Report from Dagstuhl Seminar 24051

Next Generation Protocols for Heterogeneous Systems

Stephanie Balzer*[1], Marco Carbone*[2], Roland Kuhn*[3], and Peter Thiemann*[4]

1 **Carnegie Mellon University, USA.** balzers@cs.cmu.edu
2 **IT University of Copenhagen, DK.** carbonem@itu.dk
3 **Actyx AG – München, DE.** roland@actyx.io
4 **University of Freiburg, DE.** thiemann@informatik.uni-freiburg.de

Abstract

The emergence of new computing systems, like cloud computing, blockchains, and Internet of Things (IoT), replaces the traditional monolithic software hardware stack with a distributed heterogeneous model. This change poses new demands on the programming languages for developing such systems: *compositionality*, allowing decomposition of a system into smaller, possibly heterogeneous, parts and composition of the individually verified parts into a verified whole, *security*, asserting end-to-end integrity and confidentiality, *quantitative reasoning methods*, accounting for timing and probabilistic events, and, as a cross-cutting concern, *certification* of asserted properties in terms of independently verifiable, machine-checked proofs.

Characteristics of this emerging computation model are distribution of the participating entities and message passing as the primary means of communication. Message passing is also the communication model underlying behavioral types and programming languages, making them uniquely fitted for this new application domain. Behavioral types explicitly capture the protocols of message exchange and have a strong theoretical foundation. Recent applications of behavioral types include smart contract languages, information flow control, and machine-checked proofs of safety properties. Although these early explorations are promising, the current state of the art of behavioral types and programming languages lacks a comprehensive account of the above-mentioned demands.

This Dagstuhl Seminar aims to gather experts from academia and industry to discuss the use of programming languages tailored to tackle the challenges posed by today's emerging distributed and heterogeneous computing platforms, e.g., by making use of behavioral types. It will focus on static and possibly dynamic mechanisms to support compositionality, security, quantitative reasoning, and certification.

Seminar January 28 – February 2, 2024 – https://www.dagstuhl.de/24051
2012 ACM Subject Classification Theory of computation → Process calculi; Theory of computation → Type structures
Keywords and phrases behavioural types, concurrency, programming languages, session types
Digital Object Identifier 10.4230/DagRep.14.1.108

* Editor / Organizer

Next Generation Protocols for Heterogeneous Systems, *Dagstuhl Reports*, Vol. 14, Issue 1, pp. 108–129
Editors: Stephanie Balzer, Marco Carbone, Roland Kuhn and Peter Thiemann

Dagstuhl Reports
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Executive Summary

Stephanie Balzer (Carnegie Mellon University, USA)
Marco Carbone (IT University of Copenhagen, DK)
Roland Kuhn (Actyx AG – München, DE)
Peter Thiemann (University of Friburg, DE)

This Dagstuhl Seminar followed the earlier Dagstuhl Seminars 17051 "Theory and Applications of Behavioural Types" and 21372 "Behavioural Types: Bridging Theory and Practice". Whereas Seminar 17051 was focusing on theoretical aspects of behavioural types, and Seminar 21372 focused on bridging the gap with practical application, this seminar was much broader and aimed at extending to other communities such as security and other areas of programming languages.

Initial preparations

Based on the ideas of our seminar proposal, we established four key general areas: quantitative systems, verification, mechanisation, and security. We assigned each area to a day of the week (from Monday to Thursday) and asked an invitee representative of the area to give an introductory talk. Then, each of these talks was followed by other talks and breakout rooms related to the area. Breakout rooms were established during the seminar based on discussions with the rest of the participants. As a result of this, the first part of the week consisted primarily of talks, while the second part included more time for breakout sessions.

Activities and outcomes

Throughout the seminar, the participants gathered in focused breakout groups: the findings of the breakout groups are described in more detail in the last part of the report. The participants of several breakout groups have agreed to continue their work and collaboration after the seminar.

In addition to these more structured breakout sessions there were further lively improvised meetings and discussions (especially after dinner) which are not summarised in the report.

Overall, we believe that the seminar activities were a success. At the end of the seminar the participants agreed to remain in contact to continue the discussions, and foster new collaborations. There was strong enthusiasm for organising a follow-up Dagstuhl Seminar in the future, perhaps taking place in about 1–2 years time. One concrete outcome was the submission of a position paper (cf. the working group "Typing Across Heterogeneous Components") that has been accepted and presented at PLACES 2024 (co-located with ETAPS).

2 Table of Contents

3 Overview of Talks

3.1 Area talk: Security of Heterogenous Systems: Principles, Practice, and a Case for Secure Runtimes

Aslan Askarov (Aarhus University – Aarhus, Denmark)

The classical computer security principle of the least common mechanism says that resource sharing creates security problems and should be treated carefully. This talk highlights that programming language runtimes that handle sensitive information at different confidentiality levels are such common mechanisms and, therefore, can inadvertently leak information. We examine runtime aspects such as schedulers and mailboxes and also study mitigating traffic analysis attacks using information flow techniques.

3.2 Logical Relations for Session-Typed Concurrency

Stephanie Balzer (Carnegie Mellon University – Pittsburgh, US)

Joint work of Stephanie Balzer, Farzaneh Derakhshan, Robert Harper, Yue Yao

Program *equivalence* is the fulcrum for reasoning about and proving properties of programs. For noninterference, for example, program equivalence up to the secrecy level of an observer is shown. A powerful enabler for such proofs are *logical relations*. Logical relations only were adopted for session types relatively recently – but exclusively for terminating languages. This talk scales logical relations to *general recursive session types*. It develops a logical relation for progress-sensitive noninterference (PSNI) for intuitionistic linear logic session types (ILLST), tackling the challenges non-termination and concurrency pose, and shows that logical equivalence is *sound and complete* with regard to closure of weak bisimilarity under parallel composition, using a *biorthogonality* argument. A distinguishing feature of the logical relation is its stratification with an *observation index* (as opposed to a step or unfolding index), a crucial shift to make the logical relation closed under parallel composition in a concurrent setting.

3.3 Area talk: Program Development Tools for Secure Multi-Party Computation

Marina Blanton (University at Buffalo – Buffalo, US)

Secure multi-party computation permits evaluation of a function or a program on protected private inputs in such a way that the computation participants have no access to the data in the clear throughout the computation. The security guarantees are such that only the computation outcome becomes disclosed to the designated parties. In this talk, we discussed the setup, security definitions, and their differences from the properties of other definitions

used in the programming languages community. The second part of the talk discussed PICCO, a compiler for transforming general-purpose programs intended to be executed on private data into the corresponding secure multi-party computation protocols. We discussed compiler optimizations and mechanisms for making it easier for programmers to develop efficient programs in this framework.

3.4 Contracts for Session-based Programming with Linear Dependent Types

Luis Caires (IST – Lisbon, PT)

We sketch a novel approach to linear dependent session types based on a Proposition-as-Types foundation of session-based programs, which targets the development of a linear dependent type theory with equality types for session behaviour, allowing properties of linear objects to be expressed.

3.5 Regrading Policies for Flexible Information Flow Control in Session-Typed Concurrency

Farzaneh Derakhshan (Illinois Institute of Technology – Chicago, US)

Joint work of Farzaneh Derakhshan, Stephanie Balzer, Yue Yao

Noninterference guarantees that an attacker cannot infer secrets by interacting with a program. An information flow control type system asserts noninterference by tracking the level of information learned and disallowing leakage to entities of lesser or unrelated levels. These restrictions cater to scenarios in which the information learned by an entity monotonically increases with program progression but are at odds with control flow constructs, permitting interaction with entities of lower levels in the continuation. Relaxing such restrictions is particularly challenging in a concurrent setting. This paper utilizes session types to track the flow of information and develops an information flow control type system for message-passing concurrent processes that allows downgrading the pc for the next loop iteration upon recursion. To ensure noninterference, the type system relies on regrading policies, ensuring that any confidential information learned during the high-security parts of the loop cannot be rolled forward to the next iteration. To express regrading policies, the type system is complemented with integrity to ensure that entities with different regrading policies can be safely composed. The paper develops the type system and proves progress-sensitive noninterference for well-typed programs, ruling out timing attacks that exploit the relative order of messages. The type system has been implemented in a type checker, which supports security-polymorphic processes using local security theories.

3.6 The Rational Programmer

Christos Dimoulas (Northwestern University – Evanston, US)

The productivity of developers depends on the quality of the available programming languages: their support for adequate testing, for locating and fixing mistakes, and for maintenance tasks. If a programming language does not support these routine tasks, the developer is forced to resort to labor-intensive and ineffective workarounds. Put differently, it isn't about the syntax, the types, or the semantics, but about the pragmatics of a programming language. The problem is that the PL research area has so far few tools to evaluate pragmatics.

The Rational Programmer is a new scientific instrument for that purpose. While simulations have a long history in computer science applications, the Rational Programmer method puts them to new use in PL research. The heart of the method is a simulation, namely an algorithmic abstraction of information gathering in a work context. The outcome of a rational programmer simulation is typically a strategy that a developer can employ. It may also point designers and researchers to a problematic aspect of a language. Finally, it can inform instructors how to teach students the effective use of a language. In this talk, I will demonstrate the workings of the Rational Programmer method with examples.

3.7 Special Delivery: Programming with Mailbox Types

Simon Fowler (University of Glasgow, GB)

The asynchronous and unidirectional communication model supported by mailboxes is a key reason for the success of actor languages like Erlang and Elixir for implementing reliable and scalable distributed systems. While many actors may send messages to some actor, only the actor may (selectively) receive from its mailbox. Although actors eliminate many of the issues stemming from shared memory concurrency, they remain vulnerable to communication errors such as protocol violations and deadlocks.

Mailbox types are a novel behavioural type system for mailboxes first introduced for a process calculus by de'Liguoro and Padovani in 2018, which capture the contents of a mailbox as a commutative regular expression. Due to aliasing and nested evaluation contexts, moving from a process calculus to a programming language is challenging. This paper presents Pat, the first programming language design incorporating mailbox types, and describes an algorithmic type system. We make essential use of quasi-linear typing to tame some of the complexity introduced by aliasing. Our algorithmic type system is necessarily co-contextual, achieved through a novel use of backwards bidirectional typing, and we prove it sound and complete with respect to our declarative type system. We implement a prototype type checker, and use it to demonstrate the expressiveness of Pat on a factory automation case study and a series of examples from the Savina actor benchmark suite.

3.8 Correct orchestration of Federated Learning: formalisation and verification

Silvia Ghilezan (Mathematical Institute – Belgrade, RS)

Main reference Ivan Prokic, Silvia Ghilezan, Simona Kasterovic, Miroslav Popovic, Marko Popovic, Ivan Kastelan: "Correct orchestration of Federated Learning generic algorithms: formalisation and verification in CSP", CoRR, Vol. abs/2306.14529, 2023.
URL https://doi.org/10.48550/ARXIV.2306.14529

Federated learning (FL) is a machine learning setting where clients keep the training data decentralised and collaboratively train a model either under the coordination of a central server (centralised FL) or in a peer-to-peer network (decentralised FL). Correct orchestration is one of the main challenges. In this paper, we formally verify the correctness of two generic FL algorithms, a centralised and a decentralised one, using the CSP process calculus and the PAT model checker. The CSP models consist of CSP processes corresponding to generic FL algorithm instances. PAT automatically proves the correctness of the two generic FL algorithms by proving their deadlock freeness (safety property) and successful termination (liveness property). The CSP models are constructed bottom-up by hand as a faithful representation of the real Python code and is automatically checked top-down by PAT.

3.9 Information-Flow Control in Choreographies

Andrew Hirsch (University at Buffalo – SUNY, US)

Information-flow control is an important information-security–enforcement mechanism. It requires that secret information not be allowed to influence (or be used to compute) public data. This ensures that no private data will be leaked to the outside world. However, enforcing information-flow control in the concurrent world has proven incredibly difficult. The only known versions are incredibly restrictive. In current work, we are adding information-flow control to choreographic programs, where the extra restrictiveness is not necessary. In this talk, I will explain what goes wrong with information-flow control in concurrent settings, and why choreographic programming appears to rescue it.

3.10 Actris tool presentation

Jonas Kastberg Hinrichsen (Aarhus University, DK)

URL https://iris-project.org/actris/

Binary sessions, specifying bidirectional exchanges of messages between two processes, has widely been used to model idealised reliable communication, where messages are never dropped, duplicated, or arrive out of order. Such sessions allow for sophisticated protocol structures, as evidenced by the ever expanding work on session types, a behavioural type

system for specifying the types of individual messages of a sequence of exchanges. However, the expressivity of session types is often restricted to a decidable fragment, which prohibits them from reasoning about functional correctness.

In this tool presentation, I present the ongoing story of Actris tool – a framework for session type-based reasoning in separation logic – and how it can be used to reason about reliable communication. In particular, I demonstrate the full verification of a suite of programs that combine message passing and shared memory concurrency. I additionally briefly cover the various extensions of Actris; notably how it has been applied to the verification of distributed systems and deadlock freedom.

3.11 Area talk: Mechanized verification of type systems using Iris

Robbert Krebbers (Radboud University Nijmegen, NL)

This talk gives an introduction to the "logical approach" to proving type safety. I will first present a simple version, and then scale up to a small session-typed language. I will show that this approach is well-suited for mechanization of challenging type systems in the Coq proof assistant

3.12 Behavioural Types for Local-First Software: replicated roles, full availability

Roland Kuhn (Actyx AG)

Joint work of Roland Kuhn, Hernán Melgratte, Emilio Tuosto
Main reference Roland Kuhn, Hernán C. Melgratti, Emilio Tuosto: "Behavioural Types for Local-First Software", in Proc. of the 37th European Conference on Object-Oriented Programming, ECOOP 2023, July 17-21, 2023, Seattle, Washington, United States, LIPIcs, Vol. 263, pp. 15:1–15:28, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.
URL https://doi.org/10.4230/LIPICS.ECOOP.2023.15

In this work we formalise an existing system for high-availability industry automation: the constraint that availability must be maximised – even at the cost of strong consistency – poses some interesting challenges. The basis is given by the peer-to-peer, uncoordinated, but causality-preserving event log replication of the Actyx middleware. Our work resulted in well-formedness constraints that ensure that a designed interaction protocol will achieve eventual consensus on the event trace of its execution, without any need for further coordination. This holds even in a dynamic swarm setting, where any role can be replicated any number of times and new participants can join and leave the system at any time.

3.13 Probabilistic Theories of Choreographic Programming

Marco Peressotti (University of Southern Denmark – Odense, DK)

Choreographic programming is a paradigm for developing concurrent and distributed systems, where programs are choreographies that define, from a global viewpoint, the computations and interactions that communicating processes should enact. Choreography compilation translates choreographies into the local definitions of process behaviours, given as terms in a process calculus. In this talk we present the first theory of choreographic programming language that incorporates probabilistic aspects for local computation, choreographic choice, and scheduling. We start from an established theory of choreographic programming [1, 2, 3, 4, 5] and integrate various probabilistic features while maintaining the original syntax. We show that the original compilation procedure can still be used and establish its correctness in terms of probabilistic bisimilarity via standard up-to techniques. We discuss how the various probabilistic features impact the design of the semantics and the lessons learned while integrating them.

References

1 Montesi, F. 2023. Introduction to Choreographies. Cambridge University Press. DOI:10.1017/9781108981491
2 Cruz-Filipe Luís, Montesi, F. and Peressotti, M. 2023. A Formal Theory of Choreographic Programming. Journal of Automated Reasoning. 67, 21 (2023), 1–34. DOI:10.1007/s10817-023-09665-3.
3 Cruz-Filipe Luís, Montesi, F. and Peressotti, M. 2021. Certifying Choreography Compilation. Theoretical Aspects of Computing – ICTAC 2021 – 18th International Colloquium, Virtual Event, Nur-Sultan, Kazakhstan, September 8-10, 2021, Proceedings (2021), 115–133. DOI:10.1007/978-3-031-17715-6_15
4 Cruz-Filipe Luís, Montesi, F. and Peressotti, M. 2021. Formalising a Turing-Complete Choreographic Language in Coq. 12th International Conference on Interactive Theorem Proving (ITP 2021) (Dagstuhl, Germany, 2021), 15:1–15:18. DOI:10.4230/LIPIcs.ITP.2021.15
5 Cruz-Filipe Luís, Graversen, E., Montesi, F. and Peressotti, M. 2023. Reasoning About Choreographic Programs. Coordination Models and Languages (2023), 144–162. DOI:10.1007/978-3-031-35361-1_8

3.14 Comparing Process Calculi using Encodings

Kirstin Peters (Universität Augsburg, DE)

Encodings are often used to compare process calculi. To rule out trivial or meaningless encodings, they are augmented with encodability criteria. This talk is about how to reason about the quality of encodability criteria and how to set up such criteria.

3.15 Mechanizing Session-Types: Enforcing linearity without linearity

Brigitte Pientka (McGill University – Montréal, CA)

Joint work of Brigitte Pientka, Chuta Sano, Ryan Kavanagh
Main reference Chuta Sano, Ryan Kavanagh, Brigitte Pientka: "Mechanizing Session-Types using a Structural View: Enforcing Linearity without Linearity", CoRR, Vol. abs/2309.12466, 2023.
URL https://doi.org/10.48550/ARXIV.2309.12466

Session types employ a linear type system that ensures that communication channels cannot be implicitly copied or discarded. As a result, many mechanizations of these systems require modeling channel contexts and carefully ensuring that they treat channels linearly. We demonstrate a technique that localizes linearity conditions as additional predicates embedded within type judgments, which allows us to use structural typing contexts instead of linear ones. This technique is especially relevant when leveraging (weak) higher-order abstract syntax to handle channel mobility and the intricate binding structures that arise in session-typed systems.

Following this approach, we mechanize a session-typed system based on classical linear logic and its type preservation proof in the proof assistant Beluga, which uses the logical framework LF as its encoding language. We also prove adequacy for our encoding. This shows the tractability and effectiveness of our approach in modelling substructural systems such as session-typed languages.

3.16 Router-based Analysis of Multiparty Protocols

Jorge Pérez (University of Groningen, NL)

We are interested in the rigorous verification of message-passing programs, which operate by exchanging messages across distributed networks. Ensuring that these communicating programs are correct is important but highly challenging.

Originated from the realms of Concurrency Theory and Programming Languages, Multiparty Session Types (MPSTs) offer a convenient methodology for the development and verification of message-passing programs. The methodology of MPSTs offers a structured approach to the design of advanced verification techniques, both static (via type systems) and dynamic (via monitoring architectures). Interestingly, these static and verification techniques can be defined by following principled approaches based on resource-aware logics, in particular Girard's Linear Logic.

In this talk, I will overview recent work by my group in this direction, and in particular I discuss how the concept of router of a multiparty protocol can be effective for runtime verification.

3.17 Behavioural up/down casting for statically typed languages (tool presentation)

António Ravara (NOVA University of Lisbon, PT)

URL https://github.com/jdmota/java-typestate-checker

We provide support for polymorphism in static typestate analysis for object-oriented languages with upcasts and downcasts. Recent work has shown how typestate analysis can be embedded in the development of Java programs to obtain safer behaviour at runtime, e.g., absence of null pointer errors and protocol completion. In that approach, inheritance is supported at the price of limiting casts in source code, thus only allowing those at the beginning of the protocol, i.e., immediately after objects creation, or at the end, and in turn seriously affecting the applicability of the analysis.

We provide a solution to this open problem in typestate analysis by introducing a theory based on a richer data structure, named typestate tree, which supports upcast and downcast operations at any point of the protocol by leveraging union and intersection types. The soundness of the typestate tree-based approach has been mechanised in Coq. The theory can be applied to most object-oriented languages statically analysable through typestates, thus opening new scenarios for acceptance of programs exploiting inheritance and casting. To defend this thesis, we show an application of the theory, by embedding the typestate tree mechanism in a Java-like object-oriented language, and proving its soundness.

Accepted in ECOOP'24.

3.18 Deciding Subtyping for Asynchronous Multiparty Sessions

Felix Stutz (University of Luxembourg, LU)

Joint work of Elaine Li, Felix Stutz, Thomas Wies
Main reference Elaine Li, Felix Stutz, Thomas Wies: "Deciding Subtyping for Asynchronous Multiparty Sessions", in Proc. of the Programming Languages and Systems – 33rd European Symposium on Programming, ESOP 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part I, Lecture Notes in Computer Science, Vol. 14576, pp. 176–205, Springer, 2024.
URL https://doi.org/10.1007/978-3-031-57262-3_8

Multiparty session types (MSTs) are a type-based approach to verifying communication protocols, represented as global types in the framework. We present a precise subtyping relation for asynchronous MSTs with communicating state machines (CSMs) as implementation model. We address two problems: when can a local implementation safely substitute another, and when does an arbitrary CSM implement a global type? We define safety with respect to a given global type, in terms of subprotocol fidelity and deadlock freedom. Our implementation model subsumes existing work which considers local types with restricted choice. We exploit the connection between MST subtyping and refinement to formulate concise conditions that are directly checkable on the candidate implementations, and use them to show that both problems are decidable in polynomial time. This talk was given by Felix Stutz (Max Planck Institute for Software Systems). It is based on joint work with Elaine Li and Thomas Wies (New York University). In April 2024, the corresponding paper was published in the proceedings of the 33rd European Symposium on Programming (ESOP24).

3.19 Area talk: Quantitative techniques

Emilio Tuosto (Gran Sasso Science Institute – L'Aquil, IT)

This talk gives a bird-eye watch of the literature at the intersection between quantitative techniques and behavioural specifications. More precisely, following the chronological order, the talk surveys the papers concerned with resource awareness, time, and probabilities. The talk strives to succinctly highlight the main contributions of each paper and distill some interesting open problems. Although related to the topic of the talk, data-awareness was intentionally left out of the exposition in order to maintain the focus centred on quantitative approaches.

3.20 System f^{μ}_{ω} with context-free session types

Vasco T. Vasconcelos (University of Lisbon, PT)

Joint work of Diogo Poças, Diana Costa, Andreia Mordido, Vasco T. Vasconcelos
Main reference Diogo Poças, Diana Costa, Andreia Mordido, Vasco T. Vasconcelos: "System F^{μ}_{ω} with Context-free Session Types", in Proc. of the Programming Languages and Systems – 32nd European Symposium on Programming, ESOP 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings, Lecture Notes in Computer Science, Vol. 13990, pp. 392–420, Springer, 2023.
URL https://doi.org/10.1007/978-3-031-30044-8_15

We study increasingly expressive type systems, from F^{μ} –an extension of the polymorphic lambda calculus with equirecursive types–to F^{μ}_{ω} —the higher-order polymorphic lambda calculus with equirecursive types and context-free session types. Type equivalence is given by a standard bisimulation defined over a novel labelled transition system for types. Our system subsumes the contractive fragment of F^{μ}_{ω} as studied in the literature. Decidability results for type equivalence of the various type languages are obtained from the translation of types into objects of an appropriate computational model: finite-state automata, simple grammars and deterministic pushdown automata. We show that type equivalence is decidable for a significant fragment of the type language. We further propose a message-passing, concurrent functional language equipped with the expressive type language and show that it enjoys preservation and absence of runtime errors for typable processes.

3.21 STL3: Toward Security via Free Theorems in a Session-Typed Linear Language with Locations

Andrew Wagner (Northeastern University – Boston, US)

Joint work of Andrew Wagner, Amal Ahmed

We present work in progress on a minimal extension to intuitionistic binary session types that reifies channel names into types. Along with quantification over names, this establishes a form of name parametricity with which common security properties like authenticity, binding,

and hiding can be expressed as free theorems. We identify some of the key challenges in proving these free theorems, and more specifically where standard techniques seem to be insufficient.

3.22 Area talk: Verification

Nobuko Yoshida (University of Oxford, GB)

I give the overview on verification of session types: (1) Linear Logic-Based Session Types; (2) Full Abstraction Results of System F; (3) Asynchronous Communication Optimisations in Rust; (4) Comparison of Performances in Session-Based Rust; and (5) Correspondence with Communication Automata.

4 Working groups

4.1 Breakout Group: IFC and Noninterference

Aslan Askarov, Stephanie Balzer, Marina Blanton, Christos Dimoulas, Emanuele D'Osualdo, Farzaneh Derakhshan, and Andrew Wagner

During this break-out session, the discussion focused on the following points: state of the art in information flow control, quantitative information flow and its connection to secure multi-party computation, modern formulation of noninterference policies using the epistemic (knowledge-based) framework for declassification, and comparison of different noninterference theorem statements.

4.2 Breakout Group: Secure Multiparty Computation

Amal Ahmed, Aslan Askarov, Stephanie Balzer, Andrew Wagner, Marina Blanton, Christos Dimoulas, Emanuele D'Osualdo, Farzaneh Derakhshan, and Philipp Haller

During the session, we discussed the security definitions employed in the secure multi-party computation literature, properties of the resulting protocols, and their relationship to the definitions and properties achieved in other settings, e.g., in information flow control literature. Because the model provides powerful guarantees at the level of individual operations, it becomes possible to achieve properties which otherwise would be difficult to achieve in other settings. For example, the fact that a computation participant does not see private values implies that the participant is unable to leak secret data. During the discussion we were also seeking connections with other PL concepts and recent developments as a way to build a fruitful collaboration. Topics that came up include concurrency, information flow, and orchestration.

4.3 Breakout Group: Logical Relations and Session Types

Amal Ahmed, Stephanie Balzer, Luis Caires, Emanuele D'Osualdo, Farzaneh Derakhshan, Adrian Francalanza, Ralf Jung, Robbert Krebbers, Peter Thiemann, and Andrew Wagner

This breakout session gathered people currently employing the advanced proof method of logical relations and people interested in learning more about it. Of particular concern was the recent developments that employ logical relations in a session-typed concurrent setting. As such the session allowed experts to provide an overview of the current state of the art and latest achievements. A significant portion of the discussion focused on recent logical relations for session types that are indexed not only with a single type, but a set of types. These relations are necessitated by program equivalence statements such as noninterference, demanding observations along possibly several channels. The question came up whether such relations would also support proofs of free theorems in a linear setting based on a parametricity argument.

4.4 Real-World Applications of Behavioural Types

Kirstin Peters, Silvia Ghilezan, Jesper Bengtson, Christos Dimoulas, Marco Carbone, Felix Stutz, and Antonio Ravara

- Why were attendees interested real-world applications?
  - Understanding how behavioural types can help with real-world projects and what is missing
  - Justifying need for research in grant proposals
- Some problems/examples for behavioural types
  - JEDIS bug (Java implementation of REDIS): was still trying to use a socket even though an exception was thrown before
  - tsunami prediction project
- More first-hand experiences
  - Racket contracts that only specify functional correctness and do not capture artificially introduced bugs in code; common issue that model is not expressive enough; need for specifying protocols
  - Erlang typing for gen_server: practically driven, challenges with failures, gradual typing
- Practical problems with evaluation
  - How to set up experiments?
  - What are target measures? How does one show that number of bugs reduce when using behavioural types?
  - Getting companies interested in pilot projects
- Industrial collaborations
  - often different interests than research: small steps are enough and there is little theoretical work needed to support this

- actual porting from theoretical work to practice is a lot of work and proves only worth it if something is found (risky for PhD projects)
- wish for push-bottom technology (e.g. Bedrock systems but fine with writing specifications)
- issue that universities seem to focus quite a bit on funding opportunities with such collaborations rather than research content (tone down probably?)

- Possibilities for case studies/collaborations with more applied domains, even in CS
  - it is likely that computing-heavy have software that suffers from issues BT solve
  - hospitals as part of universities: problem with data-privacy and training that is needed
  - IOT and intermittent computing

4.5 Typing Across Heterogeneous Components

Roland Kuhn, Philipp Haller, Sam Lindley, Vasco T. Vasconcelos, Simon Fowler, Alceste Scalas, Malte Viering, and Raymond Hu

A large barrier to the widespread adoption of behavioural typing disciplines is that we must, at present, select a single tool and write our entire system using that tool. In practice at the very least we want to be able to design our system to make use of different components that use different, but not completely unrelated, behavioural type disciplines and tools (for example, having part of the system able to statically verify data constraints using refinement types, and another part of the system be able to check this using some form of monitoring).

There has been some work on runtime monitoring against local session types, but at present the results there only state that an invalid incoming message was dropped. There was a desire to go further than this, for example reporting failures, requesting message re-sends / reporting rejections to senders, or perhaps doing some message reordering.

Two related topics arose out of the discussion.

The first was a language design issue: what language features should our tools, at a minimum, support in order to allow interoperability? We thought that some way of handling failures (e.g., through exceptions) was probably important in this regard, although sometimes challenging to integrate into tools.

The second issue was about reconciling compatible yet subtly-different protocols, and a paper by Dezani et al. on session isomorphisms seemed important here. There was an extensive discussion about the notion of some sort of intermediate representation that could be used to implement adapters that can for example reorder messages and manipulate data sent along the wire in order to realise these transformations in practice in a uniform way.

We hope to write a position paper for PLACES in the coming weeks.

4.6 Probabilistic Behavioural Types

Emilio Tuosto, Silvia Ghilezan, Emanuele D'Osualdo, Jorge Pérez, Nobuko Yoshida, Marco Carbone, Marco Peressotti, Kirstin Peters, and Alceste Scalas

Why probabilistic behavioural types?

Emilio: the answer is not obvious. My coauthors and I started working on them as a specification of a desired probabilistic process behaviour, that we use for monitoring `https://doi.org/10.1016/j.scico.2022.102847,https://doi.org/10.1007/978-3-030-78142-2_7`. I also have a work on probabilistic analysis of session types `https://doi.org/10.4230/LIPIcs.CONCUR.2020.14`.

Silvia Ghilezan: my interest comes from the connection between probabilistic lambda calculus and probabilistic logic

Emanuele D'Osualdo: my interest comes from my work on hyperproperties, I would like to look at systems that are probabilistic in nature

Jorge Perez: I am not sure I understand the meaning of probabilistic session types. Even if we have a probabilistic calculus (like e.g. probabilistic pi-calculus) what do we gain by putting probabilities in types?

Nobuko Yoshida: I am interested in understanding how to encode probabilistic lambda calculus in a session typed calculus, and it seems we need probabilistic (multiparty) session types

Marco Peressotti: My main motivation is understand how to extend choreographic programming to express and reason about randomness (e.g., randomness in algorithms, in the underlying communication model etc.) and eventually quantum protocols.

Kirstin Peters: I have both practical interest and quantum systems, which have probabilistic LTSs. We may use probabilistic session types to check that e.g. a program deadlocks with a certain probability.

Alceste Scalas: I share Jorge's doubts (in fact I have only worked on probabilistic session types as specifications for for run-time onitoring, with Emilio et al). I have a bit of difficulty in seeing the "killer application" where putting probabilities in a session type gives you a clearly useful verification technique that may not be achieved using other analysis techniques for probabilistic programs.

Marco Carbone: I am also looking for applications that would clarify my understanding, like Jorge and Alceste Applications of probabilistic session types.

Emilio: Ugo dal Lago's paper uses probabilistic session types for modelling 'cyptographic experiments: `https://doi.org/10.4230/LIPIcs.CONCUR.2022.37`.

Marco Peressotti: Self-stabilising algorithms may also be a field of application

Marco Carbone: Tools like Prism allow for writing models with probabilities, but some models cannot be model-checked due to the state explosion problem. Could a type discipline like probabilistic session types help?

Nobuko Yoshida, Kirstin Peters, Marco Carbone, Marco Peressotti, Emilio Tuosto: we could look at probabilistic CCS examples, probabilistic dining philosophers, and see how probabilistic session types could allow e.g. to abstract and reduce the state space while still deriving useful probabilistic information (e.g. discarding low-probability events that may not relevantly impact the result). Probabilistic behavioural types could reduce the amount of concurrency to verify, and speed up the verification. Kirstin Peters has related work on the topic `https://doi.org/10.1007/978-3-030-32505-3_12`.

Jorge: on the topic of session-based execution optimisation, there is also related work by Luis Caires and Bernardo Toninho at ESOP 2024. Still, it is not clear whether bringing these optimisations in a probabilistic setting (e.g. to optimise analysis against PRISM model checking) would require probability annotations, maybe just having session-structured interaction is enough We discuss a leader election protocol modelled in PRISM, and the properties that are verified (e.g. leader eventually elected with probability 1), and queries (i.e. computing the probability that a property eventually holds).

Everyone: the most popular probabilistic functional programming system should be Anglican `https://probprog.github.io/anglican/`. Recent work on quantum computing: `https://doi.org/10.1145/3632885`.

Emanuele: there are works proposing type systems for compositional abstraction for reasoning about the privacy of programs, with programs able to add noise to the computation (with some given probability annotations). eg `https://doi.org/10.1145/3009837.3009884`. We may do something similar for concurrency, e.g. prove that deadlocks may happen or not with some probability, depending on some probabilistic information.

Kirstin Peters: I see two lines: Adding probabilities to session types: do we gain anything from it? Take a program LTS with probabilities as input, but session types have no probabilities (a bit like probabilistic lambda calculus): how would it work?

Emilio: what if you put probabilities on the payload carried by messages? E.g. I may send an integer with probability p, or a string with probability 1-p? Can we get something?

Jorge, Alceste: mention work by Padovani at al. on fair termination; uses non-deterministic choices, what if they are probabilistic? Emanuele believes that even in this case it may not be necessary to annotate types with probabilities (or probability intervals) to achieve probabilistic results, because e.g. the work on probabilistic privacy does not do it.

Conclusion

The conclusion of the breakout group is that there is no conclusion: there is no clear direction for having behavioural types with probabilities, so multiple attempts at exploring different directions are necessary. It is possible that specific problems may suggest solutions that naturally lead to probability-annotated session types, but that is not clear yet.

4.7 Open World Choreographies

Andrew Hirsch, Lukasz Ziarek, Marco Peressotti, Malte Viering, Raymond Hu, and Roland Kuhn

What are "open" choreographies and possible directions

AH: started with a short introduction to the principles of choreographic programming and what makes the current state of the art "closed world".

MP: Choreographies are global descriptions of the interactions of a distributed system but they assume that all participants in the system are fully described by the choreography.

AH: Focus on partially specified choreographies. Starting point to fix the discussion on a concrete example: DB-WS-Client(s) where a choreography specifies only DB and WS pushing assumptions on the Client(s) behaviour.

RK: Monitors at the boundary seem to be able to address several concerns raised in the example.

AH, LK: Suggested using some form of MPST to describe the interaction across the boundary. RK, MP: MPST are not expressive enough there is a need to identify a participant across multiple endpoints across the boundary.

MP: Asked to clarify the limitations of `https://doi.org/10.1007/978-3-642-40184-8_30` or other existing works.

MV: A safe approach could be to synthesise "minimal" realisations for the unspecified part and use these to ensure properties.

RK: Example of "pluggable" choreographies used.

AH, MP: ChorLambda, Choral, and Pirouette provide higher-order composition of choreographies but still the end artefact requires full specification.

MP: Initial work on extension of ChorLambda and Choral with types with existential and universal quantification over roles. These allow to specify code for roles that join choreographies dynamically.

RH: Overall, either the choreographies allow for "step inside the boundary" or a monitor.

RH, MP: pointers to works that added compositionality to models initially "closed". For instance compositional Petri Nets, Milner's Bigraphs and IPOs, Graph rewriting DPOs.

Conclusions

From the discussion we identified to overall approaches:

1. Underspecified choreographies that identify participants without providing a full implementation of their behaviour in the choreography. The boundary between fully specified and underspecified requires some form of contract and runtime checks.
2. Choreographies provide mechanisms for adding new participants dynamically. In combination with support for cohorts of roles with uniform behaviour this can allow e.g. program systems with peers that can dynamically join or leave. An approach currently under exploration in Choral and ChorLambda are role quantifiers.

4.8 Mechanisation of Behavioural Types

Jesper, Luis, Kirstin, Robbert, Jonas, Alceste, Ralf

Jesper: defining a logic into a proof assistant is hard. Because of adequacy.

Again: depending on what you need to do, you need to pick the right proof assistant and techniques

Binders seem to be the main issue. Robert: avoid binders, but not clear how.

Discussion on nominal => Jesper thinks it's by far the best way to deal with binders, if a deep embedding is the goal.

Luis: why hard in pi-calculus and easy in lambda. Coq can do it for you. In a functional language.

Ok, they are talking about embedding, deep vs shallow. This can make a difference.

- Deep embedding, hard.
- Shallow embedding.

==> Design a framework for working on session type mechanisation. Using Iris as an example.

Luis: the use of binders is different from pi to lambda. Substitution: names for names only.

Jesper & Kirstin: use nominal for deep embedding.

Robbert explains shallow vs deep. A shallow embedding allows to make the language modular (you can add new stuff).

Jesper: use nominal!

Kirstin: there is a need to reach out to people who are not expert.

Robbert: mechanise for confidence and mechanise for improving the proof assistant.

Jonas: having tutorial material.

Message: write about experiences, people can reuse it.

Ideally we should have a framework where to mechanise behavioural types (and linear stuff).

Robbert: If you want to build something more traditional, then somebody needs to make an effort, and perhaps nominal is a good way of doing it. There is also the Iris approach (shallow embedding) – this is great but (Kirstin says) it is hard to use (given the decoupling from syntax). (Kirstin says) An explanation of how the embedding allows to model Process calculi and about the advantages in proving in combination with a tutorial for how to get things working might allow also non-experts maybe without an understanding of the relation to logic to use this framework.

Robbert: third approach, intrinsic typing – (Agda?)

Alceste: Concurrency benchmark.

Ralf: The benchmark is excluding semantic approaches (like logical relations).

4.9 Dependent Session Types

Introduction round: what do we want out of the session?

- session types are important (send number N, followed by N messages where $N = 5$)
- systems: level-dependent session types, Actris
- refinement types: not allowing the highest bidder to bid again until a yet higher bid has been submitted
- what are dependent types in the context of session types? type families? type-level computation? indexed types?
- dependent types are functions from values to types, where values should be behaviours (which can encode arbitrary data)
- should you be able to depend on messages that you haven't observed?
- tracking dependency structures in multi-party sessions is non-trivial (it is intuitive in binary sessions, however)
- logistics auction: only the winning participant can continue the process after the auction

Example discussion:

- Simon: sending JSON, not labels
  - Actris can do this by using Coq "if/else" on the type-level
- Roland: factory logistics bidding
  - Brigitte proposes «Message-aware session types» (ESOP24) as a possible solution
  - importantly, the continuation may depend on what messages the process got from other channels prior
- Jonas: round-trip example ($A \to B \to C \to A$)

- bind variable x on the first send, then refer to it for the other sends to ensure that all send the same value
- the problem is that $B \to C$ has a binder (by duality) at B, but none at C
- solution might be to trace who will learn of the value and permit those roles to have binders when they receive it
- even more complicated is when only a location is sent around pointing to a linear resource
- the setting here specifically is that the global type is not known, the endpoint types are known and need to be combined to figure out whether the composition is well-typed (basically recovering a global type)

Participants

- Sören Auer
TIB – Hannover, DE
- Piero Andrea Bonatti
University of Naples, IT
- Juan Cano de Benito
Technical University of Madrid, ES
- Andrea Cimmino
Polytechnic University of Madrid, ES
- Michael Cochez
VU Amsterdam, NL
- John Domingue
The Open University – Milton Keynes, GB
- Michel Dumontier
Maastricht University, NL
- Nicoletta Fornara
University of Lugano, CH
- Irini Fundulaki
FORTH – Heraklion, GR
- Sandra Geisler
RWTH Aachen, DE
- Anna Lisa Gentile
IBM Almaden Center – San Jose, US
- Paul Groth
University of Amsterdam, NL
- Peter Haase
Metaphacts GmbH – Walldorf, DE
- Andreas Harth
Fraunhofer IIS – Nürnberg, DE
- Olaf Hartig
Linköping University, SE
- James A. Hendler
Rensselaer Polytechnic Institute – Troy, US
- Aidan Hogan
University of Chile – Santiago de Chile, CL
- Katja Hose
TU Wien, AT
- Luis-Daniel Ibáñez
University of Southampton, GB
- Ryutaro Ichise
Tokyo Inst. of Technology, JP
- Ernesto Jiménez-Ruiz
City – University of London, GB
- Timotheus Kampik
Umeå University, SE & SAP Berlin, DE
- Sabrina Kirrane
Wirtschaftsuniversität Wien, AT
- Manolis Koubarakis
University of Athens, GR
- Luis C. Lamb
Boeing Research & Technology – Seattle, US
- Julian Padget
University of Bath, GB
- Harshvardhan J. Pandit
Dublin City University, IE
- Heiko Paulheim
Universität Mannheim, DE
- Axel Polleres
Wirtschaftsuniversität Wien, AT
- Philipp D. Rohde
TIB – Hannover, DE
- Daniel Schwabe
Rio de Janeiro, BR
- Oshani Seneviratne
Rensselaer Polytechnic – Troy, US
- Elena Simperl
King's College London, GB
- Chang Sun
Maastricht University, NL
- Aisling Third
The Open University – Milton Keynes, GB
- Ruben Verborgh
Ghent University, BE
- Maria-Esther Vidal
TIB – Hannover, DE
- Sonja Zillner
Siemens AG – München, DE