RESEARCH CENTRE

Sophia Antipolis - Méditerranée

IN PARTNERSHIP WITH:

Université de Bologne (Italie)

2020

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 Scientist

- Martin Avanzini [Inria, Researcher]

Faculty Members

- Davide Sangiorgi [Team leader, University of Bologna, Professor, HDR]
- Mario Bravetti [University of Bologna, Associate Professor]
- Ugo Dal Lago [University of Bologna, Professor]
- Maurizio Gabbrielli [University of Bologna, Professor]
- Saverio Giallorenzo [University of Bologna, Assistant Professor]
- Ivan Lanese [University of Bologna, Associate Professor]
- Cosimo Laneve [University of Bologna, Professor]
- Simone Martini [University of Bologna, Professor, HDR]
- Gianluigi Zavattaro [University of Bologna, Professor]

Post-Doctoral Fellows

- Aurore Alcolei [University of Bologna]
- Francesco Gavazzo [University of Bologna]
- Guillaume Geoffroy [University of Bologna]
- Michael Lodi [University of Bologna, from Apr 2020]
- Doriana Medic [Inria]
- Paolo Pistone [University of Bologna]
- Stefano Pio Zingaro [University of Bologna]

PhD Students

- Melissa Antonelli [University of Bologna]
- Michael Lodi [University of Bologna, until Mar 2020]
- Enguerrand Prebet [ENS Lyon]
- Gabriele Vanoni [University of Bologna]
- Adele Veschetti [University of 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, so in CBUS theories are needed that allow us to understand the systems, design or program them, 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, well address 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 at hand here: they can express causality and order information on messages [37, 36, 38], 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

5.1 Awards

Davide Sangiorgi received with his work "A Theory of Bisimulation for the pi-Calculus" the *test of time award from CONCUR* (period 1992–1995).[1] The purpose of the award is to recognize important achievements in concurrency theory that were published at the CONCUR conference and have stood the test of time.

6 New software and platforms

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 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.

[1] See `https://concur2020.forsyte.at/test-of-time.html`.

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 a more effective development and debugging experience.

In 2019 Jolie also saw the development of a new Jolie library, called TQuery, which 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 sound variant of the Aggregation Framework, the query language of the most popular document-oriented database: MongoDB. Usage scenarios for Tquery are (but not limited to) eHealth, the Internet-of-Things, and

Edge Computing, where data should be handled in an ephemeral way, i.e., in a real-time manner but with the constraint that data shall not persist in the system.

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

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

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/

Contacts: Tong Liu, Maurizio Gabbrielli

6.1.3 AIOCJ

Name: Adaptive Interaction-Oriented Choreographies in Jolie

Keyword: Dynamic adaptation

Scientific Description: AIOCJ is an open-source choreographic programming language for developing adaptive systems. It allows one to describe a full distributed system as a unique choreographic program and to generate code for each role avoiding by construction errors such as deadlocks. Furthermore, it supports dynamic adaptation of the distributed system via adaptation rules.

Functional Description: AIOCJ is a framework for programming adaptive distributed systems based on message passing. AIOCJ comes as a plugin for Eclipse, AIOCJ-ecl, allowing one to edit descriptions of distributed systems written as adaptive interaction-oriented choreographies (AIOC). From interaction-oriented choreographies the description of single participants can be automatically derived. Adaptation is specified by rules allowing one to replace predetermined parts of the AIOC with a new behaviour. A suitable protocol ensures that all the participants are updated in a coordinated way. As a result, the distributed system follows the specification given by the AIOC under all changing sets of adaptation rules and environment conditions. In particular, the system is always deadlock free. AIOCJ can interact with external services, seen as functions, by specifying their URL and the protocol they support (HTTP, SOAP, ...). Deadlock-freedom guarantees of the application are preserved provided that those services do not block.

URL: http://www.cs.unibo.it/projects/jolie/aiocj.html

Contacts: Saverio Giallorenzo, Ivan Lanese

Participants: Ivan Lanese, Jacopo Mauro, Maurizio Gabbrielli, Mila Dalla Preda, Saverio Giallorenzo

6.1.4 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 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 2020 a main revision of CauDEr has been started, and will be concluded in 2021. 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. Additional features of the language, in particular related to distribution and error handling have been also considered. The new version (still unstable) is available at: https://github.com/mistupv/cauder-v2

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

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

Contact: Ivan Lanese

Participant: Ivan Lanese

Partner: Universitat Politècnica de València

6.1.5 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`

Contacts: Tong Liu, Roberto Amadini, Jacopo Mauro, Maurizio Gabbrielli

6.1.6 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: The inference procedure has been further optimized, documentation has been added and the experimental results were significantly extended.

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

Contact: Martin Avanzini

6.1.7 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 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

Contacts: Adele Veschetti, Cosimo Laneve

6.1.8 Tquery

Keywords: Ephemeral Data, Microservices, Big data, Querying

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

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

Contacts: Saverio Giallorenzo, Fabrizio Montesi, Larisa Safina, Stefano Pio Zingaro

Partner: University of Southern Denmark

6.1.9 APP

Name: Allocation Priority Policies

Keywords: Serverless, Scheduling, Cloud computing, Optimisation

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

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

Contacts: Saverio Giallorenzo, Gianluigi Zavattaro, Jacopo Mauro, Giuseppe De Palma

6.1.10 Choral

Keywords: Choreographic Programming, Compilation, Modularity, Distributed programming

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.

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 following 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.

Release Contributions: First release

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

Contacts: Saverio Giallorenzo, Fabrizio Montesi

Participants: Saverio Giallorenzo, Fabrizio Montesi, Marco Peressotti

Partner: University of Southern Denmark

7 New results

7.1 Service-oriented and Cloud Computing

Participants Mario Bravetti, Maurizio Gabbrielli, Saverio Giallorenzo, Fabrizio Montesi, Davide Sangiorgi, Gianluigi Zavattaro.

7.1.1 Service-Oriented Computing

Automata models are well-established in many areas of computer science and are supported by a wealth of theoretical results including a wide range of algorithms and techniques to specify and analyse systems. In [14], we have introduced choreography automata for the choreographic modelling of communicating systems. The projection of a choreography automaton yields a system of communicating finite-state machines. We have considered both the standard asynchronous semantics of communicating systems and a synchronous variant of it. For both, the projections of well-formed automata are proved to be live as well as lock- and deadlock-free.

7.1.2 Cloud Computing

Serverless computing is a paradigm for programming cloud applications in terms of stateless functions, executed and scaled in proportion to inbound requests. In [19] we have revisited SKC, a calculus capturing the essential features of serverless programming. By exploring the design space of the language, we refined the integration between the fundamental features of the two calculi that inspire SKC: the λ- and the π-calculus. That investigation led us to a revised syntax and semantics, which support an increase in the expressiveness of the language. In particular, now function names are first-class citizens and can be passed around. These new features have been applied to two non-trivial use cases from artificial intelligence, which model, respectively, a perceptron and an image tagging system into compositions of serverless functions. We have also illustrated how SKC supports reasoning on serverless implementations, i.e., the underlying network of communicating, concurrent, and mobile processes which execute serverless functions in the cloud. To that aim, we have presented an encoding from SKC to the asynchronous π-calculus and proved it correct in terms of an operational correspondence.

Finally, following previous work on the automated deployment of component based applications, we have presented in [27] a formal model specifically tailored for reasoning on the deployment of microservice architectures. The first result that we have presented is a formal proof of decidability of the problem of synthesizing optimal deployment plans for microservice architectures, a problem which was previously proved to be undecidable for generic component-based applications. Then, given that such proof translates the deployment problem into a constraint satisfaction problem, we have presented the implementation of a tool that, by exploiting state-of-the-art constraint solvers, can be used to actually synthesize optimal deployment plans. We have also evaluated the applicability of our tool on a realistic microservice architecture taken from the literature, namely, the deployment of an email processing pipeline that needs to scale-in or -out depending on the amount of incoming emails.

7.2 Models for Reliability

Participants Ivan Lanese, Doriana Medic.

We have continued the study of reversibility started in the past years. A main line of research this year has been trying to distil the essence of causal-consistent reversibility (where any action of a concurrent system can be undone provided that its consequences have been undone beforehand) so to extract general results and techniques from the plethora of ad hoc approaches in the literature. Following this idea, in [21] we considered systems based on causal-consistent reversibility as Labelled Transition Systems with Independence and investigated which axioms need to hold in order to ensure that relevant properties from the literature (such as the Parabolic Lemma or the Causal-Consistency Theorem) hold. It turned out that few axioms are enough to guarantee most of them. Also, we defined two new properties, Causal Liveness and Causal Safety, which state, respectively, that a past action can be undone if and only if all its consequences have been undone. These properties directly formalise the common informal presentation of causal-consistent reversibility. In [20] we defined a technique to take a forward system defined as a Labelled Transition System satisfying suitable constraints and automatically build its causal-consistent reversible extension. This nicely complements the work above, and indeed we showed that the built model satisfies the axioms proposed in [21] and hence enjoys a number of relevant properties. The general studies above were expressive enough to cover many ad hoc models in the literature, such as reversible higher-order π and reversible Core Erlang. We were also able to extend the fragment of Core Erlang covered by previous approaches, tackling also error handling constructs based on links. In [9] we presented a parametric framework for reversible π-calculi and exploited it to study three different notions of causal relation defined for π-calculus. Since the main difference between the approaches is in the treatment of parallel extrusions of the same name, the framework is parametric in the bookkeeping data structure used to keep track of the object dependency. We showed that the three corresponding instances of our model enjoy the standard properties for a reversible calculus mentioned above (Parabolic Lemma and Causal-Consistency).

On the application side, in [23] we defined a formal framework to model Software Transactional Memories (STM), a concurrency control mechanism for shared memory systems where concurrent accesses are allowed, but undone in case they create interferences. In particular, when a transaction aborts, all the updates it made are reversed and the system is brought back to the state before the transaction is executed. We have shown that with minor variations it is possible to model two common policies for STM: reader preference and writer preference.

Finally, we have participated to a large dissemination effort to present the results of the European COST Action IC1405 on “Reversible Computation - Extending Horizons of Computing”, which took place in the years 2015-2019. The COST Action covered most areas of reversible computation, and we have contributed to the areas of foundations [26] and software and systems [30] as well as to the case study on debugging [29]. Since the COST Action involved 33 (mostly European) countries, the dissemination effort included most of the results obtained by the European reversible computation community in the last 5 years.

7.3 Probabilistic and Quantum Systems

Participants Martin Avanzini, Ugo Dal Lago, Guillaume Geoffroy, 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 2020.

Recently, Focus has also begun to take an interest in the foundational aspects of quantum computing and in particular in quantum programming languages and quantum computational models. In presence of measurements, in fact, quantum programs have an essentially probabilistic evolution and therefore any techniques for termination and complexity analysis can potentially also be applied to quantum programs. The resource of interest here includes the number of qubits, a parameter of paramount importance given the inherent scarcity of this resource in concrete quantum architectures.

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 Probabilistic Program Analysis and Higher-Order Calculi

In the last two decades, there has been much progress on model checking of both probabilistic systems and higher-order programs. Dal Lago, together with Kobayashi and Grellois, have initiated a study [7] on the *probabilistic higher-order model checking problem*, by giving some first theoretical and experimental results. Interestingly, reachability analysis has been proven to be undecidable already at order two.

Dal Lago, in a joint work with Heijtjes and Guerrieri [18], has introduced a probabilistic lambda-calculus in which the probabilistic choice operator is decomposed into two syntactic constructs, thus recovering a form of confluence which is impossible to achieve without this decomposition.

Avanzini et al. [12] has introduced a novel methodology for the automated resource analysis of probabilistic imperative programs, which gives rise to a *modular approach*. Program fragments are analysed in full independence, thereby allowing the methodology to scale to larger programs. This methodology has also been implemented in the tool eco-IMP, see Section 6. Ample experimental evidence shows that this tool runs typically at least one *order of magnitude faster* than existing tools, while retaining their precision.

In joint work with Crubillé and Barak [13], Dal Lago has introduced a new model of complexity-bounded probabilistic higher-order computation in which every algorithm is by construction polynomial time computable, and which is thus perfectly adequate as a language for cryptographic construction. Interestingly, certain cryptographic primitives are shown to be impossible to generalize to an higher-order setting.

7.3.2 Quantum Computational Models and Programming Languages

Martini, together with Guerrini and Masini [6], has proposed a definition of quantum Turing machines more general than the usual ones, at the same time extending and unifying the existing definitions due to Deutsch and Bernstein & Vazirani. In particular, an arbitrary quantum input is allowed, together with meaningful superpositions of computations, some of them being “terminated” with an “output”, some others not.

7.3.3 Differences in Probabilistic and Higher-Order Computatbility

Geoffroy and Pistone [24] have shown that the standard metric on real numbers can be lifted to higher-order types in a novel way, yielding a metric semantics of the simply typed lambda-calculus in which types are interpreted as quantale-valued partial metric spaces. Using such metrics, a class of higher-order denotational models, called diameter space models, has been shown to provide a quantitative semantics of approximate program transformations.

7.4 Verification Techniques

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

7.4.1 Coinductive Techniques

It is known that proving behavioural equivalences in higher-order languages can be hard, because interactions involve complex values, namely terms of the language. 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 [4] we have adapted the technique of unique solution of equations, that implicitly captures 'up-to context' reasoning, to the setting of the Higher-Order π-calculus.

In [10] we have studied the formalisation (in the HOL theorem prover HOL4) of the theory of unique solution of equations and contraction, for CCS as well as for the main results in Milner's book on the the Calculus of Communicating Systems (CCS). The formalisation consists of about 24,000 lines (1MB) of code. Some refinements of the theory of 'unique solution of contractions' itself have thereby been derived.

Below we summarise other papers that are about applications of the theory of coinduction to various settings.

In [11] we have revisited the Interaction Abstract Machine (IAM), a machine based on Girard's Geometry of Interaction. It is an unusual machine, not relying on environments, presented on linear logic proof nets, and whose soundness proof is convoluted and passes through various other formalisms. Here we have provided a new direct proof of its correctness, based on a variant of Sands's improvements, a form of bisimulation related to the above-mentioned contractions.

In [3] we have introduced a notion of applicative similarity in which not terms but monadic values arising from the evaluation of effectful terms, can be compared. We have proven this notion to be fully abstract whenever terms are evaluated in a call-by-name order.

The π-calculus has been advocated as a model to interpret, and give semantics to, languages with higher-order features. Often these languages make use of forms of references (viewing a store as set of references). In [25] we have developed coinductive proof techniques to reason about the representation of references in the π-calculus, investigating both their soundess and their completeness with respect to contextual behavioural equivalences such as barbed congruence.

7.4.2 Deadlock Analysis

In [15] we have discussed how to ensure relevant communication properties of communicating systems such as deadlock freedom in a compositional way. The basic idea is that communicating systems can be composed by taking two of their participants and transforming them into coupled forwarders connecting the two systems. We have investigated how to adaptat the idea to settings with asynchronous and synchronous communications.

7.4.3 Static Analysis of Properties of Concurrent Programs

In [2] we have reviewed techniques, mainly based on the theory of process calculi, that we used over the past twenty years to prove results about the expressiveness of coordination languages and behavioural contracts for Service-Oriented Computing. Then, we have shown how such techniques recently contributed to the clarification of aspects of session types such as asynchronous session subtyping.

In [17] we have presented a type-based analysis ensuring memory safety and object protocol completion in the Java-like language Mungo. Objects are annotated with usages, typestates-like specifications of the admissible sequences of method calls. The type system has been implemented in the form of a type checker and a usage inference tool.

7.4.4 Concurrent Constraint Languages

Concurrent Constraint Programming (CCP) is a declarative model for concurrency where agents interact by telling and asking constraints (pieces of information) in a shared store. Some previous works have developed (approximated) declarative debuggers for CCP languages. However, the task of debugging concurrent programs remains difficult. In [5] we have defined a dynamic slicer for CCP (and other language variants) and we have shown it to be a useful companion tool for the existing debugging techniques. Our slicer allows for marking part of the state of the computation and assists the user to eliminate most of the redundant information in order to highlight the errors. We have shown that this technique can be tailored to several variants of CCP, such as the timed language ntcc, linear CCP (an extension of CCP based on linear logic where constraints can be consumed) and some extensions of CCP dealing with epistemic and spatial information. We have also developed a prototypical implementation freely available for making experiments.

7.4.5 Logical Relations

Logical relations are one of the most powerful techniques in the theory of programming languages, and have been used extensively for proving properties of a variety of higher-order calculi. However, there are properties that cannot be immediately proved by means of logical relations, for instance program continuity and differentiability in higher-order languages extended with real-valued functions. Informally, the problem stems from the fact that these properties are naturally expressed in terms of non-ground type (or, equivalently, on open terms of base type), and there is no apparent good definition for a base case (i.e. for closed terms of ground types). To overcome this issue, in [16] we have studied a generalization of the concept of a logical relation, called open logical relation. Our setting is a simply-typed λ-calculus enriched with real numbers and real-valued first-order functions from a given set, such as the one of continuous or differentiable functions. We have shown by way of open logical relations the correctness of the core of a recently published algorithm for forward automatic differentiation. Finally, we have defined a refinement-based type system for local continuity in an extension of our calculus with conditionals, and proven its soundness using open logical relations.

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), in the context of K-12 education. 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); stereotypes about computer science (teachers' and students' mindset); teacher training (both non-specialist and disciplinary teachers); innovative teaching methodologies (primarily based on constructivist and constructionist learning theories).

Apart from these investigations, this year we have also been involved in the development of methodologies for assessing dropout rates of students, to benefit students as well as education institutions.

7.5.1 Computational Thinking

We have reviewed several definitions of computational thinking [8], finding they share a lot of common elements, of very different nature. We have classified them in mental processes, methods, practices, and transversal skills. Many of these elements seem to be shared with other disciplines and resonate with the current narrative on the importance of 21st-century skills. Our classification helps on shedding light on the misconceptions related to each of the four categories, showing that, not to dilute the concept, elements of computational thinking should be intended inside the discipline of Informatics, being its "disciplinary way of thinking".

7.5.2 Assessing Dropout Rates

Among the many open problems in the learning process, students dropout is one of the most complicated and negative ones, both for the student and the institutions, and being able to predict it could help to alleviate its social and economic costs. To address this problem we have developed in [28] a tool that, by exploiting machine learning techniques, allows one to predict the dropout of a first-year undergraduate student. The proposed tool allows one to estimate the risk of quitting an academic course, and it can be used either during the application phase or during the first year, since it selectively accounts for personal data, academic records from secondary school and also first year course credits. Our experiments have been performed by considering real data of students from eleven schools of a major University.

8 Bilateral contracts and grants with industry

8.1 Bilateral contracts with industry

SEAWALL

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.

9 Partnerships and cooperations

In the following we list only those projects held by members of Focus which are managed by INRIA, notably, this excludes the ERC CoG DIAPAsON "Differential Program Semantics" [2].

9.1 International initiatives

9.1.1 Inria associate team not involved in an IIL

CRECOGI

Title: *Concurrent, Resourceful and Effectful COmputation, by Geometry of Interaction*

Duration: 2018 - *2020*

Coordinator: Ugo Dal Lago

Partners: Research Institute for Mathematical Sciences, Kyoto (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.

[2] See `https://site.unibo.it/diapason`.

TC(PRO)[3] Martin Avanzini is member of the INRIA associate team TC(PRO)[3][3] headed by Romain Péchoux (EPI Mocqua). The associate team is concerned, in collaboration with the University of Innsbruck, in the development of formal methodologies for reasoning about termination and complexity of probabilistic and quantum systems.

9.2 International research visitors

Due to the ongoing COVID-19 pandemic, no international research visits have been conducted this year.

9.3 European initiatives

9.3.1 FP7 & H2020 Projects

BEHAPI 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.

9.3.2 Collaborations with major European organizations

- Universitat Politecnica de Valencia, Spain (on reversibility for Erlang). Contact person(s) in Focus: Lanese. Active exchange but no research visits due to COVID-19 crisis.
- Max Planck Institute for Security and Privacy, Bochum (on static analysis of probabilistic programs). Contact person(s) in Focus: Avanzini and Dal Lago. Mostly confined to online collaboration due to COVID-19 crisis.

9.4 National initiatives

9.4.1 DCore

DCore (Causal debugging for concurrent systems) is a 4-years ANR project that started on March 2019 and that will end in August 2023.

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.

[3]See https://members.loria.fr/RPechoux/ea-tcpro%C2%B3/.

9.4.2 REPAS

REPAS (Reliable and Privacy-Aware Software Systems via Bisimulation Metrics) is an ANR Project that started on October 2016 and that finished on October 2020.

The project aims at investigating quantitative notions and tools for proving program correctness and protecting privacy. In particular, the focus was put on bisimulation metrics, which are the natural extension of bisimulation to quantitative systems. As a key application, we developed mechanisms to protect the privacy of users when their location traces are collected.

Participants Ugo Dal Lago, Francesco Gavazzo, Davide Sangiorgi.

9.4.3 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.

9.4.4 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

- IFIP Int. Conference on Formal Techniques for Distributed Objects, Components and Systems (I. Lanese, chair of steering committee)

Member of the organizing committees

- Third Workshop on Probabilistic Interactive and Higher-Order Computation; ANR PPS and ERC CoG DIAPASoN project workshop (U. Dal Lago, Organizer)
- 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)

Chair of conference program committees

- 12th Conference on Reversible Computation (I. Lanese)
- 21st International Workshop on Logic and Computational Complexity (M. Avanzini)
- Joint 6th International Workshop on Linearity and 4th International Workshop on Trends in Linear Logic and its Applications (U. Dal Lago)
- International Conference on Microservice, part of the Bologna federated Conference on Programming Languages (G. Zavattaro)

Member of the conference program committees

- 35th Annual ACM/IEEE Symposium on Logic in Computer Science (D. Sangiorgi, U. Dal Lago)
- IEEE International Conference on Big Data 2020 (M. Bravetti)
- 24th International Conference on Foundations of Software Science and Computation Structures (U. Dal Lago)
- 20th IEEE International Conference on Software Quality, Reliability, and Security (M. Bravetti)
- 15th Workshop on Coalgebraic Methods in Computer Science (D. Sangiorgi)
- 47th ACM SIGPLAN Symposium on Principles of Programming Languages (U. Dal Lago)

10.1.2 Journal

Member of the editorial boards

- Journal of Universal Computer Science (M. Bravetti)
- Electronics Journal, section Computer Science & Engineering (M. Bravetti)
- Logical Methods in Computer Science (U. Dal Lago)
- Mathematical Structures in Computer Science (U. Dal Lago)
- Acta Informatica (U. Dal Lago)
- Book "Reversible Computation: Extending Horizons of Computing - Selected Results of the COST Action IC1405". Lecture Notes in Computer Science 12070, Springer 2020, ISBN 978-3-030-47360-0 (I. Lanese)
- Distributed Computing, RAIRO – Theoretical Informatics and Applications, Foundations and Trends in Programming Languages, SN Computer Science, Springer. (D. Sangiorgi)

10.1.3 Invited talks

- BUsec, the security seminar at Boston University, on "Higher-Order Cryptography", May 2020 (U. Dal Lago)

10.1.4 Leadership within the scientific community

- 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, 2017-2021.

10.1.5 Research administration

- M. Gabbrielli is Deputy Head of the Department of Computer Science and Engineering, University of Bologna, since May 2018.
- D. Sangiorgi is coordinator of postgraduate studies at the Department of Computer Science and Engineering, University of Bologna.
- G. Zavattaro is coordinator of undergraduate studies at the Department of Computer Science and Engineering, University of Bologna.

10.2 Teaching - Supervision - Juries

10.2.1 Teaching

- Martin Avanzini
  - "Introduction to Theoretical Computer Science", 30 hours, 1st year, University of Innsbruck, Austria.
- Mario Bravetti
  - "Linguaggi, Compilatori e Modelli Computazionali", 120 hours, 1st year Master, University of Bologna, Italy.
  - "Models and Languages for Service-Oriented and Cloud Computing", 16 hours, PhD program, University of Bologna, Italy.
- Ugo Dal Lago
  - "Algorithms and Data Structures for Biology", 60 hours, 2nd year, University of Bologna, Italy. 20 hours, 1st year, University of Bologna, Italy.
  - "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.
  - "Algorithms and Data Structures for Computational Biology", 28 hours, 1st year Master, University of Bologna, Italy.
  - "Computational Methods for Bioinformatics", 36 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, University of Bologna, Italy.
  - “Algoritmi e Strutture di Dati”, 32 hours, undergraduate program in Comperuter Science and Management, University of Bologna, Italy.
  - “Scalable and Cloud Programming”, 50 hours, Master, University of Bologna, Italy.
  - “Languages and Algorithms for Artificial Intelligence”, 32 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.
- Gabriele Vanoni, November 2018. “Optimal Reduction, Geometry of Interaction, and the Space-Time Tradeoff”. Supervisor Ugo Dal Lago.
- Enguerrand Prebet, Sept 2019, “The pi-calculus model of programming languages”. Supervisors Daniel Hirschkoff and Davide Sangiorgi.

PhD theses completed in 2020:

- Michael Lodi, April 2020, “Introducing Computational Thinking in K-12 Education: Historical, Epistemological, Cognitive and Affective Aspects” [31]. Supervisor Simone Martini.
- Adrien Durier, June 2020, “Proving behavioural properties of higher-order concurrent languages”, ENS de Lyon and University of Bologna. Supervisors: Daniel Hirschkoff and Davide Sangiorgi.

10.2.3 Juries

- G. Zavattaro has been member of the PhD evaluation committee at the Department of Computer Science, University of Pisa, Italy.
- I. Lanese has been member of the PhD evaluation committee of Kyriaki Psara, supervisor Anna Philippou, University of Cyprus, Cyprus.

10.3 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), `http://www.olimpiadiproblemsolving.com`; this involves preparation of material and supervision and jury during the finals.
- S. Martini gave the talk “Peut-on définir ce qu’est un algorithme ?”, during the interdisciplinary seminars on “Artificial Intelligence Stakes in the Society”, Una Europa, Université Paris 1 Panthéon-Sorbonne. Décembre 2020.
- M. Lodi gave the talk “Creative computing with Scratch” during the event “How to teach Informatics in K-8 education”. National CINI Laboratory “Informatics and School”, University of Milan. February 2020.
- M. Lodi gave the talk “Coding and programming... also in distance learning” at Ozzano Future Lab. May 2020. [4]

History of Programming Languages Apart from the above mentioned contributions, Martini has taken steps in a larger project—reflecting and tracing the interaction between mathematical logic and programming (languages), identifying some of the driving forces of this process.

Despite the insight of some of the pioneers (Turing, von Neumann, Curry, Böhm), programming the early computers was a matter of fiddling with small architecture-dependent details. Only in the sixties some form of “mathematical program development” will be in the agenda of some of the most influential players of that time. A “Mathematical Theory of Computation” is the name chosen by John McCarthy for his approach, which uses a class of recursively computable functions as an (extensional) model of a class of programs. It is the beginning of that grand endeavour to present programming as a mathematical activity, and reasoning on programs as a form of mathematical logic. An important part of this process is the standard model of programming languages-the informal assumption that the meaning of programs should be understood on an abstract machine with unbounded resources, and with true arithmetic. In [22] we present some crucial moments of this story, concluding with the emergence, in the seventies, of the need of more “intensional” semantics, like the sequential algorithms on concrete data structures.

11 Scientific production

11.1 Publications of the year

International journals

[1] M. Avanzini, U. Dal Lago and A. Yamada. ‘On probabilistic term rewriting’. In: *Science of Computer Programming* 185 (Jan. 2020), p. 102338. DOI: `10.1016/j.scico.2019.102338`. URL: `https://hal.inria.fr/hal-02381877`.

[2] M. Bravetti and G. Zavattaro. ‘Process calculi as a tool for studying coordination, contracts and session types’. In: *Journal of Logical and Algebraic Methods in Programming* 112 (Apr. 2020), p. 100527. DOI: `10.1016/j.jlamp.2020.100527`. URL: `https://hal.inria.fr/hal-03102438`.

[3] U. Dal Lago, F. Gavazzo and R. Tanaka. ‘Effectful applicative similarity for call-by-name lambda calculi’. In: *Theoretical Computer Science* 813 (Apr. 2020), pp. 234–247. DOI: `10.1016/j.tcs.2019.12.025`. URL: `https://hal.inria.fr/hal-02991694`.

[4] A. Durier, D. Hirschkoff and D. Sangiorgi. ‘Towards ’up to context’ reasoning about higher-order processes’. In: *Theoretical Computer Science* 807 (2020), pp. 154–168. DOI: `10.1016/j.tcs.2019.09.036`. URL: `https://hal.archives-ouvertes.fr/hal-01857391`.

[4] Available also on YouTube, see `https://www.youtube.com/watch?v=wyx6SoysTIU`.

[5] M. Falaschi, M. Gabbrielli, C. Olarte and C. Palamidessi. 'Dynamic slicing for Concurrent Constraint Languages'. In: *Fundamenta Informaticae* 177.3-4 (10th Dec. 2020), pp. 331–357. DOI: 10.3233/FI-2020-1992. URL: https://hal.archives-ouvertes.fr/hal-02423973.

[6] S. Guerrini, S. Martini and A. Masini. 'Quantum Turing Machines: Computations and Measurements'. In: *Applied Sciences* 10.16 (Aug. 2020). DOI: 10.3390/app10165551. URL: https://hal.inria.fr/hal-02915924.

[7] N. Kobayashi, U. Dal Lago and C. Grellois. 'On the Termination Problem for Probabilistic Higher-Order Recursive Programs'. In: *Logical Methods in Computer Science* (2nd Oct. 2020). DOI: 10.23638/LMCS-16(4:2)2020. URL: https://hal.inria.fr/hal-03120859.

[8] M. Lodi. 'Informatical Thinking'. In: *Olympiads in Informatics: An International Journal* 14 (Sept. 2020), pp. 113–132. DOI: 10.15388/ioi.2020.09. URL: https://hal.inria.fr/hal-02981734.

[9] D. Medić, C. A. Mezzina, I. Phillips and N. Yoshida. 'A parametric framework for reversible π-calculi'. In: *Information and Computation* 275 (Dec. 2020), p. 104644. DOI: 10.1016/j.ic.2020.104644. URL: https://hal.archives-ouvertes.fr/hal-03132462.

[10] C. Tian and D. Sangiorgi. 'Unique solutions of contractions, CCS, and their HOL formalisation'. In: *Information and Computation* 275 (Dec. 2020), p. 104606. DOI: 10.1016/j.ic.2020.104606. URL: https://hal.inria.fr/hal-03120563.

International peer-reviewed conferences

[11] B. Accattoli, U. Dal Lago and G. Vanoni. 'The Machinery of Interaction'. In: PPDP '20 - 22nd International Symposium on Principles and Practice of Declarative Programming. Bologna, Italy, 8th Sept. 2020, pp. 1–15. DOI: 10.1145/3414080.3414108. URL: https://hal.inria.fr/hal-03089342.

[12] M. Avanzini, G. Moser and M. Schaper. 'A Modular Cost Analysis for Probabilistic Programs'. In: *Proceedings of OOPSLA 2020.* OOPSLA 2020 - Conference on Object-oriented Programming, Systems, Languages, and Applications part of SPLASH 2020. Chicago / Online, United States, 15th Nov. 2020. URL: https://hal.inria.fr/hal-03013544.

[13] B. Barak, R. Crubillé and U. Dal Lago. 'On Higher-Order Cryptography'. In: ICALP 2020 - 47th International Colloquium on Automata, Languages, and Programming. Saarbrucken, Germany, 8th July 2020. DOI: 10.4230/LIPIcs.ICALP.2020.108. URL: https://hal.inria.fr/hal-03120781.

[14] F. Barbanera, I. Lanese and E. Tuosto. 'Choreography Automata'. In: COORDINATION 2020 - 22nd International Conference on Coordination Models and Languages. Valletta, Malta, 2020, pp. 86–106. DOI: 10.1007/978-3-030-50029-0_6. URL: https://hal.inria.fr/hal-03005377.

[15] F. Barbanera, I. Lanese and E. Tuosto. 'Composing Communicating Systems, Synchronously'. In: ISoLA 2020 - 9th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation. Rhodes, Greece, 19th Oct. 2020. URL: https://hal.inria.fr/hal-03005380.

[16] G. Barthe, R. Crubillé, U. Dal Lago and F. Gavazzo. 'On the Versatility of Open Logical Relations: Continuity, Automatic Differentiation, and a Containment Theorem'. In: *Programming Languages and Systems - 29th European Symposium on Programming, ESOP 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, {ETAPS} 2020, Dublin, Ireland, April 25-30, 2020, Proceedings.* ESOP 2020 - 29th European Symposium on Programming. Held as Part of the ETAPS 2020 - European Joint Conferences on Theory and Practice of Software. Dublin, Ireland, 25th Apr. 2020. URL: https://hal.inria.fr/hal-02991652.

[17] M. Bravetti, A. Francalanza, I. Golovanov, H. Hüttel, M. S. Jakobsen, M. K. Kettunen and A. Ravara. 'Behavioural Types for Memory and Method Safety in a Core Object-Oriented Language'. In: APLAS 2020 - 18th Asian Symposium on Programming Languages and Systems. Fukuoka / Virtual, Japan: https://conf.researchr.org/home/aplas-2020, 30th Nov. 2020. URL: https://hal.inria.fr/hal-03102375.

[18] U. Dal Lago, G. Guerrieri and W. Heijltjes. 'Decomposing Probabilistic Lambda-Calculi'. In: FOSSACS 2020 - Foundations of Software Science and Computation Structures - 23rd International Conference. Dublin, Ireland, 17th Apr. 2020, pp. 136–156. DOI: 10.1007/978-3-030-45231-5_8. URL: https://hal.inria.fr/hal-03120783.

[19] S. Giallorenzo, I. Lanese, F. Montesi, D. Sangiorgi and S. P. Zingaro. 'The Servers of Serverless Computing: A Formal Revisitation of Functions as a Service'. In: Recent Developments in the Design and Implementation of Programming Languages. Recent Developments in the Design and Implementation of Programming Languages, Italy, 27th Nov. 2020. DOI: 10.4230/OASIcs.Gabbrielli.2020.5. URL: https://hal.inria.fr/hal-03076904.

[20] I. Lanese and D. Medić. 'A General Approach to Derive Uncontrolled Reversible Semantics'. In: CONCUR 2020 - 31st International Conference on Concurrency Theory. Wien / Online, Austria, 1st Sept. 2020. DOI: 10.4230/LIPIcs.CONCUR.2020.33. URL: https://hal.inria.fr/hal-03005374.

[21] I. Lanese, I. Phillips and I. Ulidowski. 'An Axiomatic Approach to Reversible Computation'. In: FoSSaCS 2020 - 23rd International Conference on Foundations of Software Science and Computation Structures. Dublin, Ireland, 2020, pp. 442–461. DOI: 10.1007/978-3-030-45231-5_23. URL: https://hal.inria.fr/hal-03004421.

[22] S. Martini. 'The Standard Model for Programming Languages: The Birth of a Mathematical Theory of Computation'. In: Recent Developments in the Design and Implementation of Programming Languages. Bologna, Italy, 27th Nov. 2020. DOI: 10.4230/OASIcs.Gabbrielli.8. URL: https://hal.inria.fr/hal-03028634.

[23] D. Medić, C. Antares Mezzina, I. Phillips and N. Yoshida. 'Towards a formal account for software transactional memory'. In: RC 2020 - 12th International Conference on Reversible Computation. Oslo, Norway, 9th July 2020. URL: https://hal.inria.fr/hal-03005449.

Conferences without proceedings

[24] G. Geoffroy and P. Pistone. 'A Partial Metric Semantics of Higher-Order Types and Approximate Program Transformations'. In: Computer Science Logic, CSL 2021. Lubjana, Slovenia, 25th Jan. 2021. DOI: 10.4230/LIPIcs.CSL.2021.23. URL: https://hal.archives-ouvertes.fr/hal-03009790.

[25] D. Hirschkoff, E. Prebet and D. Sangiorgi. 'On the Representation of References in the Pi-Calculus'. In: CONCUR 2020 - 31st International Conference on Concurrency Theory. Vienna / Virtual, Austria, 2020. DOI: 10.4230/LIPIcs.CONCUR.2020.31. URL: https://hal.archives-ouvertes.fr/hal-03053368.

Scientific book chapters

[26] B. Aman, G. Ciobanu, R. Glück, R. Kaarsgaard, J. Kari, M. Kutrib, I. Lanese, C. A. Mezzina, Ł. Mikulski, R. Nagarajan, I. Phillips, G. M. Pinna, L. Prigioniero, I. Ulidowski and G. Vidal. 'Foundations of Reversible Computation'. In: *Reversible Computation: Extending Horizons of Computing - Selected Results of the COST Action IC1405*. Vol. 12070. 2020, pp. 1–40. DOI: 10.1007/978-3-030-47361-7_1. URL: https://hal.inria.fr/hal-03005384.

[27] M. Bravetti, S. Giallorenzo, J. Mauro, I. Talevi and G. Zavattaro. 'A Formal Approach to Microservice Architecture Deployment *'. In: *Microservices, Science and Engineering*. 2020. URL: https://hal.inria.fr/hal-03077047.

[28] F. Del Bonifro, M. Gabbrielli, G. Lisanti and S. P. Zingaro. 'Student Dropout Prediction'. In: *Artificial Intelligence in Education*. 30th June 2020, pp. 129–140. DOI: 10.1007/978-3-030-52237-7_11. URL: https://hal.inria.fr/hal-02983978.

[29] J. Hoey, I. Lanese, N. Nishida, I. Ulidowski and G. Vidal. 'A Case Study for Reversible Computing: Reversible Debugging of Concurrent Programs'. In: *Reversible Computation: Extending Horizons of Computing - Selected Results of the COST Action IC1405*. 2020, pp. 108–127. DOI: 10.1007/978-3-030-47361-7_5. URL: https://hal.inria.fr/hal-03005383.

[30] C. A. Mezzina, R. Schlatte, R. Glück, T. Haulund, J. Hoey, M. Holm Cservenka, I. Lanese, T. Æ. Mogensen, H. Siljak, U. P. Schultz and I. Ulidowski. 'Software and Reversible Systems: A Survey of Recent Activities'. In: *Reversible Computation: Extending Horizons of Computing - Selected Results of the COST Action IC1405*. 2020, pp. 41–59. DOI: 10.1007/978-3-030-47361-7_2. URL: https://hal.inria.fr/hal-03005386.

Doctoral dissertations and habilitation theses

[31] M. Lodi. 'Introducing Computational Thinking in K-12 Education: Historical, Epistemological, Pedagogical, Cognitive, and Affective Aspects'. Dipartimento di Informatica - Scienza e Ingegneria, Alma Mater Studiorum - Università di Bologna, 3rd Apr. 2020. URL: https://hal.inria.fr/tel-02981951.

Reports & preprints

[32] D. Hirschkoff, E. Prebet and D. Sangiorgi. *On the Representation of References in the pi-calculus*. 10th July 2020. DOI: 10.4230/LIPIcs.CONCUR.2020.31. URL: https://hal.archives-ouvertes.fr/hal-02895654.

[33] I. Lanese and D. Medić. *A General Approach to Derive Uncontrolled Reversible Semantics (TR)*. INRIA Sophia Antipolis - Méditerranée; Universita di Bologna, 1st Sept. 2020. URL: https://hal.archives-ouvertes.fr/hal-02902204.

11.2 Other

Scientific popularization

[34] M. Lodi, R. Davoli, R. Montanari and S. Martini. 'Informatica senza e con computer nella Scuola Primaria'. In: *Coding e oltre: l'Informatica nella scuola*. https://www.liscianiscuola.it/prodotto/coding-e-oltre-linformatica-nella-scuola/, Sept. 2020. URL: https://hal.inria.fr/hal-02379212.

[35] M. Lodi, S. Martini, M. Sbaraglia and S. P. Zingaro. '(Non) parliamo di pensiero computazionale'. In: *Proceedings of XXXIV Convegno Nazionale "Incontri con la Matematica"*. XXXIV Convegno Nazionale "Incontri con la Matematica". Castel San Pietro Terme / Online, Italy, 6th Nov. 2020. URL: https://hal.inria.fr/hal-02981770.

11.3 Cited publications

[36] 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.

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

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