RESEARCH CENTRE

Sophia Antipolis - Méditerranée

IN PARTNERSHIP WITH:

Université de Bologne (Italie)

2021
ACTIVITY REPORT

Project-Team

FOCUS

Foundations of Component-based Ubiquitous Systems

IN COLLABORATION WITH: Dipartimento di Informatica - Scienza e Ingegneria (DISI), Universita' di Bologna

DOMAIN

Networks, Systems and Services, Distributed Computing

THEME

Distributed programming and Software engineering

Contents

Project-Team FOCUS

Creation of the Project-Team: 2010 January 01

Keywords

Computer sciences and digital sciences

- A1. – Architectures, systems and networks
- A1.3. – Distributed Systems
- A1.4. – Ubiquitous Systems
- A2.1.1. – Semantics of programming languages
- A2.1.6. – Concurrent programming
- A2.1.7. – Distributed programming
- A2.4.3. – Proofs

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 Scientists

- Martin Avanzini [Inria, Researcher]
- Saverio Giallorenzo [Universita' di Bologna, Researcher]

Faculty Members

- Davide Sangiorgi [Team leader, Universita' di Bologna, Professor, HDR]
- Mario Bravetti [Universita' di Bologna, Associate Professor]
- Ugo Dal Lago [Universita' di Bologna, Professor, HDR]
- Maurizio Gabbrielli [Universita' di Bologna, Professor]
- Ivan Lanese [Universita' di Bologna, Associate Professor]
- Cosimo Laneve [Universita' di Bologna, Professor]
- Simone Martini [Universita' di Bologna, Professor, HDR]
- Gianluigi Zavattaro [Universita' di Bologna, Professor]

Post-Doctoral Fellows

- Aurore Alcolei [Univ de Provence]
- Francesco Gavazzo [Universita' di Bologna]
- Guillaume Geoffroy [Universita' di Bologna, until Sept 21]
- Doriana Medic [Inria, until Jul 2021]
- Paolo Pistone [Universita' di Bologna]
- Stefano Pio Zingaro [Universita' di Bologna]

PhD Students

- Melissa Antonelli [Universita' di Bologna]
- Andrea Colledan [Universita' di Bologna, from Nov 2021]
- Enguerrand Prebet [École Normale Supérieure de Lyon, from Mar 2021]
- Gabriele Vanoni [Universita' di Bologna]
- Adele Veschetti [Universita' di Bologna]

Administrative Assistant

- Christine Claux [Inria]

External Collaborators

- Claudio Guidi [italianaSoftware s.r.l.]
- Daniel Hirschkoff [École Normale Supérieure de Lyon]
- Fabrizio Montesi [University of Southern Denmark]

2 Overall objectives

Ubiquitous Computing refers to the situation in which computing facilities are embedded or integrated into everyday objects and activities. Networks are large-scale, including both hardware devices and software agents. The systems are highly mobile and dynamic: programs or devices may move and often execute in networks owned and operated by others; new devices or software pieces may be added; the operating environment or the software requirements may change. The systems are also heterogeneous and open: the pieces that form a system may be quite different from each other, built by different people or industries, even using different infrastructures or programming languages; the constituents of a system only have a partial knowledge of the overall system, and may only know, or be aware of, a subset of the entities that operate on the system.

A prominent recent phenomenon in Computer Science is the emerging of interaction and communication as key architectural and programming concepts. This is especially visible in ubiquitous systems. Complex distributed systems are being thought of and designed as structured composition of computational units, usually referred to as *components*. These components are supposed to interact with each other and such interactions are supposed to be orchestrated into conversations and dialogues. In the remainder, we will write *CBUS* for Component-Based Ubiquitous Systems.

In CBUS, the systems are complex. In the same way as for complex systems in other disciplines, such as physics, economics, biology, in CBUS theories are needed that allow us to understand the systems, to design or program them, and to analyze them.

Focus investigates the semantic foundations for CBUS. The foundations are intended as instrumental to formalizing and verifying important computational properties of the systems, as well as to proposing linguistic constructs for them. Prototypes are developed to test the implementability and usability of the models and the techniques. Throughout our work, ‘interaction’ and ‘component’ are central concepts.

The members of the project have a solid experience in algebraic and logical models of computation, and related techniques, and this is the basis for our study of ubiquitous systems. 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 Research program

3.1 Foundations 1: Models

The objective of Focus is to develop concepts, techniques, and possibly also tools, that may contribute to the analysis and synthesis of CBUS. Fundamental to these activities is *modeling*. Therefore designing, developing and studying computational models appropriate for CBUS is a central activity of the project. The models are used to formalise and verify important computational properties of the systems, as well as to propose new linguistic constructs.

The models we study are in the process calculi (e.g., the π-calculus) and λ-calculus tradition. Such models, with their emphasis on algebra, address properly compositionality—a central property in our approach to problems. Accordingly, the techniques we employ are mainly operational techniques based on notions of behavioural equivalence, and techniques based on algebra, mathematical logics, and type theory.

3.2 Foundations 2: Foundational calculi and interaction

Modern distributed systems have witnessed a clear shift towards interaction and conversations as basic building blocks for software architects and programmers. The systems are made by components, that are supposed to interact and carry out dialogues in order to achieve some predefined goal; Web services are a good example of this. Process calculi are models that have been designed precisely with the goal of understanding interaction and composition. The theory and tools that have been developed on top of process calculi can set a basis with which CBUS challenges can be tackled. Indeed, industrial proposals of languages for Web services such as BPEL are strongly inspired by process calculi, notably the π-calculus.

3.3 Foundations 3: Type systems and logics

Type systems and logics for reasoning on computations are among the most successful outcomes in the history of the research in λ-calculus and (more recently) in process calculi. Type systems can also represent a powerful means of specifying dialogues among components of CBUS. For instance—again referring to Web services—current languages for specifying interactions only express basic connectivity, ignoring causality and timing aspects (e.g., an intended order on the messages), and the alternative is to use Turing Complete languages that are however undecidable. Types can come in handy here: they can express causality and order information on messages [52, 51, 53], while remaining decidable systems.

3.4 Foundations 4: Implicit computational complexity

A number of elegant and powerful results have been obtained in implicit computational complexity concerning the λ-calculus, where ideas from Linear Logics enable a fine-grained control over computations. This experience can be profitable when tackling issues of CBUS related to resource consumption, such as resource allocation, access to resources, certification of bounds on resource consumption (e.g., ensuring that a service will answer to a request in time polynomial with respect to the size of the input data).

4 Application domains

4.1 Ubiquitous Systems

The main application domain for Focus are ubiquitous systems, i.e. systems whose distinctive features are: mobility, high dynamicity, heterogeneity, variable availability (the availability of services offered by the constituent parts of a system may fluctuate, and similarly the guarantees offered by single components may not be the same all the time), open-endedness, complexity (the systems are made by a large number of components, with sophisticated architectural structures). In Focus we are particularly interested in the following aspects.

- *Linguistic primitives* for programming dialogues among components.
- *Contracts* expressing the functionalities offered by components.
- *Adaptability and evolvability* of the behaviour of components.
- *Verification* of properties of component systems.
- Bounds on component *resource consumption* (e.g., time and space consumed).

4.2 Service Oriented Computing and Cloud Computing

Today the component-based methodology often refers to Service Oriented Computing. This is a specialized form of component-based approach. According to W3C, a service-oriented architecture is “a set of components which can be invoked, and whose interface descriptions can be published and discovered”. In the early days of Service Oriented Computing, the term “services” was strictly related to that of Web Services. Nowadays, it has a much broader meaning as exemplified by the XaaS (everything as a service) paradigm: based on modern virtualization technologies, Cloud computing offers the possibility to build sophisticated service systems on virtualized infrastructures accessible from everywhere and from any kind of computing device. Such infrastructures are usually examples of sophisticated service oriented architectures that, differently from traditional service systems, should also be capable to elastically adapt on demand to the user requests.

5 Highlights of the year

- Saverio Giallorenzo, Fabrizio Montesi, and co-authors have received the “distinguished paper award” for the paper “Multiparty Languages: The Choreographic and Multitier Cases” at 35th European Conference on Object-Oriented Programming (ECOOP) 2021.

- D. Sangiorgi has been elected Fellow of EATCS (European Association for Theoretical Computer Science).
- Fabrizio Montesi and Saverio Giallorenzo collaborated in the curation of “Data Choreographies”, an exhibition at Catch (Kulturværftet, Helsingør, Denmark) of art inspired by research on choreographic programming. The curation of the exhibition entailed close and inspiring work between professionals from computer science and art. The exhibit is aimed at making the audience familiar with scientific ideas, how computer networks influence our daily lives, and how computer scientists are trying to make computer networks reliable and secure.

6 New software and platforms

Let us describe new/updated software.

6.1 New software

6.1.1 JOLIE

Name: Java Orchestration Language Interpreter Engine

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 concepts around Focus activities.

Release Contributions: There are many fixes to the HTTP extension, improvements to the embedding engine for Javascript programs, and improvements to the support tools jolie2java and wsdl2jolie.

News of the Year: In 2021 Jolie moved to version 1.10.x. The main upgrade of this new release is the implementation of a new module system, inspired by Python’s "from ... import ..." mechanism. The general principle is to keep things simple and provide (often automatic) facilities to avoid namespace pollution. The new release also brings along two major overhauls. First, the introduction of a new "service" block which allows users to define service-instantiation parameters. Second, a new syntax for foreign services (services implemented in Java or JavaScript, instead of Jolie), which are now imported and executed by other client services without requiring them to know their implementation technology—this supports more reusable code as changing how a service is implemented does not alter the clients that use it). Minor changes regard the adoption of a snowflake-like mechanism for marking message ids and offer more advanced support for network-wide tracking and improvements/fixes such as: closing the existing operators under commutativity in implicit type casts, extending the HTTP protocol to support arbitrary header read/write, and fixing a concurrency bug of the "for ... in ..." block as well as fixing bugs found in HTTP, fault management, and Java services. Finally, to provide further support for the development of Jolie libraries and architectures, since version 1.10.x, the Jolie libraries for developing Java Services and Jolie code analysis tools are now available on Maven Central.

In 2020 Jolie transitioned to version 1.9.x. The new major release includes a more advanced tracing system, forward and backward documentation primitives, the support for configurations with JSON files (useful, for example, in the development of Docker images), and the extension of the support to Java 11+. The release also includes minor fixes like more complete support for the HTTP(S) protocol, runtime checks for infinite alias loops, and other performance optimisations.

Also the Jolie ecosystem expanded in 2020 with new tools like: Jolier, which aids in the publication of Jolie APIs following the REST style, jolie2openapi, which produces an OpenAPI definition from a Jolie interface, openapi2jolie, which produces a Jolie interface from an OpenAPI definition, jolietraceviewer, which makes use of the updated tracing system to visualise the execution trace of a service, joliedoc, which is a preexisting tool that received major improvements, including support for forward and backward documentation primitives and a facelift to the documents that it generates.

The documentation of the language received a major restyle, both content-wise but also structure-wise, to distinguish between features that belong to different versions of the language — e.g. so that users do not mistakenly assume the presence of some more modern features in older versions of the language and can access consistent documentation dedicated to the version they are using.

Development-wise, the Jolie build system has been ported to Maven and now includes continuous-integration routines to expedite the inclusion of new features and fixes in the main development branch.

During 2019 the Jolie project saw three major actions.

The first action regards the build system used for the development of the language, which has been transitioned to Maven, the main build automation tool used for Java projects. The move to Maven is dictated by two needs. The first is to streamline the development and release processes of Jolie, as Maven greatly helps in obtaining, updating, and managing library dependencies. The second necessity addressed by Maven is helping in partitioning the many sub-projects that constitute the Jolie codebase, reducing development and testing times. Having Jolie as a Maven project also helps in providing Jolie sub-components (as Maven libraries) to other projects. Finally, the move to Maven is set within a larger effort to expedite the inclusion in the main Jolie development branch of contributions by new members of its growing community.

The second action regards the transition to Netty as a common framework to support communication protocols and data formats in Jolie. Netty is a widely-adopted Java framework for the development of network applications, and it was used in 2018 to successfully support several IoT communication protocols and data formats in a Jolie spin-off project, called JIoT. The work in 2019 integrated into the Jolie codebase the protocols and data format developed within the JIoT project and pushed towards the integration of the Netty development branch into the main branch of the Jolie project (i.e., re-implementing using Netty the many protocol and data-formats already supported by Jolie). The Netty development branch is currently in a beta phase and it is subject to thorough in-production tests, to ensure that the behaviour remains consistent with the previous implementation.

The third action regards the development and support for a new official integrated development environment (IDE) for Jolie. Hence, along with the ones already existing for the Atom and Sublime Text text editors, Jolie developers can use the Jolie plugin (based on the Language Server Protocol) for the Visual Studio Code text editor to obtain syntax highlighting, documentation aids, file navigation, syntax checking, semantic checking, and quick-run shortcuts for their Jolie programs.

In addition to the above actions, in 2019 Jolie transitioned through three minor releases and a major one, from 1.7.1 to 1.8.2. The minor releases mainly fixed bugs, improved performance, and included new protocol/data-format functionalities. The major release included a slim-down of the notation for the composition of statements, types definitions, and tree structures, for a terser codebase. Upgrades to 1.8.2 also introduced: timeouts for solicit-response invocations to handle the interruption of long-standing requests, more user-friendly messages from the Jolie interpreter, including easier-to-parse errors and the pretty-printing of data structures, for more effective development and debugging experience.

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

Contact: Fabrizio Montesi

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

6.1.2 NightSplitter

Keyword: Constraint-based programming

Functional Description: Nightsplitter deals with the group preference optimization problem. We propose to split users into subgroups trying to optimize members' satisfaction as much as possible. In a large city with a huge volume of activity information, designing subgroup activities and avoiding time conflict is a challenging task. Currently, the Demo is available only for restaurant and movie activities in the city of Paris.

URL: `http://cs.unibo.it/t.liu/nightsplitter/`

Contact: Maurizio Gabbrielli

6.1.3 CauDEr

Name: Causal-consistent Debugger for Erlang

Keywords: Debug, Reversible computing

Scientific Description: The CauDEr reversible debugger 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 misbehaviour to the bug.

News of the Year: In 2021 a main revision of CauDEr has been concluded. Main novelties include moving the debugger from Core Erlang to Erlang, updating of the interface and integration between user-driven and log-driven forward execution. Support for distribution and related language primitives (e.g., creation of new nodes and remote spawn) has also been integrated.

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

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

Contact: Ivan Lanese

Participant: Ivan Lanese

Partner: Universitat Politècnica de València

6.1.4 SUNNY-AS

Name: SUNNY FOR ALGORITHM SELECTION

Keywords: Optimisation, Machine learning

Functional Description: SUNNY-AS is a portfolio solver derived from SUNNY-CP for Algorithm Selection Problems (ASLIB). The goal of SUNNY-AS is to provide a flexible, configurable, and usable portfolio solver that can be set up and executed just like a regular individual solver.

URL: `https://github.com/lteu/oasc`

Contact: Maurizio Gabbrielli

6.1.5 eco-imp

Name: Expected Cost Analysis for Imperative Programs

Keywords: Software Verification, Automation, Runtime Complexity Analysis, Randomized algorithms

Functional Description: Eco-imp is a cost analyser for probabilistic and non-deterministic imperative programs. Particularly, it features dedicated support for sampling from distributions, and can thereby accurately reason about the average case complexity of randomized algorithms, in a fully automatic fashion. The tool is based on an adaptation of the ert-calculus of Kaminski et al., extended to the more general setting of cost analysis where the programmer is free to specify a (non-uniform) cost measure on programs. The main distinctive feature of eco-imp, though, is the combination of this calculus with an expected value analysis. This provides the glue to analyse program components in complete independence, that is, the analysis is modular and thus scalable. As a consequence, confirmed by our experiments, eco-imp runs on average orders of magnitude faster than comparable tools: execution times of several seconds become milliseconds.

News of the Year: This year, we have refactored the central inference machinery and improved the underlying constraint solver.

URL: `http://www-sop.inria.fr/members/Martin.Avanzini/software/eco-imp/`

Publication: hal-03013544

Contact: Martin Avanzini

6.1.6 PRISM+

Keyword: Stochastic process

Functional Description: PRISM is a probabilistic model checker, a tool for formal modelling and analysis of systems that exhibit random or probabilistic behaviour. We extend the language in order to model the Bitcoin system. The tool now supports three dynamic data types: block, ledger and list. As a consequence, it is now possible to perform simulations and analyse transient probabilities, i.e. probabilities that are dependent on time, for the Bitcoin protocol. It has been used to understand how the system changes during the execution and to analyse the probabilities of reaching an inconsistent state in different settings.

URL: `https://github.com/adeleveschetti/bitcoin-analysis`

Contact: Adele Veschetti

6.1.7 Tquery

Keywords: Ephemeral Data, Microservices, Big data, Querying

Scientific Description: The adoption of edge/fog systems and the introduction of privacy-preserving regulations compel the usage of tools for expressing complex data queries in an ephemeral way—ensuring the queried data does not persist.

Database engines partially address this need, as they provide domain-specific languages for querying data. Unfortunately, using a database in an ephemeral setting has inessential issues related to throughput bottlenecks, scalability, dependency management, and security (e.g., query injection). Moreover, databases can impose specific data structures and data formats, which can hinder the development of microservice architectures that integrate heterogeneous systems and handle semi-structured data.

Tquery is the first query framework designed for ephemeral data handling in microservices. Tquery joins the benefits of a technology-agnostic, microservice-oriented programming language, Jolie, and of one of the most widely-used query languages for semi-structured data in microservices, the MongoDB aggregation framework. With Tquery, users express in a terse syntax how to collect data from heterogeneous sources and how to query it in local memory, defining pipelines of high-level operators. The development of Tquery follows a "cleanroom software engineering process", based on the definition of a theory for querying semi-structured data compatible with Jolie and inspired by a consistent variant of the key operators of the MongoDB aggregation framework.

Functional Description: Tquery is a query framework integrated into the Jolie language for the data handling/querying of Jolie trees.

Tquery is based on a tree-based instantiation (language and semantics) of MQuery, a formalisation of a sound fragment of the Aggregation Framework, the query language of the most popular document-oriented database: MongoDB.

Tree-shaped documents are the main format in which data flows within modern digital systems - e.g., eHealth, the Internet-of-Things, and Edge Computing. Tquery is particularly suited to develop real-time, ephemeral scenarios, where data shall not persist in the system.

Release Contributions: first release

News of the Year: In 2021 Tquery matured in a more stable and performant library for data query in Jolie. It quickly transitioned to version 0.4.10 to include five operators (inspired by MongoDB): "match", "unwind", "project", "group", "lookup". Moreover, since version 0.4, Tquery includes a "pipeline" operator that allows users to express chains of queries that can be performed in a batch fashion, avoiding the overheads of sequences of calls between the clients and the library.

In 2020 Tquery saw its first release, version 0.1-beta, which includes the prototypical implementations of the three operators "match", "unwind", and "project", inspired by the MongoDB Aggregation Framework.

URL: `https://github.com/jolie/tquery`

Contact: Saverio Giallorenzo

Partner: University of Southern Denmark

6.1.8 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: first release

News of the Year: In 2021, the APP language has been extended to be able to capture scheduling scripts of groups of workers with policies that can span more than one scheduling controller.

The first beta release of APP dates to the beginning of 2020. The language is able to express scheduling policies that define the scheduling of Serverless functions by specifying which workers they execute on, in what conditions, and with what recovery strategy.

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

Contact: Saverio Giallorenzo

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: First release

News of the Year: In 2021, Choral received some minor improvements: fixes and extensions of the runtime support library. The main efforts in 2021 include projects on developing plug-in systems for the Choral compiler, to equip it with additional capabilities (e.g., information-flow analysis), and the development of more abstract choreographic languages that compile to Choral programs.

Choral was first released in 2020.

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

Contact: Saverio Giallorenzo

Participants: Saverio Giallorenzo, Fabrizio Montesi, Marco Peressotti

Partner: University of Southern Denmark

6.1.10 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: - project them on communicating finite state machines representing the behaviour of single participants - compute a product c-automaton corresponding to the concurrent execution of two c-automata - synchronize two participants of a c-automaton transforming them into couple gateways - check well-formedness conditions ensuring that the system of participants obtained via projection behaves well

News of the Year: In 2021 Corinne got some bug fixing and refinement, and has been presented in a conference, see:

Simone Orlando, Vairo Di Pasquale, Franco Barbanera, Ivan Lanese, Emilio Tuosto: Corinne, a Tool for Choreography Automata. FACS 2021: 82-92

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

Publication: hal-03468190

Contact: Ivan Lanese

Partner: Gran Sasso Science Institute

7 New results

7.1 Service-oriented and Cloud Computing

Participants: Lorenzo Bacchiani, Mario Bravetti, Saverio Giallorenzo, Ivan Lanese, Fabrizio Montesi, Gianluigi Zavattaro.

7.1.1 Service-Oriented Computing

Session types and behavioural contracts are used for modeling service interaction. Their relationship was studied only in the context of synchronous communication. In [16] we study their relationship under the assumption that processes communicate asynchronously. We show the existence of a fully abstract interpretation of session types into a fragment of contracts that maps session subtyping into binary compliance-preserving behavioural contract refinement. Concerning session subtyping for asynchronous communication, recent work has shown that it is undecidable. To cope with this negative result, the only approaches we are aware of either restrict the syntax of session types or limit communication (by considering forms of bounded asynchrony). In [15] we proceed differently by presenting an algorithm for checking subtyping which is sound, but not complete (in some cases it terminates without returning a decisive verdict). The algorithm is based on a tree representation of the coinductive definition of asynchronous subtyping. In [31] we observe that asynchronous session subtyping rules out candidates subtypes that occur naturally in communication protocols where, e.g., two parties simultaneously send an unbounded amount of messages before removing them from their respective buffers. To address this shortcoming, we study fair compliance over asynchronous session types and we present a sound algorithm that deals with services featuring potentially unbounded buffering. The asynchronous session subtyping algorithm implementations, introduced in the above two papers, have been integrated and improved in a visual tool [49] with an easy-to-use GUI.

Multiparty session types and choreographies are used for modeling multiple services interaction from a global viewpoint: services behaving to such specifications avoid deadlock by construction. In [12] multiparty sessions are considered as open systems by allowing one to compose multiparty sessions by transforming two of their participants into a pair of coupled gateways, forwarding messages between the two sessions. Gateways need to be compatible. We show that the session resulting from the composition can be typed, and its type can be computed from the global types of the starting sessions. Moreover, in [40] we discuss an automata-based representation of choreographies and present Corinne, a tool allowing one to render, compute projections of and compose choreography automata, as well as to check well-formedness conditions.

Besides investigating service composition from a foundational point of view, we have also considered the problem of integrating service orchestration and choreography into mainstream programming languages and software development methodologies. In [37] we show that choreographic languages and multitier languages, that enable programming computation that spans across several tiers of a distributed system, are surprisingly similar. We substantiate our claim by isolating the core abstractions that differentiate the two approaches and by providing algorithms that translate one into the other in a straightforward way. In [36] we present the first preliminary study of how the LEMMA framework (Language Ecosystem for Modeling Microservice Architecture) and the Jolie service oriented programming language can cross-polinate. We establish a common ground for comparing the two technologies in terms of metamodels and discuss practical enhancements that can be derived from the comparison.

7.1.2 Cloud Computing

In the context of Cloud Computing, we have investigated issues related to costs and performances, both at the low-level, of the exploited virtualization technology, and at the high-level, of the (micro)services deployment. In particular, in [21] we present a collection of reference benchmarks for developers on best-performing Virtualization Technologies (VT). To gather our benchmarks in a resource-wise comprehensive and comparable way, we introduce VTmark: a semi-automatic open-source suite that assembles off-the-shelf tools for benchmarking the different resources used by applications (CPU, RAM, etc.). By releasing VTmark , we provide DevOps with an open-source, extendable tool to assess the (resource-wise) costs of VTs. In [30], on the other hand, we develop a novel approach for run-time global adaptation of the deployment of microservice applications, based on the synthesis of architecture-level reconfiguration orchestrations. More precisely, we devise an algorithm for automatic reconfiguration that reaches a target system Maximum Computational Load by performing optimal deployment orchestrations, and we validate the applicability of our approach on a realistic microservice application taken from the literature: an Email Pipeline Processing System. This validation process includes the realization of a novel integrated timed architectural modeling/execution language based on an extension of the actor-based object-oriented Abstract Behavioral Specification (ABS) language.

7.2 Models for Reliability

Participants: Doriana Medic, Ivan Lanese.

Reversibility We have continued the study of reversibility started in the past years. As usual, most of our efforts target concurrent systems and follow the causal-consistent reversibility approach, where any action can be undone provided that its consequences, if any, are undone beforehand. We tackled both theoretical questions and practical applications in the areas of robotics and of debugging of concurrent Erlang programs. From a theoretical point of view, we compared the two reversible extensions of CCS found in the literature, RCCS and CCSK, showing that they give rise to isomorphic labelled transition systems [22]. We also studied an alternative presentation of reversible CCS where the key generation mechanism needed to enable reversibility is made explicit and different notions of non-deterministic choice are compared [29]. Finally, we provided a notion of strong bisimilarity for CCSK able to distinguish forward from backward moves which is a congruence and which coincides with a notion of strong barbed congruence [38]. We also proved correct a number of relevant axioms, enabling axiomatic reasoning on reversible systems.

Reversible debugging of concurrent systems can be tackled using causal-consistent debugging, which prescribes to follow causal dependencies backwards from a visible misbehaviour to the bug causing it. A main technical tool to this end is causal-consistent rollback, allowing one to undo an action far in the past including all and only its consequences. In the setting of Erlang, we studied how to replay a computation inside the debugger using a log from a real execution. We proposed causal-consistent replay, to replay a future action from the log including all and only its causes, and studied its interplay with causal-consistent rollback [23]. We showed that a debugging approach based on these notions faithfully replays misbehaviours occurring in the logged computation. We also extended this approach to cover distribution aspects [34], enabled by primitives to create new nodes at runtime and spawn remote processes.

In a sequential setting, we studied the use of reversibility to ensure reliability in industrial robots [50]. The idea here is that reversibility allows a robot to automatically retries an action in case of failure: since execution is not always precise in the real world, the action may succeed upon retry. However, one has to be careful since not all the actions are reversible (e.g., cutting), and some reverse actions are not dual to the corresponding forward action (e.g., pulling is not the dual of pushing, since it requires gripping). Reversibility can be combined with AI techniques such as planning to program industrial robots.

7.3 Quantitative Analysis

Participants: Melissa Antonelli, Martin Avanzini, Ugo Dal Lago, Guillaume Geoffroy, Francesco Gavazzo, Simone Martini, Paolo Pistone.

In Focus, we are interested in studying probabilistic higher-order programming languages and, more generally, the fundamental properties of probabilistic computation when placed in an interactive scenario, for instance the one of concurrent systems.

One of the most basic but nevertheless desirable properties of programs is of course termination. Termination can be seen as a minimal guarantee about the time complexity of the underlying program. When probabilistic choice comes into play, termination can be defined by stipulating that a program is terminating if its probability of convergence is 1, this way giving rise to the notion of almost sure termination. Termination, already undecidable for deterministic (universal) programming languages, remains so in the presence of probabilistic choice, becoming provably harder. A stronger notion of termination is the one embodied in positive almost sure termination, which asks the average runtime of the underlying program to be finite. If the average computation time is not only finite, but also suitably limited (for example by a polynomial function), one moves towards a notion of bounded average runtime complexity. Over the recent years, the Focus team has established various formal systems for reasoning about (positive) almost sure termination and average runtime complexity, and has even established methodologies for deriving average runtime bounds in a fully automated manner. This trend continued in 2021.

In addition to the analysis of complexity, which can be seen as a property of individual programs, Focus has also been interested, for some years now, in the study of relational properties of programs. More specifically, we are interested in how to evaluate the differences between behaviours of distinct programs, going beyond the concept of program equivalence, but also beyond that of metrics. In this way, only approximate correct program transformations can be justified, while it becomes possible to give a measure of how close a program is to a certain specification.

Below we describe the results obtained by Focus this year, dividing them into three strands.

7.3.1 Randomization in Programming Languages and Logic

Randomized higher-order computation can be seen as being captured by a λ-calculus endowed with a single algebraic operation, namely a construct for binary probabilistic choice. We showed [19] that intersection types are capable of precisely characterizing (positive) almost sure termination inside a single system of types: the probability of convergence of any λ-term can be underapproximated by its type, while the underlying derivation's weight gives a lower bound to the term's expected number of steps to normal form. Noticeably, both approximations are tight: not only soundness but also completeness holds. The crucial ingredient is non-idempotency, without which it would be impossible to reason on the expected number of reduction steps which are necessary to completely evaluate any term. Besides, the kind of approximation we obtain is proved to be optimal recursion theoretically: no recursively enumerable formal system can do better than that.

We have also turned our attention to probabilistic complexity classes. The latter, despite capturing the notion of feasibility, have escaped any treatment by the tools of so-called implicit-complexity. Their inherently semantic nature is of course a barrier to the characterization of classes like BPP or ZPP, but not all classes are semantic. We have introduced [33] a recursion-theoretic characterization of the probabilistic class PP, using recursion schemata with pointers, following the celebrated work by Bellantoni and Cook on safe recursion.

Along these lines, we have also exploited program transformations [11] to systematically extract cost functions, eliminating probabilistic effects. This then effectively allows reasoning about the expected cost of probabilistic, higher-order programs through off-the-shelf, non-probabilistic tools.

We also studied [43] the logic obtained by endowing the language of first-order arithmetic with second-order measure quantifiers. This new kind of quantification allows us to express that the argument formula is true in a certain portion of all possible interpretations of the quantified variable. We showed that first-order arithmetic with measure quantifiers is capable of formalizing simple results from probability theory and, most importantly, of representing every recursive random function. Moreover, we introduce a realizability interpretation of this logic in which programs have access to an oracle from the Cantor space. On a more foundational level, we contributed to the proof-theory of modalities with a general framework for natural deduction systems for normal modal logics [26].

We also studied the foundations of the execution probabilistic programs in the abstract [35], by developing a theory rewriting systems for monadic effects and by isolating general conditions on monads and effect-producing operations that ensure desirable properties of program evaluation (such as confluence and standardization). Remarkably, this has established a new connection between the need to evaluate probabilistic programs to multi-distributions — rather than to traditional distributions — and abstract theories of relation algebras, which actually extends to arbitrary monadic effects, this way going beyond probabilistic nondeterminism.

Finally, we have studied [32] the interaction between resources and computational effects, moving from the observation that the interaction between higher-order functions and (both pure and probabilistic) nondeterminism is deeply resource-sensitive, leading to the validation of program transformations that are simply not available in resource-agnostic settings. We have introduced a new form of monadic transition systems capable of tracking resource production and consumption; on top of that we have defined a resource-sensitive and effectful bisimulation for a linear λ-calculus with algebraic operations and exponential types, which we have proved to coincide with contextual equivalence.

7.3.2 Differential Semantics

We investigated [42] a new approach to the construction of cartesian closed categories of generalized metric spaces. Generalized metrics arise from Lawvere's view of metric spaces as enriched categories and have been widely applied in denotational semantics as a way to measure to which extent two programs behave in a similar, although non equivalent, way. However, the application of generalized metrics to higher-order languages like the simply typed λ-calculus has so far proved unsatisfactory. Our starting point is a quantitative semantics based on a generalization of usual logical relations. Within this setting, we show that several families of generalized metrics provide ways to extend the Euclidean metric to all higher-order types. On the same line, we have also established [20] a new connection between quantitative semantics as given by differential logical relations and incremental computing. Relying on such a connection, we have also extended differential logical relations to calculi with full recursion.

We also described [46] a mathematical structure that can give extensional denotational semantics to higher-order probabilistic programs. It is not limited to discrete probabilities, and it is compatible with integration in a way the models that have been proposed before are not. It is organised as a model of propositional linear logic in which all the connectives have intuitive probabilistic interpretations. In addition, it has least fixed points for all maps, so it can interpret recursion.

7.3.3 On the Space Consumption of Functional Programs

Evaluating higher-order functional programs through abstract machines inspired by Girard's geometry of interaction is known to induce space efficiencies, the price being time performances often poorer than those obtainable with traditional, environment-based, abstract machines. Although families of λ-terms for which the former is exponentially less efficient than the latter do exist, it is currently unknown how general this phenomenon is, and how far the inefficiencies can go, in the worst case. We answered [10] the aforementioned questions formulating four different well-known abstract machines inside a common definitional framework, this way being able to give sharp results about the relative time efficiencies. We also prove that non-idempotent intersection type theories are able to precisely reflect the *time* performances of the interactive abstract machine (IAM in the following), this way showing that its time-inefficiency ultimately descends from the presence of higher-order types.

In another work, we also introduced [28] an intersection type system precisely measuring the *space* consumption of the IAM on the typed term. More specifically, we show that the space consumption of the IAM is connected to a structural modification of intersection types, turning multisets into trees. Tree intersection types lead to a finer understanding of some space complexity results from the literature. They also shed new light on the conjecture about reasonable space: we show that the usual way of encoding Turing machines into the λ-calculus cannot be used to prove that the space of the IAM is a reasonable cost model.

7.4 Verification Techniques

Participants: Mario Bravetti, Ugo Dal Lago, Francesco Gavazzo, Daniel Hirschkoff, Cosimo Laneve, Enguerrand Prebet, Davide Sangiorgi, Gianluigi Zavattaro.

In Focus, we are interested in studying techniques for assessing qualitative properties of concurrent programs. These properties can generally not be measured with a numerical result. Examples of qualitative properties include classical type correctness and behavioural correctness, such as deadlock freedom and the conformance to an abstract protocol. In this axis we mainly use techniques based on (behavioural) types, observational equivalences (e.g., bisimulation and coinduction) and, more recently (stochastic) model checking for analyzing the behaviour of concurrent and/or distributed processes.

Below we describe the results obtained by Focus this year, dividing them into three strands.

7.4.1 Coinductive Techniques

It is known that proving behavioural equivalences in higher-order languages can be hard, because interactions involve terms of the language, which are complex values. In coinductive (i.e., bisimulation-like) techniques for these languages, a useful enhancement is the 'up-to context' reasoning, whereby common pieces of context in related terms are factorised out and erased. In higher-order process languages, however, such techniques are rare, as their soundness is usually delicate and difficult to establish. In [25] we transport the 'up-to context' reasoning onto languages whose bisimilarity and labeled transition system (LTS) go beyond those of first-order models. The approach consists in exhibiting fully abstract translations of the more sophisticated LTSs and bisimilarities onto the first-order ones. This allows us to reuse directly the large corpus of up-to techniques that are available on first-order LTSs.

The only ingredient that has to be manually supplied is the compatibility of basic up-to techniques that are specific to the new languages. We investigate the method on the π-calculus, the λ-calculus, and a (call-by-value) λ-calculus with references.

In [41] we observe that contexts of process calculi, such as the π-calculus, exhibit arbitrary concurrency, making them very discriminating. This may prevent validating desirable behavioural equivalences in cases when more disciplined contexts are expected. In this paper we focus on two such common disciplines: sequentiality, meaning that at any time there is a single thread of computation, and well-bracketing, meaning that calls to external services obey a stack-like discipline. We formalize the disciplines by means of type systems. The main focus of the paper is on studying the consequence of the disciplines on behavioural equivalence. We define and study labelled bisimilarities for sequentiality and well-bracketing. These relations are coarser than ordinary bisimilarities. We prove that they are sound for the respective (contextual) barbed equivalence, and also complete under a certain technical condition. We show the usefulness of our techniques on a number of examples, that have mainly to do with the representation of functions and store.

In [14] we observe that the standard axiom $rec\, X.\,(\tau.X + E) = rec\, X.\,\tau.E$ of process calculi is no longer sound when actions have priority and, for example, actions of type δ have a lower priority than silent τ actions. Such a form of priority is common in timed process algebra, where, due to the interpretation of δ as a time delay, it naturally arises from the maximal progress assumption. We here present our solution, based on introducing an auxiliary operator $pri(E)$ defining a “priority scope”, to the long time open problem of axiomatizing priority using standard observational congruence: we provide a complete axiomatization for a basic process algebra with priority and (unguarded) recursion. We also show that, when the setting is extended by considering static operators of a discrete time calculus, an axiomatization that is complete over (a characterization of) finite-state terms can be developed by re-using techniques devised in the context of a cooperation with Prof. Jos Baeten.

7.4.2 Static Analysis of Properties of Concurrent Programs

Systems need to be updated to last for a long time in a dynamic environment. The updates can be performed both statically, by restarting the system, or dynamically. In any case they have to preserve the relevant properties of the system, while possibly enforcing new ones. In [17], we consider a simple yet general update mechanism, replacing a component of the system with a new one. We then define contexts and components as Constraint Automata interacting via either asynchronous or synchronous communication, and we express properties using Constraint Automata too. Then we build most general updates preserving specific properties, considering both a single property and all the properties satisfied by the original system, in a given context or in all possible contexts. To tackle also dynamic update, we consider the state transfer problem, namely how to find the state in which the new component should be started to ensure a correct overall behaviour.

7.4.3 Analysis of Blockchain Protocols and Smart Contracts

Recently Focus has developed an interest in the analysis of (stochastic) protocols used by (permissionless) blockchain systems under varying values of parameters, such as rates of block creations, communication rates, topologies. In particular, in [13] we analyze the protocol of the Bitcoin blockchain by using the PRISM probabilistic model checker. In particular, we (i) extend PRISM with the ledger data type, (ii) model the behaviour of the key participants in the protocol-the miners-and (iii) describe the whole protocol as a parallel composition of processes. The probabilistic analysis of the model highlights how forks happen and how they depend on specific parameters of the protocol, such as the difficulty of the cryptopuzzle and the network communication delays. Our results confirm that considering transactions in blocks at depth larger than 5 as permanent is reasonable because the majority of miners have consistent blockchains up-to that depth with probability of almost 1. We also study the behaviour of networks with churn miners, which may leave the network and rejoin afterwards, and with different topologies.

In [18] we define a technique for analyzing updates of smart contracts balances due to transfers of digital assets. The analysis addresses a lightweight smart contract language and consists of a two-step translation. First, we define the input-output behaviours of smart contract functions by means of a simple functional language with static dispatch. Then we associate the terms of this intermediate language with cost equations that compute the loss or gain of digital assets. The resulting equations can be fed to an off-the-shelf cost analyzer to provide upper bounds to the loss or gain. Our analysis has been prototyped and we report its assessments and discuss extensions with additional features.

7.5 Computer Science Education

Participants: Maurizio Gabbrielli, Michael Lodi, Simone Martini.

We have studied why and how to teach computer science principles (nowadays often referred to as “computational thinking”, CT), both in K-12 education and in the context of introductory courses (CS1) for non-majors. We have been interested in philosophical, sociological, and historical motivations to teach computer science. Furthermore, we have studied what concepts and skills related to computer science are not only technical abilities but have a general value for all students. Finally, we have tried to find/produce/evaluate suitable materials (tools, languages, lesson plans) to teach these concepts, taking into account: difficulties in learning computer science concepts (particularly programming); teacher training (both non-specialist and disciplinary teachers); innovative teaching methodologies (primarily based on constructivist approaches).

7.5.1 Computational Thinking

We used [24] a historical approach to review and discuss the original context in which the expression “computational thinking” originated (in the 1980s by Seymour Papert) and compared it with the one that brought the concept to the attention of the CS community (in 2006 by Jeannette Wing). If correctly understood, hence avoiding naive claims of skills transfer, both approaches are valuable. From Wing’s proposal, we should retain CS centrality, CT being the (scientific and cultural) substratum of the technical competencies, that provides a set of categories for understanding the algorithmic fabric of today’s world. From Papert, we should retain the constructionist idea that only students’ social and affective involvement in the technical content will make programming an interdisciplinary tool for learning (also) other disciplines.

7.5.2 CS1

We redesigned, because of the 2020 COVID-19 pandemic, a CS1 course for Math undergraduates, to be held online yet reflecting the face-to-face (F2F) experience as much as possible [48, 39], discussing what can be improved to strengthen the perception of a face-to-face (F2F) experience and mitigate the “presence paradox” we found: despite students being enthusiastic about the online format, most would still prefer a F2F course.

We proposed [27] an original “necessity driven” learning design, where novice students are put in an apparently well-known situation, but where they miss an essential ingredient to solve the problem. Struggling to solve it, they experience the necessity of that concept. At that moment, the instruction phase happens. Finally, students return to the problem with the necessary knowledge. We developed examples of necessity learning sequences in moments when the abstraction inside the programming language changes, e.g., when a new construct is introduced.

8 Bilateral contracts and grants with industry

8.1 Bilateral contracts with industry

SEAWALL (SEAmless loW latency cLoud pLatforms) is coordinated by a company in Bologna "Poggipolini". M. Gabbrielli is coordinating the University of Bologna unit. The industrial partners are Aetna, Bonfiglioli Riduttori, IMA, Sacmi, Philip Morris, Siemens, CDM, Digital River.

The project started in July 2020, and is expected to take 18 months.

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.

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

CRECOGI

Title: Concurrent, Resourceful and Effectful Computation by Geometry of Interaction

Duration: 2018 ->

Coordinator: Naohiko Hoshino (naophiko@kurims.kyoto-u.ac.jp)

Partners:

- Kyoto University, Japan

Inria contact: Ugo Dal Lago

Summary: The field of denotational semantics has successfully produced useful compositional reasoning principles for program correctness, such as program logics, fixed-point induction, logical relations, etc. The limit of denotational semantics was however that it applies only to high-level languages and to extensional properties. The situation has changed after the introduction of game semantics and the geometry of interaction (GoI), in which the meaning of programs is formalized in terms of movements of tokens, through which programs “talk to” or “play against” each other, thus having an operational flavour which renders them suitable as target language for compilers. The majority of the literature on GoI and games only considers sequential functional languages. Moreover, computational effects (e.g. state or I/O) are rarely taken into account, meaning that they are far from being applicable to an industrial scenario. This project’s objective is to develop a semantic framework for concurrent, resourceful, and effectful computation, with particular emphasis on probabilistic and quantum effects. This is justified by the greater and greater interest which is spreading around these two computation paradigms, motivated by applications to AI and by the efficiency quantum parallelism induces.

TCPro3

Title: Termination and Complexity Properties of Probabilistic Programs

Duration: 2019 ->

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 on quantum programs will be studied.

9.2 International research visitors

9.2.1 Visits of international scientists

Other international visits to the team

Guilhem Jaber

Status Researcher

Institution of origin: University of Nantes

Country: France

Dates: 1 week in December, 21

Context of the visit: Collaboration with D. Sangiorgi on "game semantics"

Mobility program/type of mobility: research stay

Georg Moser

Status (researchers)

Institution of origin: University of Innsbruck

Country: Austria

Dates: August 22 to September 5, 2021

Context of the visit: collaboration about verification of quantum security protocols

Mobility program/type of mobility: research stay.

Gilles Barthe, Giulio Malavolta, Itsaka Rakotonirina

Status (researchers)

Institution of origin: MPI Security and Privacy, Bochum

Country: Germany

Dates: September 6 to 10, 2021

Context of the visit: collaboration about complexity verification of probabilistic programs

Mobility program/type of mobility: research stay.

9.2.2 Visits to international teams

Research stays abroad

Martin Avanzini

Visited institution: University of Innsbruck

Country: Austria

Dates: July 11 to 23, 2021

Context of the visit: Research visit about complexity analysis of probabilistic programs

Mobility program/type of mobility: research stay

Francesco Gavazzo

Visited institution: ENS Lyon

Country: France

Dates: November 28 to December 4, 2021

Context of the visit: Research visit and invited talk at seminar CHOCOLA

Mobility program/type of mobility: research stay and seminar

Saverio Giallorenzo

Visited institution: University of Southern Denmark

Country: Denmark

Dates: October 11 to 15, 2021

Context of the visit: ongoing collaborations, project proposals

Mobility program/type of mobility: research stay, seminar

9.3 European initiatives

9.3.1 Horizon Europe

BEHAPI (Behavioural Application Program Interfaces) is an European Project H2020-MSCA-RISE-2017, running in the period March 2018 — February 2022. The topic of the project is behavioural types, as a suite of technologies that formalise the intended usage of API interfaces. Indeed, currently APIs are typically flat structures, i.e. sets of service/method signatures specifying the expected service parameters and the kind of results one should expect in return. However, correct API usage also requires the individual services to be invoked in a specific order. Despite its importance, the latter information is either often omitted, or stated informally via textual descriptions. The expected benefits of behavioural types include guarantees such as service compliance, deadlock freedom, dynamic adaptation in the presence of failure, load balancing etc. The project aims to bring the existing prototype tools based on these technologies to mainstream programming languages and development frameworks used in industry.

Participants: Mario Bravetti, Maurizio Gabbrielli, Ivan Lanese, Cosimo Laneve, Stefano Pio Zingaro, Davide Sangiorgi, Gianluigi Zavattaro.

9.4 National initiatives

9.4.1 DCore

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

The overall objective of the project is to develop a semantically well-founded, novel form of concurrent debugging, which we call "causal debugging". Causal debugging will comprise and integrate 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, Doriana Medic.

9.4.2 PROGRAMme

PROGRAMme (What is a program? Historical and philosophical perspectives) is an ANR project started on October 2017 and that will finish on October 2022; PI: Liesbeth De Mol (CNRS/Université de Lille3).

The aim of this project is to develop a coherent analysis and pluralistic understanding of "computer program" and its implications to theory and practice.

Participants: Simone Martini.

9.4.3 PPS

PPS (Probabilistic Programming Semantics) is an ANR PCR project that started on January 2020 and that will finish on July 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 shown crucial in various works in Bayesian inference and Machine Learning. Motivated by the rising impact of such probabilistic languages, the aim of this project is 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.

10 Dissemination

10.1 Promoting scientific activities

10.1.1 Scientific events: organisation

General chair, scientific chair

- International Federated Conference on Distributed Computing Techniques – DisCoTec (G. Zavattaro, chair of the steering committee)
- IFIP Int. Conference on Formal Techniques for Distributed Objects, Components and Systems (I. Lanese, chair of steering committee)

Member of the organizing committees

- Fourth Workshop on Probabilistic Interactive and Higher-Order Computation; ANR PPS and ERC CoG DIAPASoN project workshop, (U. Dal Lago, Organizer)
- Foundations of Software Science and Computation Structures (U. Dal Lago, Steering committee member)
- Workshop on Logic and Computational Complexity (U. Dal Lago, Steering committee member)
- Conference on Reversible Computation (I. Lanese, SC member)
- Interaction and Concurrency Experience (I. Lanese, SC member)
- International Federated Conference on Distributed Computing Techniques (I. Lanese, SC member)

10.1.2 Scientific events: selection

Member of the conference program committees

- 17th International Workshop on Termination (M. Avanzini)
- 12th International Workshop on Computing with Terms and Graphs (M. Avanzini)
- IEEE International Conference on Big Data 2021 (M. Bravetti)
- 21th IEEE International Conference on Software Quality, Reliability, and Security (M. Bravetti)
- 9th IPM International Conference on Fundamentals of Software Engineering (M. Bravetti)
- 48th International Colloquium on Automata, Languages, and Programming (U. Dal Lago)
- 24th International Conference on Foundations of Software Science and Computation Structures (U. Dal Lago)
- International Workshop on Big Services (BigService) 2021 (S. Giallorenzo)
- ACM International Conference on Information Technology for Social Good (GoodIT) 2021 (S. Giallorenzo)
- Workshop Agility with Microservices Programming 2021, workshop co-located with International Conference on Agile Software Development (XP) (S. Giallorenzo)
- 13th Conference on Reversible Computation (I. Lanese)
- 17th International Conference on Formal Aspects of Component Software (I. Lanese)
- 23rd International Conference on Coordination Models and Languages (I. Lanese)
- 14th Interaction and Concurrency Experience (I. Lanese)
- 3rd Workshop on Artificial Intelligence and fOrmal VERification, Logic, Automata, and sYnthesis (I. Lanese)
- 6th International Conference on the History and Philosophy of Computing (S. Martini)
- 48th International Colloquium on Automata, Languages, and Programming (D. Sangiorgi)
- 18th International Colloquium on Theoretical Aspects of Computing (D. Sangiorgi)
- 19th International Conference on Service-Oriented Computing (G. Zavattaro)
- 9th International Conference on Software Engineering and Formal Methods (G. Zavattaro)

10.1.3 Journal

Member of the editorial boards

- Journal of Universal Computer Science (M. Bravetti)
- Electronics Journal, section Computer Science and Engineering (M. Bravetti)
- Logical Methods in Computer Science (U. Dal Lago)
- Mathematical Structures in Computer Science (U. Dal Lago)
- Acta Informatica (U. Dal Lago)
- Distributed Computing, RAIRO – Theoretical Informatics and Applications, Foundations and Trends in Programming Languages, SN Computer Science, Springer. (D. Sangiorgi)

10.1.4 Invited talks

- 16th International Computer Science Symposium in Russia Online Worldwide Seminar on Logic and Semantics (U. Dal Lago)
- Seminar on “Challenges of Serverless: can languages help?” as visiting researcher at the Artificial Intelligence, Cybersecurity, and Programming Languages group, University of Southern Denmark, Denmark (S. Giallorenzo)
- Seminar on "Serverless vs Microservices Architectures" presented at the webinar "Microservies-based architectures for next-gen industrial applications" organised by CRIT S.r.l., Modena, Italy (S. Giallorenzo)
- 32nd International Conference on Concurrency Theory (D. Sangiorgi)

10.1.5 Leadership within the scientific community

- The “Microservices Community” is a European-based, international, non-profit organisation purposed to promote the development of microservices by bridging research, education, and innovation within and between businesses, universities, organisations and individuals. Members of Focus have played active roles in the Community since its inception in 2019. The organisation 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 (Swiss). Montesi is the president of the organisation, Guidi is a board member, 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 councils of the Italian Chapter of the EATCS, and of the Italian Association on Logic and its Applications.
- S. Martini is a member of the Council of the Commission on History and Philosophy of Computing, an organism of the International Union for History and Philosophy of Science, 2017-2023.

10.1.6 Research administration

- G. Zavattaro is coordinator of undergraduate studies (bachelor and master) at the Department of Computer Science and Engineering, University of Bologna (until October 2021).

10.2 Teaching - Supervision - Juries

10.2.1 Teaching

- Martin Avanzini
  - “Advanced Logic”, 34 hours, Université Côte d’Azur, France.
- Mario Bravetti
  - “Linguaggi, Compilatori e Modelli Computazionali”, 120 hours, 1st year Master, University of Bologna, Italy.
  - “Programming, Algorithms and Data Structures” module of “Programming and Computer Architectures”, 36 hours, 1st year Master, University of Bologna, Italy.
- Saverio Giallorenzo
  - - “Algorithms and Data Structures”, 20 hours, 2nd year Bachelor, University of Bologna, Italy
  - “Social Network Analysis”, 30 hours, 2nd year Master, University of Bologna, Italy
- Ugo Dal Lago
  - “Optimization”, 36 hours, 2nd year, University of Bologna, Italy.
  - “Foundations of Logic for Computer Science”, 24 hours, 2nd year Master, 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.
- Ivan Lanese
  - “Architettura degli Elaboratori”, 66 hours, 1st year, University of Bologna, Italy.
  - “Ingegneria del Software Orientata ai Servizi”, 22 hours, 2nd year Master, University of Bologna, Italy.
  - “Computational Methods for Bioinformatics”, 58 hours, 1st year Master, University of Bologna, Italy.
- Michael Lodi
  - “Computer Science Education”, 18 hours, 2nd year Master, University of Bologna, Italy.
- Simone Martini
  - “Programmazione”, 78 hours, 1st year, University of Bologna, Italy.
  - “Introduction to Algorithms and Programming”, 40 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.
  - “Architectures and Platforms for Artificial Intelligence”, 24 hours, Master in Artificial Intelligence, University of Bologna, Italy.

10.2.2 Supervision

Below are the details on the PhD students in Focus: starting date, topic or provisional title of the thesis, supervisor(s).

- Melissa Antonelli, November 2019. “Probabilistic Arithmetic and Almost-sure Termination”. Supervisor Ugo Dal Lago.
- Andrea Colledan, November 2021. “Complexity Analysis of Quantum Functional Programs”. Supervisor: Ugo Dal Lago.
- Gabriele Vanoni, November 2018. “Optimal Reduction, Geometry of Interaction, and the Space-Time Tradeoff”. Supervisor Ugo Dal Lago.
- Enguerrand Prebet, September 2019, “The pi-calculus model of programming languages”. Supervisors Daniel Hirschkoff and Davide Sangiorgi.

10.2.3 Juries

- Ugo Dal Lago has been member of the HdR evaluation of Lionel Vaux, Aix-Marseille Université, France.
- S. Martini has been member of the PhD evaluation committee of Davide Barbarossa, supervisors Giulio Manzonetto and Lorenzo Tortora de Falco, Université Paris 13, Sorbonne Paris Nord.

10.3 Popularization

10.3.1 Education

Michael Lodi and Simone Martini have carried out extended work of scientific popularization, including the following.

- They are members of the technical committee of Olimpiadi del Problem Solving (at Italian Ministry of Education); this involves preparation of material and supervision and jury during the finals.
- S. Martini gave the lecture “Programmer, c’est un peu comme jouer aux LEGO© : dimensions non-verbales de la programmation”, UNA Europa, Université Paris 1 Panthéon-Sorbonne, novembre 2021.

11 Scientific production

11.1 Major publications

[1] M. Bravetti and G. Zavattaro. ‘A Foundational Theory of Contracts for Multi-party Service Composition’. In: *Fundam. Inform.* 89.4 (2008), pp. 451–478.

[2] N. Busi, M. Gabbrielli and G. Zavattaro. ‘On the expressive power of recursion, replication and iteration in process calculi’. In: *Mathematical Structures in Computer Science* 19.6 (2009), pp. 1191–1222.

[3] P. Coppola and S. Martini. ‘Optimizing optimal reduction: A type inference algorithm for elementary affine logic’. In: *ACM Trans. Comput. Log.* 7.2 (2006), pp. 219–260.

[4] M. Gabbrielli and S. Martini. *Programming Languages: Principles and Paradigms*. Springer, 2010.

[5] D. Hirschkoff, É. Lozes and D. Sangiorgi. ‘On the Expressiveness of the Ambient Logic’. In: *Logical Methods in Computer Science* 2.2 (2006).

[6] U. D. Lago and M. Gaboardi. ‘Linear Dependent Types and Relative Completeness’. In: *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011*. IEEE Computer Society, 2011, pp. 133–142.

[7] I. Lanese, C. A. Mezzina and J.-B. Stefani. 'Reversibility in the higher-order \(\pi\)-calculus'. In: *Theor. Comput. Sci.* 625 (2016), pp. 25–84. DOI: 10.1016/j.tcs.2016.02.019. URL: https://doi.org/10.1016/j.tcs.2016.02.019.

[8] F. Montesi, C. Guidi and G. Zavattaro. 'Composing Services with JOLIE'. In: *Fifth IEEE European Conference on Web Services (ECOWS 2007)*. 2007, pp. 13–22.

[9] D. Sangiorgi. *An introduction to Bisimulation and Coinduction*. Cambridge University Press, 2012.

11.2 Publications of the year

International journals

[10] B. Accattoli, U. Dal Lago and G. Vanoni. 'The (In)Efficiency of interaction'. In: *Proceedings of the ACM on Programming Languages* 5.POPL (4th Jan. 2021), pp. 1–33. DOI: 10.1145/3434332. URL: https://hal.inria.fr/hal-03346750.

[11] M. Avanzini, G. Barthe and U. Dal Lago. 'On continuation-passing transformations and expected cost analysis'. In: *Proceedings of the ACM on Programming Languages* 5.ICFP (22nd Aug. 2021), pp. 1–30. DOI: 10.1145/3473592. URL: https://hal.inria.fr/hal-03338493.

[12] F. Barbanera, M. Dezani-Ciancaglini, I. Lanese and E. Tuosto. 'Composition and Decomposition of Multiparty Sessions'. In: *Journal of Logical and Algebraic Methods in Programming* (Feb. 2021). URL: https://hal.inria.fr/hal-03338671.

[13] S. Bistarelli, R. De Nicola, L. Galletta, C. Laneve, I. Mercanti and A. Veschetti. 'Stochastic Modelling and Analysis of the Bitcoin Protocol in Presence of Block Communication Delays'. In: *Concurrency and Computation: Practice and Experience* (8th Dec. 2021). URL: https://hal.archives-ouvertes.fr/hal-03347912.

[14] M. Bravetti. 'Axiomatizing Maximal Progress and Discrete Time'. In: *Logical Methods in Computer Science* (21st Jan. 2021). DOI: 10.23638/LMCS-17(1:1)2021. URL: https://hal.inria.fr/hal-03340630.

[15] M. Bravetti, M. Carbone, J. Lange, N. Yoshida and G. Zavattaro. 'A Sound Algorithm for Asynchronous Session Subtyping and its Implementation'. In: *Logical Methods in Computer Science* (4th Mar. 2021). DOI: 10.23638/LMCS-17(1:20)2021. URL: https://hal.inria.fr/hal-03340689.

[16] M. Bravetti and G. Zavattaro. 'Asynchronous session subtyping as communicating automata refinement'. In: *Software and Systems Modeling* 20 (4th Jan. 2021), pp. 311–333. DOI: 10.1007/s10270-020-00838-x. URL: https://hal.inria.fr/hal-03340699.

[17] D. Bresolin and I. Lanese. 'Static and Dynamic Property-Preserving Updates'. In: *Information and Computation* (Aug. 2021). URL: https://hal.inria.fr/hal-03338673.

[18] L. Cosimo and C. Sacerdoti Coen. 'Analysis of Smart Contracts Balances'. In: *Blockchain: Research and Applications*. Volume 2, Issue 3, September 2021, 2.3 (2021). URL: https://hal.archives-ouvertes.fr/hal-03347233.

[19] U. Dal Lago, C. Faggian and S. Ronchi Della Rocca. 'Intersection Types and (Positive) Almost-Sure Termination'. In: *Proceedings of the ACM on Programming Languages* (Jan. 2021). DOI: 10.1145/3434313. URL: https://hal.archives-ouvertes.fr/hal-03044416.

[20] U. Dal Lago and F. Gavazzo. 'Differential logical relations, part II increments and derivatives'. In: *Theoretical Computer Science* 895 (Dec. 2021), pp. 34–47. DOI: 10.1016/j.tcs.2021.09.027. URL: https://hal.inria.fr/hal-03520721.

[21] S. Giallorenzo, J. Mauro, M. G. Poulsen and F. Siroky. 'Virtualization Costs: Benchmarking Containers and Virtual Machines Against Bare-Metal'. In: *SN Computer Science* 2 (7th Aug. 2021). DOI: 10.1007/s42979-021-00781-8. URL: https://hal.inria.fr/hal-03337920.

[22] I. Lanese, D. Medić and C. A. Mezzina. 'Static versus Dynamic Reversibility in CCS'. In: *Acta Informatica* (Apr. 2021). URL: https://hal.inria.fr/hal-03338675.

[23] I. Lanese, A. Palacios and G. Vidal. ‘Causal-Consistent Replay Reversible Semantics for Message Passing Concurrent Programs’. In: *Fundamenta Informaticae* (15th Jan. 2021). DOI: 10.3233/FI-2021-2005. URL: https://hal.inria.fr/hal-03338674.

[24] M. Lodi and S. Martini. ‘Computational Thinking, Between Papert and Wing’. In: *Science and Education* 30.4 (Aug. 2021), pp. 883–908. DOI: 10.1007/s11191-021-00202-5. URL: https://hal.inria.fr/hal-03338884.

[25] J.-M. Madiot, D. Pous and D. Sangiorgi. ‘Modular coinduction up-to for higher-order languages via first-order transition systems’. In: *Logical Methods in Computer Science* Volume 17, Issue 3 (17th Sept. 2021). DOI: 10.46298/lmcs-17(3:25)2021. URL: https://hal.archives-ouvertes.fr/hal-03350199.

[26] S. Martini, A. Masini and M. Zorzi. ‘From 2-Sequents and Linear Nested Sequents to Natural Deduction for Normal Modal Logics’. In: *ACM Transactions on Computational Logic* 22.3 (24th June 2021), pp. 1–29. DOI: 10.1145/3461661. URL: https://hal.inria.fr/hal-03342394.

[27] M. Sbaraglia, M. Lodi and S. Martini. ‘A necessity-driven ride on the abstraction rollercoaster of CS1 programming’. In: *Informatics in Education* 20.4 (Dec. 2021), pp. 641–682. DOI: 10.15388/infedu.2021.28. URL: https://hal.inria.fr/hal-03466065.

International peer-reviewed conferences

[28] B. Accattoli, U. D. Lago and G. Vanoni. ‘The Space of Interaction’. In: LICS 2021 - 36th Annual ACM/IEEE Symposium on Logic in Computer Science. Vol. 5. Rome, France: IEEE, 4th Jan. 2021, pp. 1–13. DOI: 10.1109/LICS52264.2021.9470726. URL: https://hal.inria.fr/hal-03346767.

[29] C. Aubert and D. Medić. ‘Explicit Identifiers and Contexts in Reversible Concurrent Calculus’. In: RC 2021 - 13th International Conference on Reversible Computation. Nagoya / Virtual, Japan, 7th July 2021. URL: https://hal.inria.fr/hal-03300726.

[30] L. Bacchiani, M. Bravetti, S. Giallorenzo, J. Mauro, I. Talevi and G. Zavattaro. ‘Microservice Dynamic Architecture-Level Deployment Orchestration’. In: COORDINATION 2021 - 23rd IFIP WG 6.1 International Conference as Part of the 16th International Federated Conference on Distributed Computing Techniques. Vol. LNCS-12717. Coordination Models and Languages. Valletta / Virtual, Malta: Springer International Publishing, 14th June 2021, pp. 257–275. URL: https://hal.inria.fr/hal-03338602.

[31] M. Bravetti, J. Lange and G. Zavattaro. ‘Fair Refinement for Asynchronous Session Types’. In: FOSSACS’21 - 24th International Conference on Foundations of Software Science and Computation Structures. Luxembourgh, Luxembourg, 2021. DOI: 10.1007/978-3-030-71995-1. URL: https://hal.inria.fr/hal-03340696.

[32] U. Dal Lago and F. Gavazzo. ‘Resource Transition Systems and Full Abstraction for Linear Higher-Order Effectful Programs’. In: FSCD 2021 - 6th International Conference on Formal Structures for Computation and Deduction. Buenos Aires, Argentina, 17th July 2021. URL: https://hal.inria.fr/hal-03520742.

[33] U. Dal Lago, R. Kahle and I. Oitavem. ‘A Recursion-Theoretic Characterization of the Probabilistic Class PP’. In: *Proceedings of MFCS 2021*. MFCS 2021 - 46th International Symposium on Mathematical Foundations of Computer Science. Tallinn, Estonia, 23rd Aug. 2021. DOI: 10.4230/LIPIcs.MFCS.2021.35. URL: https://hal.inria.fr/hal-03346791.

[34] G. Fabbretti, I. Lanese and J.-B. Stefani. ‘Causal-Consistent Debugging of Distributed Erlang Programs’. In: *Reversible Computation*. RC 2021 - 13th Conference on Reversible Computation. Vol. 12805. Lecture Notes in Computer Science. Nagoya, Japan: Springer, 23rd June 2021, pp. 79–95. DOI: 10.1007/978-3-030-79837-6_5. URL: https://hal.inria.fr/hal-03338670.

[35] C. Faggian and F. Gavazzo. ‘A Relational Theory of Monadic Rewriting Systems, Part I’. In: 2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS). Rome, France: IEEE, 29th June 2021, pp. 1–14. DOI: 10.1109/LICS52264.2021.9470633. URL: https://hal.archives-ouvertes.fr/hal-03455778.

[36] S. Giallorenzo, F. Montesi, M. Peressotti, F. Rademacher and S. Sachweh. 'Jolie and LEMMA: Model-Driven Engineering and Programming Languages Meet on Microservices'. In: 23th International Conference on Coordination Languages and Models (COORDINATION). Vol. LNCS-12717. Coordination Models and Languages. Valletta, Malta: Springer International Publishing, 2021, pp. 276–284. DOI: 10.1007/978-3-030-78142-2_17. URL: https://hal.inria.fr/hal-03347326.

[37] S. Giallorenzo, F. Montesi, M. Peressotti, D. Richter, G. Salvaneschi and P. Weisenburger. 'Multiparty Languages: The Choreographic and Multitier Cases'. In: ECOOP 2021 - European Conference on Object-Oriented Programming. Aarhus, Denmark, 12th July 2021. DOI: 10.4230/LIPIcs.ECOOP.2021.23. URL: https://hal.inria.fr/hal-03337915.

[38] I. Lanese and I. Phillips. 'Forward-Reverse Observational Equivalences in CCSK'. In: RC 2021 - 13th Conference on Reversible Computation. Nagoya, Japan, 2021, pp. 126–143. DOI: 10.1007/978-3-030-79837-6_8. URL: https://hal.inria.fr/hal-03338669.

[39] M. Lodi, M. Sbaraglia, S. P. Zingaro and S. Martini. 'The Online Course Was Great: I Would Attend It Face-to-Face: The Good, The Bad, and the Ugly of IT in Emergency Remote Teaching of CS1'. In: GoodIT '21: Conference on Information Technology for Social Good. Roma, Italy: ACM, 9th Sept. 2021, pp. 242–247. DOI: 10.1145/3462203.3475902. URL: https://hal.inria.fr/hal-03338808.

[40] S. Orlando, V. D. Pasquale, F. Barbanera, I. Lanese and E. Tuosto. 'Corinne, a Tool for Choreography Automata'. In: Formal Aspects of Component Software. Vol. 13077. Lecture Notes in Computer Science. Grenoble (online due to covid), France: Springer International Publishing, 28th Oct. 2021, pp. 82–92. DOI: 10.1007/978-3-030-90636-8_5. URL: https://hal.inria.fr/hal-03468190.

Conferences without proceedings

[41] D. Hirschkoff, E. Prebet and D. Sangiorgi. 'On sequentiality and well-bracketing in the π-calculus'. In: 2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS). Rome, Italy: IEEE, 29th June 2021, pp. 1–13. DOI: 10.1109/LICS52264.2021.9470559. URL: https://hal.archives-ouvertes.fr/hal-03203191.

[42] P. Pistone. 'On Generalized Metric Spaces for the Simply Typed Lambda-Calculus'. In: 2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS). Rome, Italy: IEEE, 29th June 2021, pp. 1–14. DOI: 10.1109/LICS52264.2021.9470696. URL: https://hal.archives-ouvertes.fr/hal-03346950.

Scientific book chapters

[43] M. Antonelli, U. Dal Lago and P. Pistone. 'On Measure Quantifiers in First-Order Arithmetic'. In: *Proceedings of CIE 2021*. 2nd July 2021, pp. 12–24. DOI: 10.1007/978-3-030-80049-9_2. URL: https://hal.inria.fr/hal-03346804.

Reports & preprints

[44] C. Aubert and D. Medić. *Enabling Replications and Contexts in Reversible Concurrent Calculus*. 26th Mar. 2021. URL: https://hal.archives-ouvertes.fr/hal-03183053.

[45] M. Avanzini, G. Moser, R. Péchoux, S. Perdrix and V. Zamdzhiev. *Quantum Expectation Transformers for Cost Analysis*. 23rd Jan. 2022. URL: https://hal.inria.fr/hal-03540366.

[46] G. Geoffroy. *Extensional Denotational Semantics of Higher-Order Probabilistic Programs, Beyond the Discrete Case*. 12th Apr. 2021. URL: https://hal.archives-ouvertes.fr/hal-03187624.

[47] D. Hirschkoff, E. Prebet and D. Sangiorgi. *On sequentiality and well-bracketing in the π-calculus (Long Version)*. 13th Dec. 2021. URL: https://hal.archives-ouvertes.fr/hal-03478052.

Other scientific publications

[48] M. Sbaraglia, M. Lodi, S. P. Zingaro and S. Martini. 'The Good, The Bad, and The Ugly of a Synchronous Online CS1'. In: ITiCSE 2021: 26th ACM Conference on Innovation and Technology in Computer Science Education. Vol. 2. 26th ACM Conference on Innovation and Technology in Computer Science Education V. 2 (ITiCSE 2021). Virtual Event, Germany: ACM, 26th June 2021, pp. 660–660. DOI: 10.1145/3456565.3460075. URL: https://hal.inria.fr/hal-03338864.

11.3 Other

Scientific popularization

[49] L. Bacchiani, M. Bravetti, J. Lange and G. Zavattaro. 'A Session Subtyping Tool'. In: COORDINATION 2021 - 23rd IFIP WG 6.1 International Conference Coordination Models and Languages, Held as Part of the 16th International Federated Conference on Distributed Computing Techniques. Vol. LNCS-12717. Coordination Models and Languages. Valletta / Virtual, Malta: Springer International Publishing, 14th June 2021, pp. 90–105. DOI: 10.1007/978-3-030-78142-2_6. URL: https://hal.inria.fr/hal-03340750.

[50] I. Lanese, U. P. Schultz and I. Ulidowski. 'Reversible Execution for Robustness in Embodied AI and Industrial Robots'. In: *IT Professional* 23 (1st May 2021), pp. 12–17. DOI: 10.1109/mitp.2021.3073757. URL: https://hal.inria.fr/hal-03338672.

11.4 Cited publications

[51] M. Carbone, K. Honda and N. Yoshida. 'A Calculus of Global Interaction based on Session Types'. In: *Electr. Notes Theor. Comput. Sci.* 171.3 (2007), pp. 127–151.

[52] A. Igarashi and N. Kobayashi. 'Resource usage analysis'. In: *POPL conference*. ACM Press, 2002, pp. 331–342.

[53] N. Kobayashi and D. Sangiorgi. 'A hybrid type system for lock-freedom of mobile processes'. In: *ACM Trans. Program. Lang. Syst.* 32.5 (2010).