the modular type checking of complex systems; they enable the generation of monitors to verify, at runtime, that the participants of a heterogeneous distributed system behave as expected, even if only some or none of them have been type checked against their supposed or claimed behavior.

Overall, the survey provides substantial evidence that behavioral types have sprinkled a remarkable interest in the research community concerned with programming languages. The adoption of behavioral types beside the academic context proceeds more slowly, but nonetheless there are encouraging signals. As a matter of fact, it is known that programming languages tend to evolve slowly, especially when it comes to the integration of sophisticated typing disciplines. In this respect, approaches that rely on the *encoding* of behavioral types using conventional type constructors (§3.3), that allow for the verification of existing code (§4.2), or the type-driven generation of runtime monitors (Chapters 5 and 8), enable developers to fill the gap between theory and practice of behavioral typing with little or no changes to their programming environment and development workflow. The survey also contains pointers to industrial projects in which behavioral types already play a key role: the Ocean Observatories Initiative, which aims at the realization of a planetary-scale network for the trasmission of environmental data (§8.2), and the programming language Sing$^{\#}$, developed by Microsoft, which offers behavioral types as a native and key feature (Chapter 6). These early examples of industrial applications of behavioral types indirectly hint at their effectiveness in supporting the development of complex, large-scale systems for which correctness and reliability guarantees are of paramount importance.

References

Jonathan Aldrich, Joshua Sunshine, Darpan Saini, and Zachary Sparks. Typestate-oriented programming. In *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object-Oriented Programming Systems Languages and Applications*, pages 1015–1022. ACM, 2009.

David S. Allison, Miriam A. M. Capretz, Hany F. E. L. Yamany, and Shuying Wang. Privacy protection framework with defined policies for service-oriented architecture. *Journal of Software Engineering and Applications*, 5 (3):200–215, 2012.

Nuno Alves, Raymond Hu, Nobuko Yoshida, and Pierre-Malo Deniélou. Secure execution of distributed session programs. In *Proceedings of the 3rd Workshop on Programming Language Approaches to Concurrency and communication-cEntric Software*, volume 69 of *Electronic Proceedings in Theoretical Computer Science*, pages 1–11, 2010.

Davide Ancona, Sophia Drossopoulou, and Viviana Mascardi. Automatic Generation of Self-Monitoring MASs from Multiparty Global Session Types in Jason. In *Proceedings of the 10th International Workshop on Declarative Agent Languages and Technologies*, volume 7784 of *Lecture Notes in Computer Science*, pages 76–95. Springer, 2012.

Davide Ancona, Matteo Barbieri, and Viviana Mascardi. Constrained global types for dynamic checking of protocol conformance in multi-agent systems. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1377–1379. ACM, 2013a.

Davide Ancona, Viviana Mascardi, and Matteo Barbieri. Global types for dynamic checking of protocol conformance in multi-agent systems. Technical report, Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi, Università di Genova, 2013b.

John Langshaw Austin. *How to Do Things with Words*. Oxford: Clarendon Press, 1962.

Nels E. Beckman, Kevin Bierhoff, and Jonathan Aldrich. Verifying correct usage of atomic blocks and typestate. In *Proceedings of the 23rd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 227–244. ACM, 2008.

Fabio L. Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing Multi-Agent Systems with JADE*. John Wiley & Sons, 2007.

Giovanni Bernardi and Matthew Hennessy. Mutually testing processes - (extended abstract). In *Proceedings of the 24th International Conference on Concurrency Theory*, volume 8052 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2013.

Lorenzo Bettini, Sara Capecchi, Mariangiola Dezani-Ciancaglini, Elena Giachino, and Betti Venneri. Session and Union Types for Object Oriented Programming. In Rocco De Nicola, Pierpaolo Degano, and José Meseguer, editors, *Concurrency, Graphs and Models*, volume 5065 of *Lecture Notes in Computer Science*, pages 659–680. Springer-Verlag, 2008a.

Lorenzo Bettini, Mario Coppo, Loris D'Antoni, Marco De Luca, Mariangiola Dezani-Ciancaglini, and Nobuko Yoshida. Global progress in dynamically interleaved multiparty sessions. In *Proceedings of the 19th International Conference on Concurrency Theory*, volume 5201 of *Lecture Notes in Computer Science*, pages 418–433. Springer, 2008b.

Lorenzo Bettini, Sara Capecchi, Mariangiola Dezani-Ciancaglini, Elena Giachino, and Betti Venneri. Deriving session and union types for objects. *Mathematical Structures in Computer Science*, 23(6):1163–1219, 2013.

Karthikeyan Bhargavan, Ricardo Corin, Pierre-Malo Deniélou, Cédric Fournet, and James J. Leifer. Cryptographic protocol synthesis and verification for multiparty sessions. In *Proceedings of the 22nd IEEE Computer Security Foundations Symposium*, pages 124–140. IEEE, 2009.

Kevin Bierhoff and Jonathan Aldrich. Modular typestate checking of aliased objects. In *Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 301–320. ACM, 2007.

Laura Bocchi, Tzu-Chun Chen, Romain Demangeon, Kohei Honda, and Nobuko Yoshida. Monitoring networks through multiparty session types. In *Proceedings of the 8th International Federated Conference on Distributed Computing Techniques*, volume 7892 of *Lecture Notes in Computer Science*, pages 50–65. Springer, 2013.

Laura Bocchi, Weizhen Yang, and Nobuko Yoshida. Timed multiparty session types. In *Proceedings of the 25th International Conference on Concurrency Theory*, volume 8704 of *Lecture Notes in Computer Science*, pages 419–434. Springer, 2014.

Laura Bocchi, Julien Lange, and Nobuko Yoshida. Meeting deadlines together. In *Proceedings of the 26th International Conference on Concurrency Theory*, volume 42 of *Leibniz International Proceedings in Informatics*, pages 283–296. Schloss Dagstuhl, 2015.

Viviana Bono and Luca Padovani. Typing Copyless Message Passing. *Logical Methods in Computer Science*, 8:1–50, 2012.

Viviana Bono, Chiara Messa, and Luca Padovani. Typing Copyless Message Passing. In *Proceedings of the 20th European Symposium on Programming*, volume 6602 of *Lecture Notes in Computer Science*, pages 57–76. Springer, 2011.

Viviana Bono, Luca Padovani, and Andrea Tosatto. Polymorphic Types for Leak Detection in a Session-Oriented Functional Language. In *Proceedings of the 8th International Federated Conference on Distributed Computing Techniques*, volume 7892 of *Lecture Notes in Computer Science*, pages 83–98. Springer, 2013.

Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming Multi-Agent Systems in AgentSpeak Using Jason*. John Wiley & Sons, 2007.

Michele Boreale and Mario Bravetti. Advanced mechanisms for service composition, query and discovery. In Martin Wirsing and Matthias M. Hölzl, editors, *Results of the SENSORIA Project*, volume 6582 of *Lecture Notes in Computer Science*, pages 282–301. Springer, 2011.

John Boyland. Checking interference with fractional permissions. In Radhia Cousot, editor, *Proceedings of the 10th International Symposium on Static Analysis*, volume 2694 of *Lecture Notes in Computer Science*, pages 55–72. Springer, 2003.

John Boyland. Fractional permissions. In Dave Clarke, James Noble, and Tobias Wrigstad, editors, *Aliasing in Object-Oriented Programming. Types, Analysis and Verification*, volume 7850 of *Lecture Notes in Computer Science*, pages 270–288. Springer, 2013.

BPMN. Business Process Model and Notation. http://www.omg.org/spec/BPMN/2.0/, 2011.

Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *Journal of ACM*, 30:323–342, 1983.

Mario Bravetti and Gianluigi Zavattaro. Towards a unifying theory for choreography conformance and contract compliance. In Markus Lumpe and Wim Vanderperren, editors, *Software Composition*, volume 4829 of *Lecture Notes in Computer Science*, pages 34–50. Springer, 2007.

Mario Bravetti and Gianluigi Zavattaro. Contract compliance and choreography conformance in the presence of message queues. In Roberto Bruni and Karsten Wolf, editors, *Proceedings of the 5th International Workshop on Web Services and Formal Methods*, volume 5387 of *Lecture Notes in Computer Science*, pages 37–54. Springer, 2008a.

Mario Bravetti and Gianluigi Zavattaro. A foundational theory of contracts for multi-party service composition. *Fundamenta Informaticae*, 89(4):451–478, 2008b.

Mario Bravetti and Gianluigi Zavattaro. Service discovery and composition based on contracts and choreographic descriptions. In Guadalupe Ortiz and Javier Cubo, editors, *Adaptive Web Services for Modular and Reusable Software Development: Tactics and Solutions*, pages 60–88. IGI Global, 2013.

Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In *Proceedings of the 21th International Conference on Concurrency Theory*, volume 6269 of *Lecture Notes in Computer Science*, pages 222–236. Springer, 2010.

Joana Campos and Vasco T. Vasconcelos. MOOL. Available at http://gloss.di.fc.ul.pt/mool/, accessed May 21, 2016.

Joana Campos and Vasco T. Vasconcelos. Channels as objects in concurrent object-oriented programming. In *Proceedings of the 3rd Workshop on Programming Language Approaches to Concurrency and communication-cEntric Software*, volume 69 of *Electronic Proceedings in Theoretical Computer Science*, pages 12–28, 2010.

Sara Capecchi, Mario Coppo, Mariangiola Dezani-Ciancaglini, Sophia Drossopoulou, and Elena Giachino. Amalgamating Sessions and Methods in Object Oriented Languages with Generics. *Theoretical Computer Science*, 410:142–167, 2009.

Marco Carbone and Fabrizio Montesi. Deadlock-freedom-by-design: multiparty asynchronous global programming. In *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 263–274. ACM, 2013.

Marco Carbone, Kohei Honda, Nobuko Yoshida, Robin Milner, Gary Brown, and Steve Ross-Talbot. A theoretical basis of communication-centred concurrent programming, 2006. Available at `http://www.w3.org/2002/ws/chor/edcopies/theory/note.pdf`.

Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured communication-centered programming for web services. *ACM Transactions on Programming Languages and Systems*, 34(2):8, 2012.

Marco Carbone, Fabrizio Montesi, and Carsten Schürmann. Choreographies, logically. In *Proceedings of the 25th International Conference on Concurrency Theory*, volume 8704 of *Lecture Notes in Computer Science*, pages 47–62. Springer, 2014.

Giuseppe Castagna, Nils Gesbert, and Luca Padovani. A Theory of Contracts for Web Services. *ACM Transactions on Programming Languages and Systems*, 31(5), 2009.

Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, and Luca Padovani. On global types and multi-party session. *Logical Methods in Computer Science*, 8(1), 2012.

CDL. W3C Web Services Choreography Description Language. `http://www.w3.org/2002/ws/chor/`, 2002.

Bradford L. Chamberlain, David Callahan, and Hans P. Zima. Parallel programmability and the chapel language. *International Journal of High Performance Computing Applications*, 21(3):291–312, 2007.

Philippe Charles, Christian Grothoff, Vijay Saraswat, Christopher Donawa, Allan Kielstra, Kemal Ebcioglu, Christoph von Praun, and Vivek Sarkar. X10: an object-oriented approach to non-uniform cluster computing. In *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 519–538. ACM, 2005.

Tzu-Chun Chen, Laura Bocchi, Pierre-Malo Deniélou, Kohei Honda, and Nobuko Yoshida. Asynchronous distributed monitoring for multiparty session enforcement. In *Trustworthy Global Computing*, volume 7173 of *Lecture Notes in Computer Science*, pages 25–45. Springer, 2012.

Chor. Programming Language. Available at `http://www.chor-lang.org/`, accessed May 21, 2016.

Ernie Cohen, Markus Dahlweid, Mark Hillebrand, Dirk Leinenbach, Michał Moskal, Thomas Santen, Wolfram Schulte, and Stephan Tobies. VCC: A practical system for verifying concurrent C. In *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics*, volume 5674 of *Lecture Notes in Computer Science*, pages 23–42. Springer, 2009.

Mario Coppo, Mariangiola Dezani-Ciancaglini, and Nobuko Yoshida. Asynchronous Session Types and Progress for Object-Oriented Languages. In *Proceedings of the 9th International Conference on Formal Methods for Open Object-Based Distributed Systems*, volume 4468 of *Lecture Notes in Computer Science*, pages 1–31. Springer, 2007.

Ricardo Corin and Pierre-Malo Deniélou. A protocol compiler for secure sessions in ML. In *Trustworthy Global Computing*, volume 4912 of *Lecture Notes in Computer Science*, pages 276–293. Springer, 2008.

Ricardo Corin, Pierre-Malo Deniélou, Cédric Fournet, Karthikeyan Bhargavan, and James J. Leifer. A secure compiler for session abstractions. *Journal of Computer Security*, 16(5):573–636, 2008.

Silvia Crafa and Luca Padovani. The Chemical Approach to Typestate-Oriented Programming. In *Proceedings of the ACM International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 917–934. ACM, 2015.

Ornela Dardha, Elena Giachino, and Davide Sangiorgi. Session types revisited. In *Proceedings of the 14th symposium on Principles and practice of declarative programming*, pages 139–150. ACM, 2012.

Rocco De Nicola and Matthew Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

Robert DeLine and Manuel Fähndrich. Typestates for objects. In *Proceedings of the 18th European Conference on Object-Oriented Programming*, volume 3086 of *Lecture Notes in Computer Science*, pages 465–490. Springer, 2004.

Romain Demangeon and Kohei Honda. Nested protocols in session types. In *Proceedings of the 23rd International Conference on Concurrency Theory*, volume 7454 of *Lecture Notes in Computer Science*, pages 272–286. Springer, 2012.

Romain Demangeon, Kohei Honda, Raymond Hu, Rumyana Neykova, and Nobuko Yoshida. Practical interruptible conversations: Distributed dynamic verification with multiparty session types and Python. *Formal Methods in System Design*, 46(3):197–225, 2015.

Pierre-Malo Deniélou and Nobuko Yoshida. Dynamic multirole session types. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 435–446. ACM, 2011.

Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty session types meet communicating automata. In *Proceedings of the 21st European Symposium on Programming*, volume 7211 of *Lecture Notes in Computer Science*, pages 194–213. Springer, 2012.

Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming*, volume 7966 of *Lecture Notes in Computer Science*, pages 174–186. Springer, 2013.

Pierre-Malo Deniélou, Nobuko Yoshida, Andi Bejleri, and Raymond Hu. Parameterised multiparty session types. *Logical Methods in Computer Science*, 8(4), 2012.

Jolie development team. Jolie Programming Language. Available at `http://www.jolie-lang.org/`, accessed May 21, 2016.

Mariangiola Dezani-Ciancaglini, Nobuko Yoshida, Alexander Ahern, and Sophia Drossopoulou. l_{doos}: a Distributed Object-Oriented language with Session types. In *Trustworthy Global Computing*, volume 3705 of *Lecture Notes in Computer Science*, pages 299–318. Springer, 2005.

Mariangiola Dezani-Ciancaglini, Dimitris Mostrous, Nobuko Yoshida, and Sophia Drossopoulou. Session Types for Object-Oriented Languages. In *Proceedings of the 20th European Conference on Object-Oriented Programming*, volume 4067 of *Lecture Notes in Computer Science*, pages 328–352. Springer, 2006.

Mariangiola Dezani-Ciancaglini, Sophia Drossopoulou, Elena Giachino, and Nobuko Yoshida. Bounded Session Types for Object-Oriented Languages. In *Proceedings of the 5th International Symposium on Formal Methods for Components and Objects*, volume 4709 of *Lecture Notes in Computer Science*, pages 207–245. Springer-Verlag, 2007.

Mariangiola Dezani-Ciancaglini, Sophia Drossopoulou, Dimitris Mostrous, and Nobuko Yoshida. Objects and Session Types. *Information and Computation*, 207(5):595–641, 2009.

Sophia Drossopoulou, Mariangiola Dezani-Ciancaglini, and Mario Coppo. Amalgamating the Session Types and the Object Oriented Programming Paradigms. In *Proceedings of the Workshop on Multiparadigm Programming in Object-Oriented Languages*, 2007.

Manuel Fähndrich and Robert DeLine. Adoption and focus: Practical linear types for imperative programming. In Jens Knoop and Laurie J. Hendren, editors, *Proceedings of the 2002 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 13–24. ACM, 2002.

Manuel Fähndrich, Mark Aiken, Chris Hawblitzel, Orion Hodson, Galen Hunt, James R. Larus, and Steven Levi. Language Support for Fast and Reliable Message-based Communication in Singularity OS. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems*, pages 177–190. ACM, 2006.

MPI Forum. *MPI: A Message-Passing Interface Standard—Version 3.0*. High-Performance Computing Center Stuttgart, 2012.

Luca Fossati, Raymond Hu, and Nobuko Yoshida. Multiparty session nets. In *Trustworthy Global Computing*, volume 8902 of *Lecture Notes in Computer Science*, pages 112–127. Springer, 2014.

Foundation for Intelligent Physical Agents. FIPA ACL message structure specification. Available at `http://www.fipa.org/specs/fipa00061/SC00061G.html`, accessed May 21, 2016.

Cédric Fournet, Tony Hoare, Sriram K. Rajamani, and Jakob Rehof. Stuck-free conformance. In Rajeev Alur and Doron Peled, editors, *Proceedings of the 16th International Conference on Computer Aided Verification*, volume 3114 of *Lecture Notes in Computer Science*, pages 242–254. Springer, 2004.

Ronald Garcia, Eric Tanter, Roger Wolff, and Jonathan Aldrich. Foundations of typestate-oriented programming. *ACM Transactions on Programming Languages and Systems*, 2014.

Simon Gay. Bounded polymorphism in session types. *Mathematical Structures in Computer Science*, 18(5):895–930, 2008.

Simon Gay and Vasco T. Vasconcelos. Linear type theory for asynchronous session types. *Journal of Functional Programming*, 20(1):19–50, 2010.

Simon J. Gay, Nils Gesbert, António Ravara, and Vasco Thudichum Vasconcelos. Modular session types for objects. *Logical Methods in Computer Science*, 11(4), 2015.

Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

GMC. From communicating machines to graphical choreographies. Available at `https://bitbucket.org/julien-lange/gmc-synthesis`, accessed 21 May 2016.

GTV. Global Types Verification. Available at `http://www.disi.unige.it/person/MascardiV/Software/globalTypes.html`, accessed 21 May 2016.

Claudio Guidi, Ivan Lanese, Fabrizio Montesi, and Gianluigi Zavattaro. Dynamic error handling in service oriented applications. *Fundamenta Informaticae*, 95(1):73–102, 2009.

Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty Asynchronous Session Types. In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 273–284. ACM, 2008.

Kohei Honda, Aybek Mukhamedov, Gary Brown, Tzu-Chun Chen, and Nobuko Yoshida. Scribbling interactions with a formal foundation. In *Proceedings of the 7th International Conference on Distributed Computing and Internet Technology*, volume 6536 of *Lecture Notes in Computer Science*, pages 55–75. Springer, 2011.

Kohei Honda, Eduardo R. B. Marques, Francisco Martins, Nicholas Ng, Vasco Thudichum Vasconcelos, and Nobuko Yoshida. Verification of mpi programs using session types. In *Proceedings of the 19th European MPI Users' Group Meeting on Recent Advances in the Message Passing Interface*, volume 7490 of *Lecture Notes in Computer Science*, pages 291–293. Springer, 2012.

Kohei Honda, Raymond Hu, Rumyana Neykova, Tzu-Chun Chen, Romain Demangeon, Pierre-Malo Deniélou, and Nobuko Yoshida. Structuring communication with session types. In Gul A. Agha, Atsushi Igarashi, Naoki Kobayashi, Hidehiko Masuhara, Satoshi Matsuoka, Etsuya Shibayama, and Kenjiro Taura, editors, *Concurrent Objects and Beyond - Papers dedicated to Akinori Yonezawa on the Occasion of His 65th Birthday*, volume 8665 of *Lecture Notes in Computer Science*, pages 105–127. Springer, 2014.

Raymond Hu and Nobuko Yoshida. Hybrid session verification through API generation. In *Proceedings of the 19th International Conference on Fundamental Approaches to Software Engineering*, volume 9633 of *Lecture Notes in Computer Science*, pages 401–418. Springer, 2016.

Raymond Hu, Nobuko Yoshida, and Kohei Honda. Session-based distributed programming in java. In *Proceedings of the 22nd European Conference on Object-Oriented Programming*, volume 5142 of *Lecture Notes in Computer Science*, pages 516–541. Springer, 2008.

Raymond Hu, Dimitrios Kouzapas, Olivier Pernet, Nobuko Yoshida, and Kohei Honda. Type-safe eventful sessions in Java. In *Proceedings of the 24th European Conference on Object-Oriented Programming*, volume 6183 of *Lecture Notes in Computer Science*, pages 329–353. Springer, 2010.

Raymond Hu, Rumyana Neykova, Nobuko Yoshida, and Romain Demangeon. Practical Interruptible Conversations: Distributed Dynamic Verification with Session Types and Python. In *Proceedings of the 4th International Conference on Runtime Verification*, volume 8174 of *Lecture Notes in Computer Science*, pages 130–148. Springer, 2013.

Galen Hunt, James Larus, Martín Abadi, Mark Aiken, Paul Barham, Manuel Fähndrich, Chris Hawblitzel, Orion Hodson, Steven Levi, Nick Murphy, Bjarne Steensgaard, David Tarditi, Ted Wobber, and Brian Zill. An Overview of the Singularity Project. Technical Report MSR-TR-2005-135, Microsoft Research, 2005.

Keigo Imai, Shoji Yuen, and Kiyoshi Agusa. Session type inference in Haskell. In *Proceedings of the 3rd Workshop on Programming Language Approaches to Concurrency and communication-cEntric Software*, volume 69 of *Electronic Proceedings in Theoretical Computer Science*, pages 74–91, 2010.

Svetlana Jakšić and Luca Padovani. Exception Handling for Copyless Messaging. In *Proceedings of the 14th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming*, pages 151–162. ACM, 2012.

Svetlana Jakšić and Luca Padovani. Exception Handling for Copyless Messaging. *Science of Computer Programming*, 84:22–51, 2014.

Nicholas R. Jennings, Katia P. Sycara, and Michael Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.

Jacek Kopecký, Tomas Vitvar, Carine Bournez, and Joel Farrell. Sawsdl: Semantic annotations for wsdl and xml schema. *IEEE Xplore: IEEE Internet Computing*, 11(6):60–67, 2007.

Dimitrios Kouzapas, Ornela Dardha, Roly Perera, and Simon J. Gay. Mungo. `http://www.dcs.gla.ac.uk/research/mungo`, 2015.

Ivan Lanese, Claudio Guidi, Fabrizio Montesi, and Gianluigi Zavattaro. Bridging the gap between interaction- and process-oriented choreographies. In *Proceedings of the 6th IEEE International Conference on Software Engineering and Formal Methods*, pages 323–332. IEEE, 2008.

Julien Lange, Emilio Tuosto, and Nobuko Yoshida. From communicating machines to graphical choreographies. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 221–232. ACM, 2015.

Hugo A. López, Eduardo R. B. Marques, Francisco Martins, Nicholas Ng, César Santos, Vasco Thudichum Vasconcelos, and Nobuko Yoshida. Protocol-based verification of message-passing parallel programs. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 280–298. ACM, 2015.

Eduardo R. B. Marques, Francisco Martins, Nicholas Ng, César Santos, Vasco T. Vasconcelos, and Nobuko Yoshida. Specification and verification of protocols for MPI programs. Available at `http://www.di.fc.ul.pt/~vv/papers/marques.martins_specification-verification-mpi.pdf`, 2013a.

Eduardo R. B. Marques, Francisco Martins, Vasco T. Vasconcelos, Nicholas Ng, and Nuno Martins. Towards deductive verification of MPI programs against session types. In *Proceedings of the 6th Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software*, volume 137 of *Electronic Proceedings in Theoretical Computer Science*, pages 103–113, 2013b.

Viviana Mascardi and Davide Ancona. Attribute global types for dynamic checking of protocols in logic-based multiagent systems. *Theory and Practice of Logic Programming*, 13(4-5-Online-Supplement), 2013.

James Mayfield, Yannis Labrou, and Tim Finin. Evaluation of KQML as an agent communication language. In *Proceedings of the Workshop on Agent Theories, Architectures, and Languages*, volume 1037 of *Lecture Notes in Computer Science*, pages 347–360. Springer, 1995.

Jan Mendling and Michael Hafner. From inter-organizational workflows to process execution: Generating bpel from ws-cdl. In *On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops*, volume 3762 of *Lecture Notes in Computer Science*, pages 506–515. Springer, 2005.

Filipe Militão. Design and implementation of a behaviorally typed programming system for web services. Master's thesis, Universidade Nova de Lisboa, Faculdade de Ciências e Tecnologia. Available at `http://run.unl.pt/handle/10362/1792`, accessed May 21, 2016.

Filipe Militão and Luís Caires. An exception aware behavioral type system for object-oriented programs. In *Proceedings of the Simpósio de Informática*, 2009. Available at `http://www.cs.cmu.edu/~foliveir/papers/corta2009.pdf`.

Fabrizio Montesi. Process-aware web programming with Jolie. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 761–763. ACM, 2013a.

Fabrizio Montesi. *Choreographic Programming*. Ph.D. thesis, IT University of Copenhagen, 2013b. Available at `http://www.fabriziomontesi.com/files/choreographic_programming.pdf`.

Fabrizio Montesi and Nobuko Yoshida. Compositional choreographies. In *Proceedings of the 24th International Conference on Concurrency Theory*, volume 8052 of *Lecture Notes in Computer Science*, pages 425–439. Springer, 2013.

Fabrizio Montesi, Claudio Guidi, and Gianluigi Zavattaro. Composing services with Jolie. In *Proceedings of the 5th IEEE European Conference on Web Services*, pages 13–22. IEEE Computer Society, 2007.

Fabrizio Montesi, Claudio Guidi, and Gianluigi Zavattaro. Service-oriented programming with Jolie. In *Web Services Foundations*, pages 81–107. Springer, 2014.

Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21 (12):993–999, 1978.

Matthias Neubauer and Peter Thiemann. An implementation of session types. In *Proceedings of the 6th International Symposium on Practical Aspects of Declarative Languages*, volume 3057 of *Lecture Notes in Computer Science*, pages 56–70. Springer, 2004.

Rumyana Neykova and Nobuko Yoshida. Multiparty session actors. In *Proceedings of the 16th International Conference on Coordination Models and Languages*, volume 8459 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2014.

Rumyana Neykova, Nobuko Yoshida, and Raymond Hu. SPY: Local Verification of Global Protocols. In *Proceedings of the 4th International Conference on Runtime Verification*, volume 8174 of *Lecture Notes in Computer Science*, pages 358–363. Springer, 2013.

Nicholas Ng and Nobuko Yoshida. Pabble: parameterised Scribble. *Service Oriented Computing and Applications*, 9(3-4):269–284, 2015.

Nicholas Ng, Nobuko Yoshida, Olivier Pernet, Raymond Hu, and Yiannos Kryftis. Safe parallel programming with session java. In *Proceedings of the 13th International Conference on Coordination Models and Languages*, volume 6721 of *Lecture Notes in Computer Science*, pages 110–126. Springer, 2011.

Nicholas Ng, Nobuko Yoshida, and Kohei Honda. Multiparty session c: Safe parallel programming with message optimisation. In *Proceedings of the 50th International Conference on Objects, Models, Components, Patterns*, volume 7304 of *Lecture Notes in Computer Science*, pages 202–218. Springer, 2012.

Nicholas Ng, Jose G. F. Coutinho, and Nobuko Yoshida. Protocols by default: Safe MPI code generation based on session types. In *Proceedings of the 24th International Conference on Compiler Construction*, volume 9031 of *Lecture Notes in Computer Science*, pages 212–232. Springer, 2015.

OASIS. Web Services Business Process Execution Language. `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html`, 2007.

Pabble-MPI. MPI Generation Framework. Available at `https://github.com/sessionc/pabble-mpi`, accessed May 21, 2016.

Luca Padovani. *Contract-based Discovery and Adaptation of Web Services*, volume 5569 of *Lecture Notes in Computer Science*, pages 213–260. Springer, 2009.

Luca Padovani. Contract-Based Discovery of Web Services Modulo Simple Orchestrators. *Theoretical Computer Science*, 411:3328–3347, 2010.

Luca Padovani. A Simple Library Implementation of Binary Sessions. Technical Report hal-01216310, Dipartimento di Informatica, Università di Torino, Italy, 2015. Available at `https://hal.archives-ouvertes.fr/hal-01216310`.

Riccardo Pucella and Jesse A. Tov. Haskell session types with (almost) no class. In *Proceedings of the 1st ACM SIGPLAN Symposium on Haskell*, pages 25–36. ACM, 2008.

Zongyan Qiu, Xiangpeng Zhao, Chao Cai, and Hongli Yang. Towards the theoretical foundation of choreography. In *Proceedings of the 16th International Conference on World Wide Web*, pages 973–982. ACM, 2007.

Anand S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In *Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, volume 1038 of *Lecture Notes in Computer Science*, pages 42–55. Springer, 1996.

Arend Rensink and Walter Vogler. Fair testing. *Information and Computation*, 205(2):125–198, 2007.

Davide Sangiorgi and David Walker. *The Pi-Calculus - a theory of mobile processes.* Cambridge University Press, 2001.

Vijay Saraswat, George Almasi, Ganesh Bikshandi, Calin Cascaval, David Cunningham, David Grove, Sreedhar Kodali, Igor Peshansky, and Olivier Tardieu. The asynchronous partitioned global address space model. In *Proceedings of the 1st Workshop on Advances in Message Passing*, 2010.

Scribble. Available at `http://www.scribble.org`, accessed May 21, 2016.

Jeremy G. Siek and Walid Taha. Gradual typing for objects. In *Proceedings of the 21st European Conference on Object-Oriented Programming*, volume 4609 of *Lecture Notes in Computer Science*, pages 2–27. Springer, 2007.

Singularity OS. Available at `http://singularity.codeplex.com/`, accessed May 21, 2016.

SJ. Session J. Available at `http://code.google.com/p/sessionj/`, accessed May 21, 2016.

Guy L. Steele. Parallel programming and parallel abstractions in fortress. In *Proceedings of the 8th International Symposium on Functional and Logic Programming*, volume 3945 of *Lecture Notes in Computer Science*, pages 1–1. Springer, 2006.

Zachary Stengel and Tevfik Bultan. Analyzing Singularity Channel Contracts. In *Proceedings of the 18th International Symposium on Software Testing and Analysis*, pages 13–24. ACM, 2009.

Robert E. Strom and Shaula Yemini. Typestate: A programming language concept for enhancing software reliability. *IEEE Transactions on Software Engineering*, 12(1):157–171, 1986.

Joshua Sunshine, Karl Naden, Sven Stork, Jonathan Aldrich, and Éric Tanter. First-class state change in Plaid. In *Proceedings of the 26th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 713–732. ACM, 2011a.

Joshua Sunshine, Sven Stork, Karl Naden, and Jonathan Aldrich. Changing state in the Plaid language. In *Proceedings of the Companion to the 26th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 37–38. ACM, 2011b.

Bernardo Toninho, Luís Caires, and Frank Pfenning. Higher-order processes, functions, and sessions: A monadic integration. In *Proceedings of the 22nd European Symposium on Programming*, volume 7792 of *Lecture Notes in Computer Science*, pages 350–369. Springer, 2013.

Vasco T. Vasconcelos, Simon Gay, and António Ravara. Typechecking a multithreaded functional language with session types. *Theoretical Computer Science*, 368(1–2):64–87, 2006.

Jules Villard, Étienne Lozes, and Cristiano Calcagno. Proving Copyless Message Passing. In *Proceedings of the 7th Asian Symposium on Programming Languages and Systems*, volume 5904 of *Lecture Notes in Computer Science*, pages 194–209. Springer, 2009.

Jules Villard, Étienne Lozes, and Cristiano Calcagno. Tracking Heaps That Hop with Heap-Hop. In *Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 of *Lecture Notes in Computer Science*, pages 275–279. Springer, 2010.

Roger Wolff, Ronald Garcia, Éric Tanter, and Jonathan Aldrich. Gradual typestate. In *Proceedings of the 25th European Conference on Object-Oriented Programming*, volume 6813 of *Lecture Notes in Computer Science*, pages 459–483. Springer, 2011.

Nobuko Yoshida and Vasco Thudichum Vasconcelos. Language primitives and type discipline for structured communication-based programming revisited: Two systems for higher-order session communication. *Electronic Notes in Theoretical Computer Science*, 171(4):73–93, 2007.