RESEARCH CENTRE

Inria Centre at Université Côte d'Azur

IN PARTNERSHIP WITH:

Université de Bologne (Italie)

2024 ACTIVITY REPORT

Project-Team OLAS

Operational, Logical, and Algebraic foundations for Software systems

DOMAIN

Algorithmics, Programming, Software and Architecture

THEME

Proofs and Verification

Inria

Contents

Project-Team OLAS

Creation of the Project-Team: 2023 October 01

Keywords

Computer sciences and digital sciences

A1.3. – Distributed Systems

A1.4. – Ubiquitous Systems

A2. – Software

A2.1. – Programming Languages

A2.1.1. – Semantics of programming languages

A2.1.4. – Functional programming

A2.1.6. – Concurrent programming

A2.1.7. – Distributed programming

A2.4. – Formal method for verification, reliability, certification

A2.4.1. – Analysis

A2.4.3. – Proofs

A7. – Theory of computation

A7.2. – Logic in Computer Science

Other research topics and application domains

B6.1. – Software industry

B6.3. – Network functions

B6.4. – Internet of things

B9.5.1. – Computer science

1 Team members, visitors, external collaborators

Research Scientist

- Martin Avanzini [INRIA, Researcher]

Faculty Members

- Davide Sangiorgi [Team leader, UNIV BOLOGNE, Professor]
- Ugo Dal Lago [UNIV BOLOGNE, Professor]
- Saverio Giallorenzo [UNIV BOLOGNE, Associate Professor]
- Ivan Lanese [UNIV BOLOGNE, Associate Professor]
- Gianluigi Zavattaro [UNIV BOLOGNE, Professor]

Post-Doctoral Fellows

- Vikraman Choudhury [UNIV BOLOGNE]
- Zeinab Galal [UNIV BOLOGNE, Post-Doctoral Fellow, from Jul 2024]
- Alexis Ghyselen [UNIV BOLOGNE]

PhD Students

- Andrea Colledan [UNIV BOLOGNE]
- Giuseppe De Palma [UNIV BOLOGNE]
- Pietro Lami [UNIV BOLOGNE]
- Sourabh Pal [UNIV BOLOGNE]
- Matteo Trentin [UNIV BOLOGNE]

Administrative Assistant

- Christine Claux [INRIA]

External Collaborators

- Maurizio Gabbrielli [UNIV BOLOGNE]
- Daniel Hirschkoff [ENS Lyon]
- Cosimo Laneve [UNIV BOLOGNE]
- Simone Martini [UNIV BOLOGNE]

2 Overall objectives

Software is more and more transforming our daily lives. However, it is also becoming more and more complex. This raises tremendous challenges when it comes to ensuring that software works correctly and efficiently. Correctness is about ensuring that a program satisfies expected requirements. Efficiency is about reducing the resource usage of the runs of a program without affecting the overall behavior.

In OLAS, we study models and techniques for reasoning about the correctness and efficiency of modern software systems. We focus on languages and formalisms that are higher-order, in that they allow, either syntactically or semantically, the representation of general functions, including functions that take other functions as arguments. A distinctive feature of higher-order languages is open-endedness: the visibility that a term has of its environment may change over time, because the interaction of the term with its environment will affect the future capabilities of interactions. Another feature is abstraction, both on data and on behavior. Higher-order constructs are important in modern high-level programming languages. For instance, software is typically open-ended because it is connected to the Internet; and abstraction is important for writing concise code and for enhancing modularity. Indeed, modern mainstream programming languages normally include higher-order constructs.

While the higher-order languages that we investigate may present even strikingly different features (e.g., constructs for concurrency, distribution, probability), we follow a common and unifying methodology. The first and most important aspect of this methodology is given by the kind of techniques employed, namely *operational* techniques—whereby the meaning of systems is expressed in terms of the dynamics of programs, i.e., how the programs evolve in a stepwise fashion—complemented with mathematical *logic* (e.g., type systems) and *algebraic* reasoning. The other major unifying aspect of our methodology has to do with *models*: we first experiment on models, aiming at identifying the ones with solid logical and algebraic roots, and then move up towards languages and software systems.

We believe that such a methodology is well adapted to reason compositionally and under the 'open world' assumption of higher-order languages. Compositionality is a central principle in our approaches because of the complexity of software systems.

3 Research program

3.1 Models

The topic and objective of OLAS is the development of semantics, concepts, techniques, and possibly also linguistic constructs and tools, for specifying and reasoning about higher-order software systems. Fundamental to these activities is *modeling*. Therefore designing, developing and studying appropriate computational models is a central activity in OLAS. The models are used to formalize and verify important computational properties of the systems, as well as to propose new linguistic constructs. The models we study have their roots in *algebra* and *logic*, following the tradition stemming from the λ-calculus and process calculi. As such, they well address compositionality—a central property in our approach to problems. The use of foundational models inevitably leads to opportunities for developing the foundational models themselves, with particular interest for issues of expressiveness and for the transplant of concepts or techniques from a model to another one.

3.2 Behavioral equivalences and metrics

Behavioral equivalences equate processes that "behave in the same way" in all contexts, that is, under all environments in which they could possibly be used. Equivalence is particularly useful as a tool for justifying program transformations, ("we can validly replace P by Q because they have the same behavior in all contexts"), performed either by programmers during system development or by optimizing phases of compilers. Such transformations require equivalence relations to be congruences, i.e. preserved by all operators of the underlying languages, a property that is also fundamental to perform compositional reasoning on complex systems.

A useful refinement of behavioral equivalence is represented by metrics. Metrics allow one to be more informative about the comparison between two systems, so to reveal "how different" two programs are. Two systems may not be exactly behaviorally equal, but they may still be "similar", and the difference

between them may be acceptable. For instance, a program may approximate another one by performing less precise real number calculations but may thus consume less energy. The use of metrics, in place of ordinary behavioral equivalence, is particularly relevant for languages that capture quantitative aspects such as probabilities.

3.3 Types

The reason why we enhance our operational and algebraic techniques with logical formalisms such as types is that types appear to offer a good trade-off between expressiveness and amenability to efficient verification and validation techniques. By showing that a program has a certain type one may be capable of guaranteeing certain desirable behavioral properties, such as termination (the property that a program will not run forever). Types may also provide formal descriptions of the interaction protocols (the dialogues) among the components of system.

3.4 Proof Theory

Proof theory is a branch of mathematical logic which has been proved to have many applications to computer science. One paradigmatic example is Girard's Linear Logic: defined more than thirty years ago, it has been applied to several distinct domains in computer science, from programming language theory to security, from automatic theorem proving to computational complexity. From the perspective of OLAS, Linear Logic offers some elegant tools for resource-control, which we often use also as a mean for enhancing type systems. We use such techniques for expressing bounds on different kinds of resources, both spatial (having to do with the memory needs of a program) and temporal. The bounds may also formalize different kinds of analysis, including a worse case complexity, an average case complexity, as well as forms of tail probability (informally, measuring how far a certain complexity can spread from its mean, thus indicating the likelihood of occurrence of certain behavioral anomalies in a system).

4 Application domains

OLAS targets models and techniques for reasoning about higher-order software systems. These systems are found in different application domains. In OLAS we are particularly interested in the following ones:

- *Concurrent and distributed systems*, including *service-oriented systems* (e.g., *microservices*, and *serverless architectures*);
- *Bayesian languages* (roughly, probabilistic programs that feature, besides sampling, also operations for conditioning via observations or scoring);
- *Quantum programming* (where resource control is of paramount importance);
- *Cryptographic systems*, in particular cryptographic proof assistants.

While these areas are very wide in scope, the aspects that are of interest in OLAS represent a relatively small part of the areas themselves. As an example, we study boundaries in probabilistic program verification with relevance to the verification of cryptographic systems, itself a narrow subfield of cryptography.

The unifying aspects of our approach – the focus on higher-order languages, notably reasoning techniques which stem from the common basis of operational semantics supported by algebras and logics — favors the transfer of techniques and ideas between languages developed for different application domains.

5 Social and environmental responsibility

Our research activities impact the environment negatively, primarily due to the need to attend scientific events. To limit our impact, we give preference to remote participation or environment friendly modes of transport, such as train, when feasible.

6 New software, platforms, open data

6.1 New software

6.1.1 Corinne

Keywords: Choreography automata, Communicating finite state machines

Scientific Description: Corinne relies on the theory of choreography automata, which is described in:

Franco Barbanera, Ivan Lanese, Emilio Tuosto: Choreography Automata. COORDINATION 2020: 86-106

Franco Barbanera, Ivan Lanese, Emilio Tuosto: Composition of choreography automata. CoRR abs/2107.06727 (2021)

Functional Description: Choreography automata (c-automata) are finite state automata whose transitions are labelled with interactions of the form A -> B : m, representing a communication in which participant A sends a message (of type) m to participant B, and participant B receives it. Corinne allows one to display c-automata represented in the dot format, and: (a) project them on communicating finite state machines representing the behavior of single participants, (b) compute a product c-automaton corresponding to the concurrent execution of two c-automata, (c) synchronize two participants of a c-automaton transforming them into coupled gateways, and (d) check well-formedness conditions ensuring that the system of participants obtained via projection behaves well.

Release Contributions: Version 4.0 of Corinne improves the usability of the tool and refines the check for connectedness, allowing the so called late join, when a participant participate to a choreography only in some branches, after having been contacted by some other participant.

URL: https://github.com/lanese/corinne-3

Publication: hal-03468190

Contact: Ivan Lanese

Partner: Gran Sasso Science Institute

6.1.2 CauDEr

Name: Causal-consistent Debugger for Erlang

Keywords: Debug, Reversible computing

Scientific Description: The CauDEr reversible debugger for Erlang is based on the theory of causal-consistent reversibility, which states that any action can be undone provided that its consequences, if any, are undone beforehand. This theory relies on a causal semantics for the target language, and can be used even if different processes have different notions of time. Replay is based on causal-consistent replay, which allows one to replay any future action, together with all and only its causes.

Functional Description: CauDEr is a debugger allowing one to explore the execution of concurrent and distributed Erlang programs both forward and backward. Notably, when going backward, any action can be undone provided that its consequences, if any, are undone beforehand. The debugger also provides commands to automatically find relevant past actions (e.g., send of a given message) and undo them, including their consequences. Forward computation can be driven by a log taken from a computation in the standard Erlang/OTP environment. An action in the log can be selected and replayed together with all and only its causes. The debugger enables one to find a bug by following the causality links from the visible misbehavior to the bug.

URL: https://github.com/mistupv/cauder

Publications: hal-03005383v1, hal-01912894v1, hal-02313745v1

Contact: Ivan Lanese

Participant: Ivan Lanese

Partner: Universitat Politècnica de València

6.1.3 QuRA

Name: Quipper Resource Analysis

Keywords: Static analysis, Quantum programming, Programming language

Functional Description: QuRA is a static analysis tool for the verification of the resource consumption of quantum circuit description programs. QuRA takes as input a program written in a variant of Quipper called PQ and outputs two things: a type for the program and an upper bound to the size of the circuit it will build. QuRA is capable of estimating global size metrics of circuits, such as their width and gate count, as well as size metrics that are local to individual wires, such as depth. It does so in an almost fully automatic fashion, with few annotations, thanks to the use of SMT solvers.

URL: https://github.com/andreacolledan/qura

Contact: Andrea Colledan

Participants: Andrea Colledan, Ugo Dal Lago, Niki Vazou

Partner: The IMDEA Software Institute

6.1.4 Ranflood

Keywords: Cybersecurity, Ransomware

Functional Description: Ranflood is an anti-crypto-ransomware tool that counteracts the encryption phase by flooding specific folders (e.g., the attacked location, the user's folders) with decoy files and helps users recover their files after an attack.

This action has a twofold effect.

First, it confounds the genuine files of the user with decoy files, causing the attacking ransomware to waste time on sacrificial data rather than on the victim's genuine files.

Second, the file-flooding IO-intensive activity contends with the ransomware to access the victim's computing resources, further slowing down the attack of the malware.

Release Contributions: In 2024 Ranflood had two main releases, 0.6-beta and 0.7-beta.

Release 0.6-beta includes new code utilities and features, two new Ranflood flooding strategies based on Shamir's Secret Sharing, and a related restore routine for the FileChecker.

Release 0.7-beta includes an HTTP and a WebSocket server that clients can use to connect to the Ranflood daemon (useful also to support the connection with the new Ranflood webclient and additional functionalities for a more fine-grained control of the daemon's processes).

URL: https://ranflood.netlify.app/

Contact: Saverio Giallorenzo

6.1.5 APP

Name: Allocation Priority Policies

Keywords: Serverless, Scheduling, Cloud computing, Optimisation

Scientific Description: APP addresses the problem of function execution scheduling, i.e., how to schedule the execution of Serverless functions to optimise their performance against some user-defined goals, by specifying policies that inform the scheduling of function execution.

Functional Description: Serverless computing is a Cloud development paradigm where developers write and compose stateless functions, abstracting from their deployment and scaling.

APP is a declarative language of Allocation Priority Policies to specify policies that inform the scheduling of Serverless function execution to optimise their performance against some user-defined goals.

APP is currently implemented as a prototype extension of the Serverless Apache OpenWhisk platform.

Release Contributions: 0.1: APP first release introduced the APP declarative language used to write scheduling policies in serverless platform. The first release also introduced support for the OpenWhisk platform with an alternative Load Balancer for APP scripts.

0.1-tapp: This release introduces an extension of APP, named tAPP (topology-aware Allocation Priority Policies), that adds the capability to declare topological contraints on function-scheduling. An implementation on top of the OpenWhisk platform is also provided.

0.1-aapp: Another extension, dubbed aAPP (affinity-aware Allocation Priority Policies), that adds the capability to define affinity and anti-affinity constraints on 2 or more functions, together with an updated implementation on OpenWhisk.

URL: https://github.com/giusdp/openwhisk

Contact: Saverio Giallorenzo

Partner: University of Southern Denmark

6.1.6 JOLIE

Name: Jolie

Keyword: Microservices

Scientific Description: Jolie enforces a strict separation of concerns between behaviour, describing the logic of the application, and deployment, describing the communication capabilities. The behaviour is defined using the typical constructs of structured sequential programming, communication primitives, and operators to deal with concurrency (parallel composition and input choice). Jolie communication primitives comprise two modalities of interaction typical of Service-Oriented Architectures (SOAs), namely one-way (sends an asynchronous message) and request-response (sends a message and waits for an answer). A main feature of the Jolie language is that it allows one to switch among many communication media and data protocols in a simple, uniform way. Since it targets the field of SOAs, Jolie supports the main communication media (TCP/IP sockets, Bluetooth L2CAP, Java RMI, and Unix local sockets) and data protocols (HTTP, JSON-RPC, XML-RPC, SOAP and their respective SSL versions) from this area.

Functional Description: Jolie is a language for programming service-oriented and microservice applications. It directly supports service-oriented abstractions such as service, port, and session. Jolie allows to program a service behaviour, possibly obtained by composing existing services, and supports the main communication protocols and data formats used in service-oriented architectures. Differently from other service-oriented programming languages such as WS-BPEL, Jolie is based on a user-friendly Java-like syntax (more readable than the verbose XML syntax of WS-BPEL).

Moreover, the kernel of Jolie is equipped with a formal operational semantics. Jolie is used to provide proof-of-concept implementations around Olas activities.

Release Contributions: Release 1.12 is largely a stabilisation release, focusing on bugfixes and improved library support.

The main changes include: - New transaction API offered by the database library service for fine-grained transaction management. - Improved support for OpenAPI v3. - Re-introduced DELETE body requests. - Updated openapi2jolie and jolie2openapi for: - Better compatibility and error handling, including support for "anyOf" clauses and more HTTP status codes. - Adjusted the handling of regular expressions and JSON schema generation to meet OpenAPI specifications. - Modified how HTTP status codes are parsed and handled, now recognising all 2xx codes as successful. - Addressed various bugs and edge cases in HTTP and OpenAPI integrations. - Improved handling of concurrent HTTP messages, with several bugfixes and support for content negotiation with concurrent HTTP between Jolie services. - Made numerous improvements and bugfixes in database handling. A new connection pool implementation provides improved performance for transactions. - Expanded internal documentation for the Metadata and InvalidIdException classes. - Numerous low-level optimizations and code cleanups for enhanced performance and maintainability.

URL: http://www.jolie-lang.org/

Contact: Saverio Giallorenzo

Participants: Claudio Guidi, Fabrizio Montesi, Maurizio Gabbrielli, Saverio Giallorenzo, Ivan Lanese, Stefano Zingaro

6.1.7 FunLess

Keywords: Serverless, WebAssembly

Functional Description: FunLess is a Function-as-a-Service (FaaS) platform that, unlike traditional FaaS solutions that rely on resource-heavy containerisation technologies, employs WebAssembly (Wasm) as its runtime environment. Using Wasm ensures lightweight execution, reduced cold starts, and enhanced portability, enabling functions to run seamlessly on diverse hardware architectures (particularly useful in cloud-edge environments). The platform supports a consistent function development and deployment process, allowing developers to write functions in multiple supported languages (Rust, Go, and JavaScript), compile them into Wasm binaries, and execute them across heterogeneous devices without requiring additional runtime adjustments. FunLess allows flexible deployments, supporting both bare-metal installations and optional integration with container orchestration tools like Kubernetes.

URL: https://funless.dev/

Publication: hal-04826377v1

Contact: Matteo Trentin

6.1.8 JoT

Name: Jolie Testing

Keywords: Microservices, Software testing

Functional Description: JoT is a testing framework for Microservice Architectures based on technology agnosticism, a core principle of microservices.

The main advantage of JoT is that it reduces the amount of work for a) testing microservices that use different technology stacks, b) writing tests that involve multiple services, and c) reusing tests

under different deployment configurations or after changing some of its components (e.g., when, for performance, one reimplements a service with a different technology).

In JoT, tests are orchestrators that can both consume or offer operations from/to the microservice architecture under test. The language for writing JoT tests is Jolie, which provides constructs that support technology agnosticism and the definition of terse test behaviors.

Release Contributions: In 2024, JoT underwent some minor updates to fix some issues and integrate its functionalities with container technologies (Docker) and continuous integration platforms (GitHub Actions).

URL: https://github.com/jolie/jot

Contact: Saverio Giallorenzo

Partner: University of Southern Denmark

6.1.9 Choral

Keywords: Choreographic Programming, Compilation, Modularity, Distributed programming

Scientific Description: In essence, Choral developers program a choreography with the simplicity of a sequential program. Then, through the Choral compiler, they obtain a set of programs that implement the roles acting in the distributed system. The generated programs coordinate in a decentralised way and they faithfully follow the specification from their source choreography, avoiding possible incompatibilities arising from discordant manual implementations. Programmers can use or distribute the single implementations of each role to their customers with a higher level of confidence in their reliability. Moreover, they can reliably compose different Choral(-compiled) programs, to mix different protocols and build the topology that they need.

Choral currently interoperates with Java (and it is planned to support also other programming languages) at three levels: 1) its syntax is a direct extension of Java (if you know Java, Choral is just a step away), 2) Choral code can reuse Java libraries, 3) the libraries generated by Choral are in pure Java with APIs that the programmer controls, and that can be used inside of other Java projects directly.

Functional Description: Choral is a language for the programming of choreographies. A choreography is a multiparty protocol that defines how some roles (the proverbial Alice, Bob, etc.) should coordinate with each other to do something together.

Choral is designed to help developers program distributed authentication protocols, cryptographic protocols, business processes, parallel algorithms, or any other protocol for concurrent and distributed systems. At the press of a button, the Choral compiler translates a choreography into a library for each role. Developers can use the generated libraries to make sure that their programs (like a client, or a service) follow the choreography correctly. Choral makes sure that the generated libraries are compliant implementations of the source choreography.

Release Contributions: In 2024, Choral underwent some minor bug fixes and extensions of its standard libraries.

URL: https://www.choral-lang.org/

Contact: Saverio Giallorenzo

Participants: Saverio Giallorenzo, Fabrizio Montesi

Partner: University of Southern Denmark

7 New results

7.1 Service Oriented Computing and Cloud Computing

Participants: Saverio Giallorenzo, Ivan Lanese, Matteo Trentin, Giuseppe De Palma, Gianluigi Zavattaro.

Modern distributed systems face challenges in coordination, resource management, and maintaining consistency across components while ensuring security, performance, and reliability. OLAS contributions focus on developing and refining new programming and architectural paradigms, tools, and platforms to address these challenges, particularly in the context of serverless computing and service orchestration.

Several contributions regard serverless computing and FaaS platforms. In particular, we continue the development of the Allocation Priority Policies (APP) language, dedicated to the declarative specification of per-function scheduling policies, e.g., to enforce the allocation of functions on nodes that enjoy low data-access latencies thanks to proximity and connection pooling. In [28], we formalize the semantics of APP to both provide a necessary substrate useful for the application of formal analysis techniques and, as an unambiguous definition of the language, to drive implementations atop different serverless platforms. Another work on APP regards its extension to topology-awareness [26], i.e., allowing scheduling to consider the topological properties of the available computation nodes. Besides scheduling, we also explore the usage of languages for the programming of serverless functions. In [27], we introduce FunLess, a Function-as-a-Service (FaaS) platform tailored for private edge cloud systems. The peculiarity of FunLess is that it leverages the WebAssembly language (Wasm) as compilation target of the functions and, by extension, as their runtime environment. Thanks to Wasm, FunLess can dispense the usage of container technologies for function invocation, providing increased security, isolation, portability, consistent development and deployment, and a reduced memory footprint that allows functions to run on constrained edge devices.

Another important direction in this area regards the development of tools and verification approaches. In [18], we introduce the Jolie Checker Toolchain (JCT), a plugin-based system for ensuring consistency between implementations and architectural designs in microservices written in the Jolie service-oriented language. Looking at Cloud deployments, in [30], we present an integrated approach for the deployment of microservice-based applications using the ABS language, combining architectural modeling with simulation capabilities through tools like Timed SmartDepl and Zephyrus.

Regarding the application of techniques and technologies developed in OLAS, in [25], we apply Choreographic Automata to a healthcare management case study in Italy's Emilia Romagna region. This implementation revealed important insights about the theoretical requirements of the Corinne tool and highlighted areas for future improvement. In [17], we consider the networking domain and advocate, showcasing it with a case study, the use of choreographic programming to define multiparty workflows in software-defined networking, offering advantages like deadlock freedom and elimination of central orchestrator bottlenecks.

7.2 Reversible Computing

Participants: Ivan Lanese, Vikraman Choudhury, Pietro Lami.

In reversible computing, computation can proceed not only in the standard, forward direction, but also backwards, recovering past states. We have continued the study of reversible computing started in the past years, focusing on causal-consistent reversibility (suitable for concurrent systems) and with debugging as main target application. We continued experimenting using the CauDEr causal-consistent reversible debugger for Erlang [24, 8] considering in particular the table that Erlang provides to link labels to process identifiers [6]. This is particularly interesting since it provides imperative features inside Erlang which is functional. We settled the basis to further extend the fragment of Erlang supported by CauDEr by studying (in the more abstract setting of process calculi) the interplay between reversibility and time [3],

by defining a causal-consistent reversible extension of Hennessy and Regan Temporal Process Calculus. We also started to study the interplay between reversibility and failures that can occur in distributed systems [15]. Failures may lead to a loss of history information, hence one needs to either replicate history information, or use approaches where backward computation leads to states which approximate the desired ones.

Given that reversibility has been applied to many concurrent formalisms and languages, we also refined [7] a general theory developed in previous work allowing one to prove many properties of interest starting from a few axioms which are easier to check on concrete instances. This allows one to recover many results in the literature of concrete reversible frameworks, as well as simply prove new ones by instantiating the general theory.

We also worked in the sequential setting, on the language Janus which is reversible by construction without the need to store history information. We provided a small-step semantics for Janus and proved it equivalent to existing big-step semantics [21]. This is instrumental to the definition of extensions of Janus with concurrency, which are more easily specified using small-step semantics.

7.3 Quantitative analysis

Participants: Martin Avanzini, Andrea Colledan, Ugo Dal Lago, Zeinab Galal, Ivan Lanese.

In OLAS, we are interested in studying *quantitative aspects* of higher-order programs, such as resource consumption, not necessarily only in a pure setting but also when placed in an interactive scenario. One key interest of OLAS are quantitative properties of probabilistic programs, such as the expected runtime, or the probability that some event occurs. This is motivated, in parts, by the role that randomization plays in cryptography, or by the use of randomization as a mean to make algorithms more efficient (on average). Further, due to the rise of quantum models and the prospect of quantum machines, our focus extends also to the analysis of quantitative properties of quantum languages.

In addition to the analysis of properties of individual programs, OLAS is also interested in relational program analysis, e.g., the study of relational properties of programs. More specifically, we are interested in how to evaluate the differences between behaviors of distinct programs, going beyond the concept of program equivalence, but also beyond that of metrics. In this way, approximate correct program transformations can for instance be justified. Relational proofs play also a cruical role in game-based cryptographic proofs.

Below we describe the results obtained by OLAS this year, categorized by application areas.

7.3.1 Deterministic program analysis

Deterministic programs, following the traditional model of computation, can exhibit quantitative effects and in particular give rise to time and space costs, which should for very good reasons be kept under control. In OLAS, we study how this is possible and we thus deal with semantics and verification of deterministic programs.

The class of basic feasible functionals captures feasibility for type-2 functionals, and thus for functional programs that can take (first-order) functions as arguments. We studied how this class related to higher-order term rewriting [11], and in particular with a recently introduced notion of cost–size interpretations for higher-order term rewriting. We then prove that the class of functionals represented by higher-order terms admitting a certain kind of cost–size interpretation is exactly the class of basic feasible functionals.

The λ-calculus is the universally accepted model of functional programming. But can the λ-calculus be considered a reasonable computational model? Can we use it for measuring the space consumption of algorithms? We introduced [1] a new reasonable space cost model for the λ-calculus, based on a variant of the Krivine abstract machine. This is the first cost model which is able to accommodate logarithmic space.

We also provided a categorical framework to combine fixpoints with derivatives by looking at Cartesian differential categories with a fixpoint operator [16], the former being a semantic setting in which concepts

like program differentiation receive a proper treatment. We introduce an additional axiom relating the derivative of a fixpoint with the fixpoint of the derivative. We show how the standard examples of Cartesian differential categories where we can compute fixpoints provide canonical models of this notion.

We finally considered models of linear logic via bicategories, the latter allowing for a finer, more dynamic study, of computation. More specifically, we introduce [5] a bicategorical model of linear logic which is a novel variation of the bicategory of groupoids, profunctors, and natural transformations. Our model is obtained by endowing groupoids with additional structure, called a kit, to stabilize the profunctors by controlling the freeness of the groupoid action on profunctor elements.

7.3.2 Probabilistic program analysis

A class of programs and processes that by their very nature exhibit quantitative effects and are therefore suitable to quantitative analysis are certainly probabilistic programs. In our team, we are interested in studying probabilistic programs from a foundational and from a more applicative viewpoint.

We first of all investigated the termination problem in a calculus of *session types* with *probabilistic choice* operators. In this setting, a whole range of termination properties can be defined, from the weaker almost-sure termination to strong almost-sure termination, passing through positive almost-sure termination. We present two similar session type systems [13] closely related to classical linear logic with exponentials that guarantee the two extremal properties in such range. In both type systems, the definitional overhead that deals with the ensured termination property is kept to a minimum.

In the last years, we have also been interested in laying the foundations for a new approach to bridge logic and probabilistic computation. We published a survey paper [2] summarizing the results we obtained, in which we delineate how one can define extensions of classical and intuitionistic propositional logic with counting quantifiers, that is, quantifiers that measure to which extent a formula is true. The resulting systems admit a natural semantics, based on the Borel σ-algebra of the Cantor space, together with a sound and complete proof system. The systems we introduced, despite their logical nature, tightly relate to probabilistic *computation*: on the one hand, the validity of formulas in prenex form characterizes the corresponding level of Wagner's hierarchy of counting complexity classes. On the other hand, proofs correspond, in the sense of Curry and Howard, to typing derivations for a randomized extension of the λ-calculus.

7.3.3 Quantum program analysis

Quantum computation is a promising and emerging computational paradigm which can efficiently solve problems considered to be intractable on classical computers. However, the unintuitive nature of quantum mechanics poses interesting and challenging questions for the design and analysis of quantum programming languages, with functional correctness and complexity analysis being of prime importance.

One may then wonder to which extent reasoning on quantum programs is more difficult than, say, on classical, probabilistic ones. One way to formally address this question is to place corresponding decision problems (e.g., almost sure termination, finite runtime, weakest pre-condition reasoning) within the *arithmetical hierarchy*, a computability theoretic mean to classify undecidable problems in terms of their relative difficulties. In [10] we have shown that, if we restrict quantum gates to the Clifford+T fragment—known as an approximately universal fragment of quantum mechanics—the above mentioned decision problems can be categorized within the arithmetical hierarchy in the same way as their corresponding problems of probabilistic programs. The main intuition behind this surprising result is that Clifford+T makes it possible to retain quantum states with rational amplitudes, thus restricting the study to a countable domain.

We also managed to extend a minimal Multiparty Session Types (MPST) system with quantum data and operations, this way enabling the specification of quantum protocols [23, 22]. The resulting system of Quantum MPSTs (QMPSTs) provides a formal notation to describe quantum protocols, both at the abstract level of global types, describing which communications can take place in the system and their dependencies, and at the concrete level of local types and quantum processes, describing the expected behavior of each participant in the protocol. Beyond usual communication properties, QMPSTs also allow us to prove that qubits are owned by a single process at any time, capturing the quantum no-cloning and no-deleting theorems.

While specific languages for quantum computing can be defined and exist in the literature, most software development in this field occurs through so-called circuit description languages, in which programs are classical and produce a description of a quantum computation, in the form of a quantum circuit. Since these programs can leverage all the expressive power of high-level classical languages, circuit description languages have been successfully used to describe complex and practical quantum algorithms, whose circuits, however, may involve many more qubits and gate applications than current quantum architectures can actually master. We studied how linear dependent type-and-effect systems can be used to derive parametric upper bounds on the width of the circuits produced by a program [12], and proved that both the standard type safety results and that the resulting resource analysis is correct with respect to a big-step operational semantics.

7.3.4 Applications to cryptography and security

We have also become increasingly interested in topics of *cryptography* and *security*. Concerning *cryptography*, in [9] we have been working on the logical foundations of `EasyCrypt`, a proof assistant designed specifically for the analysis of cryptographic routines. Specifically, we have developed and implemented within `EasyCrypt` the *expectation-based Hoare logic* (`eHL`), a quantitative version of Hoare logic for probabilistic programs with adversarial code. The main distinct feature of this logic is its expressiveness (it is complete in the sense of Cook) whilst retaining usability and modularity. Not confined to cryptographic proofs, as a show-case of expressiveness, we have fully formalized a concrete implementation of *skip-lists* within `EasyCrypt` using `eHL`, a randomized data structure that is exceptionally intricate to analyze formally. In related work [31], a part of the *Dilithium* signature scheme, submitted to the NIST post-quantum cryptography project, has been proven correct with `eHL` in a concise and elegant way.

Concerning investigations related to security, in a recent work [14], we have been re-investigating the role of memory layout randomization (e.g. Kernel Layout randomization as employed in the Linux kernel) as an (in)effective mitigation strategy against attacks in the Spectre era. This work encompasses a revised, formal thread model based on the usual multi-tier memory model employed in modern kernels as well as standard defense mechanisms (such as DEP, SMAP/SMEP, etc), but where an attacker can exploit side-channel info leaks and influence speculative execution. We have proven that attacks comprising memory safety such as the blindsight and related attacks are not only possible due to practical limitations of implementations, but are indeed inherent to the attack vectors underlying modern computer architectures. To remedy, we establish *speculative memory safety* through an adaptation of the notion of constant time, a well established implementation technique to mitigate side-channel attacks. We also investigate program transformations to bridge the gap between memory safety in speculative and classical scenarios.

We also studied [20] a λ-calculus in which all programs can be evaluated in probabilistic polynomial time and in which there is sufficient structure to represent sequential cryptographic constructions and adversaries for them, even when the latter are oracle-based. A notion of observational equivalence capturing computational indistinguishability and a class of approximate logical relations have also been defined, showing that the latter represents a sound proof technique for the former.

Separation logic is a substructural logic which has proved to have numerous and fruitful applications to the verification of programs working on dynamic data structures. In a recent work, Barthe, Hsu and Liao have proposed a new way of giving semantics to separation logic formulas in which separating conjunction is interpreted as exact, *information theoretic* probabilistic independence. There is however a literature on weaker notions of independence which are *computational* in nature, i.e. independence holds only against efficient adversaries and modulo a negligible probability of success. We explored the nature of computational independence in a cryptographic scenario, in view of the aforementioned advances in separation logic. In a recently published paper [19], we show on the one hand that the semantics of separation logic can be adapted so as to account for complexity bounded adversaries, and on the other hand that the obtained logical system is useful for writing simple and compact proofs of standard cryptographic results in which the adversary remains hidden.

7.4 Qualitative semantics

Participants: Ugo Dal Lago, Alexis Ghyselen, Davide Sangiorgi.

In this area, during the past year, our efforts have gone in 2 main directions that have to do with (i) model checking higher-order functions that exhibit algebraic effects, and (ii) with proof techniques for coinduction and induction.

Model checking is a powerful technique for verifying systems and programs which, since the pioneering results by Knapik et al., Ong, and Kobayashi, is known to be applicable to functional programs with higher-order types against properties expressed by formulas of monadic second-order logic. In [4], we study what happens when the program in question, in addition to higher-order functions, also exhibits algebraic effects, such as probabilistic choice or global store. We show that higher-order programs producing algebraic effects can still be viewed as ordinary higher-order recursion schemes. We then move on to consider effect handlers, showing that in their presence the model checking problem is bound to be undecidable in the general case, while it stays decidable when handlers have a simple syntactic form, still sufficient to capture so-called generic effects. Along the way, we hint at how a general specification language could look like, this way justifying some of the results in the literature, and deriving new ones.

'Up-to techniques' represent enhancements of the coinduction proof method and are widely used on coinductive behavioral relations such as bisimilarity. Abstract formulations of these coinductive techniques exist, using fixed-points or category theory. In previous years, we had worked on developing an analogous theory for inductive behavioral relations, i.e., relations defined from inductive observables, such as traces and enriched forms of traces. However the abstract meaning of such 'inductive enhancements', remained unclear. In [29] we review the theory, and then propose an abstract account of it, using fixed-point theory in complete lattices. This can be useful both for understanding the meaning of the enhancements, and for application of the enhancements to new settings.

8 Bilateral contracts and grants with industry

8.1 Bilateral contracts with industry

Ranflood Giallorenzo co-leads a three-year project collaboration, called "Ranflood", started in July 2021, between the "Regional Environmental Protection and Energy Agency" of Emilia-Romagna (ARPAE Emilia-Romagna) and the "Department of Computer Science and Engineering" (DISI) at the University of Bologna. The collaboration regards the development of techniques and software to combat the spread of malware by exploiting resource contention.

Participants: Saverio Giallorenzo.

8.2 Bilateral Grants with Industry

Enhancing Situational Awareness: Real-Time Monitoring and Replanning with Collaborative Unmanned Vehicles and Onboard Sensing Zavattaro and Giallorenzo participate in a two-year project collaboration, called "Enhancing Situational Awareness: Real-Time Monitoring and Replanning with Collaborative Unmanned Vehicles and Onboard Sensing", started in December 2023, between Leonardo Company S.p.A., Thales Alenia Space Italia S.p.A., and the University of Bologna. The collaboration regards the analysis of techniques for managing the development and deployment of software applications in the context of multi-drone systems in disaster scenarios.

Participants: Gianluigi Zavattaro, Saverio Giallorenzo.

9 Partnerships and cooperations

9.1 International initiatives

9.1.1 Associate Teams in the framework of an Inria International Lab or in the framework of an Inria International Program

TCPro3

Title: Termination and Complexity Properties of Probabilistic Programs

Duration: 2019 — 2024

Coordinator: Romain Péchoux (Inria project team Mocqua)

Partners: Inria project team Mocqua, Inria Nancy Grand-Est; University of Innsbruck (Austria)

Inria contact: Romain Péchoux

Summary: Probabilistic languages consist in higher-order functional, imperative languages, and reduction systems with sampling and conditioning primitive instructions. While deep theoretical results have been established on the semantic properties of such languages, applications of termination and complexity analysis are restricted to academic examples so far. The associate team TCPro3 has the aim to contribute to the field by developing methods for reasoning on quantitative properties of probabilistic programs and models. Extensions of these methods to quantum programs will be studied.

9.2 International research visitors

9.2.1 Visits of international scientists

Other international visits to the team

Bruce Kapron

Status professor

Institution of origin: University of Victoria

Country: British Columbia

Dates: July 12-22, 2024

Context of the visit: cryptographic separation logic

Mobility program/type of mobility: research visit

Healdfene Goguen

Status worldwide technical director of Google Cloud Storage

Institution of origin: Google New York

Country: USA

Dates: July 30 - August 1, 2024

Context of the visit: management of distributed resources

Mobility program/type of mobility: research visit

David Basin

Status professor

Institution of origin: ETH Zurich

Country: Switzerland

Dates: November 21-24, 2024

Context of the visit: formal methods for security

Mobility program/type of mobility: research visit

Justin Hsu

Status associate professor

Institution of origin: Cornell University

Country: USA

Dates: November 19-20, 2024

Context of the visit: type systems for the control of errors

Mobility program/type of mobility: research visit

9.2.2 Visits to international teams

Research stays abroad

Martin Avanzini

Visited institution: AIST Tokyo

Country: Japan

Dates: November 25 - December 6, 2024

Context of the visit: Research collaboration with Akihisa Yamada

Mobility program/type of mobility: Research stay

9.3 European initiatives

9.3.1 Horizon Europe

ReGraDe-CS (Reversible Gray Debugging of Concurrent Systems) is a Marie-Curie Postdoc Fellowship started in December 2023 and with a 2 years duration. The fellow is Vikraman Choudhury, supervised by Ivan Lanese. The project tackles gray debugging of concurrent systems. Debugging concurrent systems is notoriously hard. Reversible causal-consistent debugging and replay allows one to log a faulty execution in production environment and replay it in the debugger. There, it can be explored backwards and forwards following causality links from the visible misbehavior to the bug causing it. ReGraDe-CS will extend the approach to gray debugging, namely debugging of systems where only part of the source code is accessible (e.g., the system invokes external services such as Google Maps).

Participants: Vikraman Choudhury, Ivan Lanese.

9.4 National initiatives

9.4.1 DCore

DCore (Causal debugging for concurrent systems) is an ANR project that started on March 2019 and ended in March 2024.

The overall objective of the project was to develop a semantically well-founded, novel form of concurrent debugging, which we call “causal debugging”. Causal debugging comprises and integrates two main engines: (i) a reversible execution engine that allows programmers to backtrack and replay a concurrent or distributed program execution and (ii) a causal analysis engine that allows programmers to analyze concurrent executions to understand why some desired program properties could be violated.

Participants: Ivan Lanese.

9.4.2 PPS

PPS (Probabilistic Programming Semantics) is an ANR PCR project that started on January 2020 and that finished on December 2024.

Probabilities are essential in Computer Science. Many algorithms use probabilistic choices for efficiency or convenience and probabilistic algorithms are crucial in communicating systems. Recently, probabilistic programming, and more specifically, functional probabilistic programming, has become crucial in various works in Bayesian inference and Machine Learning. Motivated by the rising impact of such probabilistic languages, the aim of this project was to develop formal methods for probabilistic computing (semantics, type systems, logical frameworks for program verification, abstract machines etc.) to systematize the analysis and certification of functional probabilistic programs.

Participants: Martin Avanzini, Ugo Dal Lago, Davide Sangiorgi.

9.4.3 SmartCloud

SmartCloud (Smart dynamic adaptivity for cloud computing systems) is an ANR project that started on January 2024 and that will end in July 2027.

It aims at studying techniques for cloud computing systems which can be updated in a smart, dynamic and coordinated way to cope with changing requirements and environment conditions. The project will define a new framework for adaptation combining a model of both platform and application aspects and heuristics for online optimization.

Participants: Gianluigi Zavattaro, Saverio Giallorenzo, Ivan Lanese.

10 Dissemination

10.1 Promoting scientific activities

10.1.1 Scientific events: organisation

Chair of conference program committees

- 1st International Workshop on Choreographic Programming, CP 2024 (S. Giallorenzo)
- 39th Symposium on Logic in Computer Science, LICS 2024 (U. Dal Lago)

Member of the conference program committees

- 36th International Conference on Concurrency Theory, CONCUR 2025 (D. Sangiorgi)
- 33rd EACSL Annual Conference on Computer Science Logic, CSL 2025 (D. Sangiorgi and U. Dal Lago)
- Fundamentals of Software Engineering, FSEN (D. Sangiorgi)
- 28th International Conference on Foundations of Software Science and Computation Structures, FoSSaCS 2025 (U. Dal Lago)
- 16th Conference on Reversible Computation, RC 2024 (I. Lanese)
- 20th International Conference on Formal Aspects of Component Software, FACS 2024 (I. Lanese)
- 1st International Workshop on Choreographic Programming, CP 2024 (I. Lanese)
- 22nd International Conference on Service-Oriented Computing, ICSOC 2024 (G. Zavattaro)
- Combined 31st International Workshop on Expressiveness in Concurrency and 21st Workshop on Structural Operational Semantics , EXPRESS/SOS 2024 (G. Zavattaro)
- 26th International Conference on Coordination Models and Languages (COORDINATION), Artefact Evaluation (S. Giallorenzo)
- IFIP International Conference on Formal Techniques for Distributed Objects, Components and Systems, FORTE (I. Lanese)

Member of conference steering committees

- Symposium on Logic in Computer Science, LICS (U. Dal Lago)
- International Conference on Formal Structures for Computation and Deduction, FSCD (U. Dal Lago)
- International Conference on Reversible Computation, RC (I. Lanese)
- International Federated Conference on Distributed Computing Techniques, DisCoTec (I. Lanese, G. Zavattaro)
- Agility with Microservices Programming workshop (S. Giallorenzo)
- International Conference on Coordination Models and Languages (COORDINATION) (G. Zavattaro).

10.1.2 Journal

Member of the editorial boards

- Logical Methods in Computer Science (U. Dal Lago)
- Mathematical Structures in Computer Science (U. Dal Lago)
- Acta Informatica (U. Dal Lago, D. Sangiorgi)
- TheoretiCS (U. Dal Lago)
- Distributed Computing (D. Sangiorgi)
- Foundations and Trends in Programming Languages (D. Sangiorgi)
- SN Computer Science (D. Sangiorgi)

10.1.3 Invited talks

- 20th Latin American Symposium on Mathematical Logic (D. Sangiorgi)
- 16th Workshop on Games for Logic and Programming Languages (U. Dal Lago)

10.1.4 Leadership within the scientific community

- The Microservices Community is a European-based, international, non-profit organization purposed to promote the development of microservices by bridging research, education, and innovation within and between businesses, universities, organizations and individuals. Members of OLAS have played active roles in the Community since its inception in 2019. The organization includes members from the Innopolis University (Russia), the Dortmund University of Applied Sciences and Arts (Germany), SINTEF and the University of Oslo (Norway), the University of Pisa and the University of Sassari (Italy), WSO2 (U.S.A.), and the Zurich University of Applied Sciences (Switzerland). Lanese and Giallorenzo are respectively part of the research and communication Community groups.
- I. Lanese is chair of the IFIP (International Federation for Information Processing) WG6.1 on Architectures and Protocols for Distributed Systems.
- U. Dal Lago is a member of the scientific council of the Italian Chapter of the EATCS.
- U. Dal Lago is involved since September 2022 in an Italian National initiative called CN HPC, namely a new research center about high-performance computing. U. Dal Lago is responsible for topics related to quantum computing inside the University of Bologna.

10.1.5 Research administration

- M. Avanzini is member of the "comité NICE" for welcoming external researchers (post-docs, “delegation”).

10.2 Teaching

- Ugo Dal Lago
  - ‘Optimization’, 36 hours, 2nd year, University of Bologna, Italy.
  - ‘Cryptography’, 40 hours, 2nd year Master, University of Bologna, Italy.
  - ‘Languages and Algorithms for AI: Machine Learning Theory’, 32 hours, 1st year Master, University of Bologna, Italy.
- Saverio Giallorenzo
  - ‘Programming Languages’, 30 hours, 2nd year Bachelor, University of Bologna, Italy.
  - ‘Network Analysis’, 30 hours, 2nd year Master, University of Bologna, Italy.
- Ivan Lanese
  - “Architettura degli Elaboratori”, 34 hours, 1st year, University of Bologna, Italy.
  - “Computational Methods for Bioinformatics”, 58 hours, 1st year Master, University of Bologna, Italy.
  - “Architetture Software a Microservizi”, 22 hours, 1st year master, University of Bologna, Italy.
- Davide Sangiorgi
  - ‘Operating Systems’, 110 hours, 2nd year undergraduate program, University of Bologna, Italy.
  - ‘Computer abilities’, 16 hours, 2nd year Master in Medicine, University of Bologna, Italy.

- Gianluigi Zavattaro
  - 'Algoritmi e Strutture di Dati', 70 hours, Bachelor in Computer Science, University of Bologna, Italy.
  - 'Scalable and Cloud Programming', 50 hours, Master in Computer Science, University of Bologna, Italy.

10.2.1 Juries

- G. Zavattaro has been member of the evaluation committee of the PhD thesis of Valentino Picotti at the Southern Denmark University (SDU)
- G. Zavattaro has been member of the evaluation committee of the Habilitation à diriger des recherches (HDR) of Simon Bliudze at INRIA, Lille, France.

11 Scientific production

11.1 Publications of the year

International journals

[1] B. Accattoli, U. D. Lago and G. Vanoni. 'Reasonable Space for the Lambda-Calculus, Logarithmically'. In: *Logical Methods in Computer Science* 20.4 (20th Nov. 2024). DOI: 10.46298/lmcs-20(4:15)2024. URL: https://inria.hal.science/hal-04835903 (cit. on p. 11).

[2] M. Antonelli, U. Dal Lago and P. Pistone. 'Towards logical foundations for probabilistic computation'. In: *Annals of Pure and Applied Logic* 175.9 (Oct. 2024), p. 103341. DOI: 10.1016/j.apal.2023.103341. URL: https://inria.hal.science/hal-04835866 (cit. on p. 12).

[3] L. Bocchi, I. Lanese, C. A. Mezzina and S. Yuen. 'revTPL: The Reversible Temporal Process Language'. In: *Logical Methods in Computer Science* Volume 20, Issue 1 (31st Jan. 2024). DOI: 10.46298/lmcs-20(1:11)2024. URL: https://hal.science/hal-04576057 (cit. on p. 10).

[4] U. Dal Lago and A. Ghyselen. 'On Model-Checking Higher-Order Effectful Programs'. In: *Proceedings of the ACM on Programming Languages* 8.POPL (5th Jan. 2024), pp. 2610–2638. DOI: 10.1145/3632929. URL: https://inria.hal.science/hal-04835921 (cit. on p. 14).

[5] M. Fiore, Z. Galal and H. Paquet. 'Stabilized profunctors and stable species of structures'. In: *Logical Methods in Computer Science* Volume 20, Issue 1 (29th Feb. 2024). DOI: 10.46298/LMCS-20(1:17)2024. URL: https://inria.hal.science/hal-04834939 (cit. on p. 12).

[6] P. Lami, I. Lanese, J.-B. Stefani, C. Sacerdoti Coen and G. Fabbretti. 'Reversible debugging of concurrent Erlang programs: Supporting imperative primitives'. In: *Journal of Logical and Algebraic Methods in Programming* 138 (Apr. 2024), p. 100944. DOI: 10.1016/j.jlamp.2024.100944. URL: https://inria.hal.science/hal-04575594 (cit. on p. 10).

[7] I. Lanese, I. Phillips and I. Ulidowski. 'An Axiomatic Theory for Reversible Computation'. In: *ACM Transactions on Computational Logic* 25.2 (16th Apr. 2024), pp. 1–40. DOI: 10.1145/3648474. URL: https://inria.hal.science/hal-04575972 (cit. on p. 11).

Invited conferences

[8] I. Lanese and G. Gössler. 'Causal Debugging for Concurrent Systems'. In: RC 2024 - 16th International Conference on Reversible Computation. Vol. LNCS-14680. Reversible Computation : 16th International Conference, RC 2024, Toruń, Poland, July 4–5, 2024, Proceedings. Torun, Poland: Springer Nature Switzerland, 29th May 2024, pp. 3–9. DOI: 10.1007/978-3-031-62076-8_1. URL: https://inria.hal.science/hal-04610282 (cit. on p. 10).

International peer-reviewed conferences

[9] M. Avanzini, G. Barthe, B. Grégoire, G. Moser and G. Vanoni. 'Hopping Proofs of Expectation-Based Properties: Applications to Skiplists and Security Proofs'. In: *ACM Digital Library*. OOPSLA 2024 -ACM Conference on Object Oriented Programming Systems Languages and Applications. Vol. 8. Proceedings of the ACM on Programming Languages OOPSLA1. Pasadena (CA), United States, 29th Apr. 2024, pp. 784–809. DOI: 10.1145/3649839. URL: https://inria.hal.science/hal-04834120 (cit. on p. 13).

[10] M. Avanzini, G. Moser, R. Péchoux and S. Perdrix. 'On the Hardness of Analyzing Quantum Programs Quantitatively'. In: *Lecture Notes in Computer Science*. ESOP 2024 - 33rd European Symposium on Programming. Vol. Lecture Notes in Computer Science. 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 II LNCS-14577. Luxembourg, Luxembourg, 2024, p. 28. DOI: 10.1007/978-3-031-57267-8_2. URL: https://inria.hal.science/hal-04349874 (cit. on p. 12).

[11] P. Baillot, U. Dal Lago, C. Kop and D. Vale. 'On Basic Feasible Functionals and the Interpretation Method'. In: *Lecture notes in computer science*. FoSSaCS 2024 - 27th International Conference on Foundations of Software Science and Computation Structures. Vol. LNCS-14575. Foundations of Software Science and Computation Structures : 27th International Conference, FoSSaCS 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 II. Luxembourg, Luxembourg: Springer Nature Switzerland, 6th Apr. 2024, pp. 70–91. DOI: 10.1007/978-3-031-57231-9_4. URL: https://hal.science/hal-04743265 (cit. on p. 11).

[12] A. Colledan and U. Dal Lago. 'Circuit Width Estimation via Effect Typing and Linear Dependency'. In: *Lecture notes in computer science*. ESOP 2024 - 33rd European Symposium on Programming. Vol. LNCS-14577. 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 II. Luxembourg, Luxembourg: Springer Nature Switzerland, 2024, pp. 3–30. DOI: 10.1007/978-3-031-57267-8_1. URL: https://inria.hal.science/hal-04836016 (cit. on p. 13).

[13] U. Dal Lago and L. Padovani. 'On the Almost-Sure Termination of Binary Sessions'. In: *ACM Digital Library*. PPDP 2024 - 26th International Symposium on Principles and Practice of Declarative Programming. PPDP '24: Proceedings of the 26th International Symposium on Principles and Practice of Declarative Programming. Milano, Italy: ACM, 9th Sept. 2024, pp. 1–12. DOI: 10.1145/3678232.3678239. URL: https://inria.hal.science/hal-04836087 (cit. on p. 12).

[14] D. Davoli, M. Avanzini and T. Rezk. 'On Kernel's Safety in the Spectre Era (And KASLR is Formally Dead)'. In: CCS '24: ACM SIGSAC Conference on Computer and Communications Security. Salt Lake City, United States: ACM, 14th Oct. 2024, pp. 1091–1105. URL: https://hal.science/hal-04839461 (cit. on p. 13).

[15] G. Fabbretti, I. Lanese and J.-B. Stefani. 'Reversibility with Holes'. In: *Lecture notes in computer science*. RC 2024 - 16th International Conference on Reversible Computation. Vol. LNCS-14680. Reversible Computation : 16th International Conference, RC 2024, Toruń, Poland, July 4–5, 2024, Proceedings. Torun, Poland: Springer, 2024, pp. 69–74. DOI: 10.1007/978-3-031-62076-8_5. URL: https://inria.hal.science/hal-04610283 (cit. on p. 11).

[16] Z. Galal and J.-S. Pacaud Lemay. 'Combining fixpoint and differentiation theory'. In: *ACM Digital Library*. LICS 2024 - 39th Annual ACM/IEEE Symposium on Logic in Computer Science. LICS '24: Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science. Tallinn, Estonia: ACM, 8th July 2024, pp. 1–14. DOI: 10.1145/3661814.3662108. URL: https://inria.hal.science/hal-04834931 (cit. on p. 11).

[17] S. Giallorenzo, J. Mauro, A. Melis, F. Montesi, M. Peressotti and M. Prandini. 'Choreography-Defined Networks: a Case Study on DoS Mitigation'. In: *Lecture notes in computer science*. ICSOC 2024 - 22nd International Conference on Service-Oriented Computing. Vol. LNCS-15405. Service-Oriented Computing : 22nd International Conference, ICSOC 2024, Tunis, Tunisia, December 3–6, 2024, Proceedings, Part II. Tunis, Tunisia, 2024, pp. 243–259. DOI: 10.1007/978-981-96-0808-9_18. URL: https://inria.hal.science/hal-04826429 (cit. on p. 10).

[18] S. Giallorenzo, F. Montesi, M. Peressotti, F. Rademacher, S. Sachweh and P. Wizenty. 'A Toolchain for Checking Domain-and Model-driven Properties of Jolie Microservices'. In: *Lecture notes in computer science*. ICSOC 2024 - 22nd International Conference on Service-Oriented Computing. Vol. LNCS-15405. Service-Oriented Computing : 22nd International Conference, ICSOC 2024, Tunis, Tunisia, December 3–6, 2024, Proceedings, Part II. Tunis, Tunisia, 3rd Dec. 2024. DOI: 10.1007/978-981-96-0808-9_13. URL: https://inria.hal.science/hal-04826442 (cit. on p. 10).

[19] U. D. Lago, D. Davoli and B. Kapron. 'On Separation Logic, Computational Independence, and Pseudorandomness'. In: CSF 2024 6 IEEE 37th Computer Security Foundations Symposium. Vol. 15194. 2024 IEEE 37th Computer Security Foundations Symposium (CSF). Enschede, Netherlands: IEEE, 28th Oct. 2024, pp. 80–95. DOI: 10.1109/CSF61375.2024.00040. URL: https://inria.hal.science/hal-04835958 (cit. on p. 13).

[20] U. D. Lago, Z. Galal and G. Giusti. 'On Computational Indistinguishability and Logical Relations'. In: APLAS 2024 - 22nd Asian Symposium on Programming Languages and Systems. Vol. 15194. Lecture Notes in Computer Science. Kyoto, Japan: Springer Nature Singapore, 28th Oct. 2025, pp. 241–263. DOI: 10.1007/978-981-97-8943-6_12. URL: https://inria.hal.science/hal-04834943 (cit. on p. 13).

[21] P. Lami, I. Lanese and J.-B. Stefani. 'A Small-Step Semantics for Janus'. In: *Lecture notes in computer science*. RC 2024 - 16th International Conference on Reversible Computation. Vol. LNCS-14680. Reversible Computation : 16th International Conference, RC 2024, Toruń, Poland, July 4–5, 2024, Proceedings. Torun, Poland: Springer, 2024, pp. 105–123. DOI: 10.1007/978-3-031-62076-8_8. URL: https://inria.hal.science/hal-04610285 (cit. on p. 11).

[22] I. Lanese, U. Dal Lago and V. Choudhury. 'Towards Quantum Multiparty Session Types'. In: *Lecture notes in computer science*. SEFM 2024 - 22nd International Conference on Software Engineering and Formal Methods. Vol. LNCS-15280. Software Engineering and Formal Methods : 22nd International Conference, SEFM 2024, Aveiro, Portugal, November 6-8, 2024, Proceedings. Aveiro (Portugal), Portugal: Springer Nature Switzerland, 26th Nov. 2024, pp. 385–403. DOI: 10.1007/978-3-031-77382-2_22. URL: https://inria.hal.science/hal-04832740 (cit. on p. 12).

[23] I. Lanese, U. Dal Lago and V. Choudhury. 'Towards Quantum Multiparty Session Types'. In: PLanQC 2025 - Fifth International Workshop on Programming Languages for Quantum Computing. Denver (Colorado), United States, 25th Jan. 2025. URL: https://hal.science/hal-04838281 (cit. on p. 12).

[24] I. Lanese and G. Vidal. 'Reversible Debugging of Erlang Programs in CauDEr'. In: *ACM Digital Library*. DEBT 2024 - 2nd ACM International Workshop on Future Debugging Techniques. DEBT 2024: Proceedings of the 2nd ACM International Workshop on Future Debugging Techniques. Vienna, Austria: ACM, 13th Sept. 2024, pp. 30–31. DOI: 10.1145/3678720.3685319. URL: https://inria.hal.science/hal-04830455 (cit. on p. 10).

[25] S. Pal, I. Lanese and M. Clo. 'Choreographic Automata: A Case Study in Healthcare Management'. In: *Lecture notes in computer science*. COORDINATION 2024 - International Conference on Coordination Models and Languages. Vol. LNCS-14676. Coordination Models and Languages 26th IFIP WG 6.1 International Conference, COORDINATION 2024, Held as Part of the 19th International Federated Conference on Distributed Computing Techniques, DisCoTec 2024, Groningen, The Netherlands, June 17–21, 2024, Proceedings. Groningen, Netherlands: Springer Nature Switzerland, 11th June 2024, pp. 3–19. DOI: 10.1007/978-3-031-62697-5_1. URL: https://inria.hal.science/hal-04828498 (cit. on p. 10).

[26] G. de Palma, S. Giallorenzo, J. Mauro, M. Trentin and G. Zavattaro. 'An OpenWhisk Extension for Topology-Aware Allocation Priority Policies'. In: *Lecture Notes in Computer Science*. COORDINATION 2024 - 26th IFIP WG 6.1 International Conference on Coordination Models and Languages - 26th IFIP WG 6.1 International Conference. Vol. LNCS-14676. Coordination Models and Languages : 26th IFIP WG 6.1 International Conference, COORDINATION 2024, Held as Part of the 19th International Federated Conference on Distributed Computing Techniques, DisCoTec 2024, Groningen, The Netherlands, June 17–21, 2024, Proceedings. Groningen, Netherlands: Springer Nature Switzerland, 11th June 2024, pp. 201–218. DOI: 10.1007/978-3-031-62697-5_11. URL: https://inria.hal.science/hal-04826364 (cit. on p. 10).

[27] G. de Palma, S. Giallorenzo, J. Mauro, M. Trentin and G. Zavattaro. 'FunLess: Functions-as-a-Service for Private Edge Cloud Systems'. In: *IEEE Xplore*. ICWS 2024 - IEEE International Conference on Web Services. 2024 IEEE International Conference on Web Services (ICWS). Shenzhen, China: IEEE, 7th July 2024, pp. 961–967. DOI: 10.1109/ICWS62655.2024.00114. URL: https://inria.hal.science/hal-04826377 (cit. on p. 10).

[28] G. D. Palma, S. Giallorenzo, J. Mauro, M. Trentin and G. Zavattaro. 'Function-as-a-Service Allocation Policies Made Formal'. In: Leveraging Applications of Formal Methods, Verification and Validation. REoCAS Colloquium in Honor of Rocco De Nicola - 12th International Symposium, ISoLA 2024. Vol. 15219. Lecture Notes in Computer Science. Crete, France: Springer Nature Switzerland, 9th Oct. 2025, pp. 306–321. DOI: 10.1007/978-3-031-73709-1_19. URL: https://inria.hal.science/hal-04826399 (cit. on p. 10).

[29] D. Sangiorgi. 'An Abstract Account of Up-to Techniques for Inductive Behavioural Relations'. In: 12th International Symposium Leveraging Applications of Formal Methods Verification and Validation, ISoLA 2024, REoCAS Colloquium in Honor of Rocco De Nicola. Vol. 15219. Lecture Notes in Computer Science. Crete Island, Greece: Springer Nature Switzerland, 9th Oct. 2025, pp. 62–74. DOI: 10.1007/978-3-031-73709-1_5. URL: https://hal.science/hal-04827142 (cit. on p. 14).

Scientific book chapters

[30] L. Bacchiani, M. Bravetti, S. Giallorenzo, J. Mauro and G. Zavattaro. 'Integrated Timed Architectural Modeling/Execution Language'. In: *Active Object Languages: Current Research Trends*. Vol. LNCS-14360. Lecture Notes in Computer Science. Springer Nature Switzerland, 29th Jan. 2024, pp. 169–198. DOI: 10.1007/978-3-031-51060-1_7. URL: https://inria.hal.science/hal-04826354 (cit. on p. 10).

11.2 Cited publications

[31] M. Barbosa, G. Barthe, C. Doczkal, J. Don, S. Fehr, B. Grégoire, Y. Huang, A. Hülsing, Y. Lee and X. Wu. 'Fixing and Mechanizing the Security Proof of Fiat-Shamir with Aborts and Dilithium'. In: *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part V*. Ed. by H. Handschuh and A. Lysyanskaya. Vol. 14085. Lecture Notes in Computer Science. Springer, 2023, pp. 358–389. DOI: 10.1007/978-3-031-38554-4_12 (cit. on p. 13).