RESEARCH CENTRE

Inria Paris Center

IN PARTNERSHIP WITH:

CNRS, Ecole normale supérieure de Paris

2022
ACTIVITY REPORT

Project-Team PARKAS

Parallélisme de Kahn Synchrone

IN COLLABORATION WITH: Département d'Informatique de l'Ecole Normale Supérieure

DOMAIN

Algorithmics, Programming, Software and Architecture

THEME

Embedded and Real-time Systems

Contents

Project-Team PARKAS

Creation of the Project-Team: 2012 January 01

Keywords

Computer sciences and digital sciences

A1.1.1. – Multicore, Manycore

A1.2.7. – Cyber-physical systems

A2.1.1. – Semantics of programming languages

A2.1.4. – Functional programming

A2.1.6. – Concurrent programming

A2.1.9. – Synchronous languages

A2.1.10. – Domain-specific languages

A2.2.4. – Parallel architectures

A2.2.8. – Code generation

A2.3. – Embedded and cyber-physical systems

A2.3.1. – Embedded systems

A2.3.2. – Cyber-physical systems

A2.3.3. – Real-time systems

A2.4.3. – Proofs

A3.4.5. – Bayesian methods

A6.2.1. – Numerical analysis of PDE and ODE

A6.2.2. – Numerical probability

A6.2.3. – Probabilistic methods

A6.4.1. – Deterministic control

A6.4.2. – Stochastic control

Other research topics and application domains

B5.2.1. – Road vehicles

B5.2.2. – Railway

B5.2.3. – Aviation

B6.4. – Internet of things

B6.6. – Embedded systems

B7.2.1. – Smart vehicles

B9.5.1. – Computer science

B9.5.2. – Mathematics

1 Team members, visitors, external collaborators

Research Scientists

- Guillaume Baudart [INRIA, ISFP]
- Timothy Bourke [INRIA, Researcher]
- Paul Feautrier [UDL, Emeritus]

Faculty Member

- Marc Pouzet [Team leader, SORBONNE UNIVERSITE, Professor, HDR]

PhD Students

- Paul Jeanmaire [INRIA]
- Baptiste Pauget [Ansys]
- Basile Pesin [INRIA]

Interns and Apprentices

- Gregoire Bussone [Inria, Intern, from Nov 2022]

Administrative Assistants

- Christine Anocq [INRIA]
- Nelly Maloisel [INRIA]

2 Overall objectives

Research in PARKAS focuses on the design, semantics, and compilation of programming languages which allow going from parallel deterministic specifications to target embedded code executing on sequential or multi-core architectures. We are driven by the ideal of a mathematical and executable language used both to program and simulate a wide variety of systems, including real-time embedded controllers in interaction with a physical environment (e.g., fly-by-wire, engine control), computationally intensive applications (e.g., video), and compilers that produce provably correct and efficient code.

The team bases its research on the foundational work of Gilles Kahn on the semantics of deterministic parallelism, the theory and practice of synchronous languages and typed functional languages, synchronous circuits, modern (polyhedral) compilation, and formal models to prove the correctness of low-level code.

To realize our research program, we develop languages (LUCID SYNCHRONE, REACTIVEML, LUCY-N, ZELUS), compilers, contributions to open-source projects (Sundials/ML), and formalizations in Interactive Theorem Provers of language semantics (Vélus and n-synchrony). These software projects constitute essential "laboratories": they ground our scientific contributions, guide and validate our research through experimentation, and are an important vehicle for long-standing collaborations with industry.

3 Research program

3.1 Programming Languages for Cyber-Physical Systems

We study the definition of languages for reactive and Cyber-Physical Systems in which distributed control software interacts closely with physical devices. We focus on languages that mix discrete-time and continuous-time; in particular, the combination of synchronous programming constructs with differential equations, relaxed models of synchrony for distributed systems communicating via periodic sampling or through buffers, and the embedding of synchronous features in a general purpose ML language.

The synchronous language SCADE based on synchronous languages principles, is ideal for programming embedded software and is used routinely in the most critical applications. But embedded design also involves modeling the control software together with its environment made of physical devices that are traditionally defined by differential equations that evolve on a continuous-time basis and approximated with a numerical solver. Furthermore, compilation usually produces single-loop code, but implementations increasingly involve multiple and multi-core processors communicating via buffers and shared-memory.

The major player in embedded design for cyber-physical systems is undoubtedly SIMULINK, with MODELICA a new player. Models created in these tools are used not only for simulation, but also for test-case generation, formal verification, and translation to embedded code. That said, many foundational and practical aspects are not well-treated by existing theory (for instance, hybrid automata), and current tools. In particular, features that mix discrete and continuous time often suffer from inadequacies and bugs. This results in a broken development chain: for the most critical applications, the model of the controller must be reprogrammed into either sequential or synchronous code, and properties verified on the source model have to be reverified on the target code. There is also the question of how much confidence can be placed in the code used for simulation.

We attack these issues through the development of the ZELUS research prototype, industrial collaborations with the SCADE team at ANSYS/Esterel-Technologies, and collaboration with Modelica developers at Dassault-Systèmes and the Modelica association. Our approach is to develop a *conservative extension* of a synchronous language capable of expressing in a single source text a model of the control software and its physical environment, to simulate the whole using off-the-shelf numerical solvers, and to generate target embedded code. Our goal is to increase faithfulness and confidence in both what is actually executed on platforms and what is simulated. The goal of building a language on a strong mathematical basis for hybrid systems is shared with the Ptolemy project at UC Berkeley; our approach is distinguished by building our language on a synchronous semantics, reusing and extending classical synchronous compilation techniques.

Adding continuous time to a synchronous language gives a richer programming model where reactive controllers can be specified in idealized physical time. An example is the so called quasi-periodic architecture studied by Caspi, where independent processors execute periodically and communicate by sampling. We have applied ZELUS to model a class of quasi-periodic protocols and to analyze an abstraction proposed for model-checking such systems.

Communication-by-sampling is suitable for control applications where value timeliness is paramount and lost or duplicate values tolerable, but other applications—for instance, those involving video streams—seek a different trade-off through the use of bounded buffers between processes. We developed the *n*-synchronous model and the programming language LUCY-N to treat this issue.

3.2 Compiling for Sequential and Multi-Core Processors

We develop compilation techniques for sequential and multi-core processors, and efficient parallel run-time systems for computationally intensive real-time applications (e.g., video and streaming). We study the generation of parallel code from synchronous programs, compilation techniques based on the polyhedral model, and the exploitation of synchronous Single Static Assignment (SSA) representations in general purpose compilers.

We consider distribution and parallelism as two distinct concepts.

- Distribution refers to the construction of multiple programs which are dedicated to run on specific computing devices. When an application is designed for, or adapted to, an embedded multiprocessor, the distribution task grants fine grained—design- or compilation-time—control over the mapping and interaction between the multiple programs.

- Parallelism is about generating code capable of efficiently exploiting multiprocessors. Typically this amounts to making (in)dependence properties, data transfers, atomicity and isolation explicit.

Compiling parallelism translates these properties into low-level synchronization and communication primitives and/or onto a runtime system.

We also see a strong relation between the foundations of synchronous languages and the design of compiler intermediate representations for concurrent programs. These representations are essential to the construction of compilers enabling the optimization of parallel programs and the management of massively parallel resources. Polyhedral compilation is one of the most popular research avenues in this area. Indirectly, the design of intermediate representations also triggers exciting research on dedicated runtime systems supporting parallel constructs. We are particularly interested in the implementation of non-blocking dynamic schedulers interacting with decoupled, deterministic communication channels to hide communication latency and optimize local memory usage.

While distribution and parallelism issues arise in all areas of computing, our programming language perspective pushes us to consider four scenarios:

1. designing an embedded system, both hardware and software, and codesign;
2. programming existing embedded hardware with functional and behavioral constraints;
3. programming and compiling for a general-purpose or high-performance, best-effort system;
4. programming large scale distributed, I/O-dominated and data-centric systems.

We work on a multitude of research experiments, algorithms and prototypes related to one or more of these scenarios. Our main efforts focused on extending the code generation algorithms for synchronous languages and on the development of more scalable and widely applicable polyhedral compilation methods.

3.3 Validation and Proof of Compilers

Compilers are complex software and not immune from bugs. We work on validation and proof tools for compilers to relate the semantics of source programs with the corresponding executable code.

The formal validation of a compiler for a synchronous language, or more generally for a language based on synchronous block diagrams, promises to reduce the likelihood of compiler-introduced bugs, the cost of testing, and also to ensure that properties verified on the source model hold of the target code. Such a validation would be complementary to existing industrial qualifications which certify the development process and not the functional correctness of a compiler. The scientific interest is in developing models and techniques that both facilitate the verification and allow for convenient reasoning over the semantics of a language and the behavior of programs written in it.

3.4 Probabilistic Reactive Programming

Most embedded systems evolve in an open, noisy environment that they only perceive through noisy sensors (e.g., accelerometers, cameras, or GPS). Another level of uncertainty comes from interactions with other autonomous entities (e.g., surrounding cars, or pedestrians crossing the street). Yet, to date, existing tools for cyber-physical system have had limited support for modeling uncertainty, to simulate the behavior of the systems, or to infer parameters from noisy observations. The classic approach consists in hand-coding robust stochastic controllers. But this solution is limited to well-understood and relatively simple tasks like the *lane following assist* system. However, no such controller can handle, for example, the difficult to anticipate behavior of a pedestrian crossing the street. A modern alternative is to rely on deep-learning techniques. But neural networks are black-box models that are notoriously difficult to understand and verify. Training them requires huge amounts of curated data and computing resources which can be problematic for corner-case scenarios in embedded control systems.

Over the last few years, Probabilistic Programming Languages (PPL) have been introduced to describe probabilistic models and automatically infer distributions of parameters from observed data. Compared to deep-learning approaches, probabilistic models show great promise: they overtly represent uncertainty, and they enable explainable models that can capture both expert knowledge and observed data.

A probabilistic reactive language provides the facilities of a synchronous language to write control software, with probabilistic constructs to model uncertainties and perform inference-in-the-loop. This approach offers two key advantages for the design of embedded systems with uncertainty: 1) Probabilistic models can be used to simulate an uncertain environment for early stage design and incremental development. 2) The embedded controller itself can rely on probabilistic components which implement skills that are out of reach for classic automatic controllers.

4 Application domains

4.1 Embedded Control Software

Embedded control software defines the interactions of specialized hardware with the physical world. It normally ticks away unnoticed inside systems like medical devices, trains, aircraft, satellites, and factories. This software is complex and great effort is required to avoid potentially serious errors, especially over many years of maintenance and reuse.

Engineers have long designed such systems using block diagrams and state machines to represent the underlying mathematical models. One of the key insights behind synchronous programming languages is that these models can be executable and serve as the base for simulation, validation, and automatic code generation. This approach is sometimes termed Model-Based Development (MBD). The SCADE language and associated code generator allow the application of MBD in safety-critical applications. They incorporate ideas from LUSTRE, LUCID SYNCHRONE, and other programming languages.

4.2 Hybrid Systems Design and Simulation

Modern embedded systems are increasingly conceived as rich amalgams of software, hardware, networking, and physical processes. The terms Cyberphysical System (CPS) or Internet-of-Things (IoT) are sometimes used as labels for this point of view.

In terms of modeling languages, the main challenges are to specify both discrete and continuous processes in a single *hybrid* language, give meaning to their compositions, simulate their interactions, analyze the behavior of the overall system, and extract code either for target control software or more efficient, possibly online, simulation. Languages like Simulink and Modelica are already used in the design and analysis of embedded systems; it is more important than ever to understand their underlying principles and to propose new constructs and analyses.

5 Highlights of the year

5.1 Awards

Ingkarat Rak-amnouykit, Ana Milanova, Guillaume Baudart, Martin Hirzel and Julian Dolby received a *ACM SIGSOFT Distinguished Papers* for "The Raise of Machine Learning Hyperparameter Constraints in Python Code" [14] at the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA) in July 2022.

6 New software and platforms

6.1 New software

6.1.1 Heptagon

Keywords: Compilers, Synchronous Language, Controller synthesis

Functional Description: Heptagon is an experimental language for the implementation of embedded real-time reactive systems. It is developed inside the Synchronics large-scale initiative, in collaboration with Inria Rhones-Alpes. It is essentially a subset of Lucid Synchrone, without type inference, type polymorphism and higher-order. It is thus a Lustre-like language extended with hierchical automata in a form very close to SCADE 6. The intention for making this new language and compiler is to develop new aggressive optimization techniques for sequential C code and compilation methods for generating parallel code for different platforms. This explains much of the simplifications we have made in order to ease the development of compilation techniques.

The current version of the compiler includes the following features: - Inclusion of discrete controller synthesis within the compilation: the language is equipped with a behavioral contract mechanisms, where assumptions can be described, as well as an "enforce" property part. The semantics of this latter is that the property should be enforced by controlling the behaviour of the node equipped with the contract. This property will be enforced by an automatically built controller, which will act on free controllable variables given by the programmer. This extension has been named BZR in previous works. - Expression and compilation of array values with modular memory optimization. The language allows the expression and operations on arrays (access, modification, iterators). With the use of location annotations, the programmer can avoid unnecessary array copies.

URL: `https://gitlab.inria.fr/synchrone/heptagon`

Contact: Gwenaël Delaval

Participants: Adrien Guatto, Brice Gelineau, Cédric Pasteur, Eric Rutten, Gwenaël Delaval, Léonard Gérard, Marc Pouzet

Partners: UGA, ENS Paris, Inria, LIG

6.1.2 SundialsML

Name: Sundials/ML

Keywords: Simulation, Mathematics, Numerical simulations

Scientific Description: Sundials/ML is a comprehensive OCaml interface to the Sundials suite of numerical solvers (CVODE, CVODES, IDA, IDAS, KINSOL). Its structure mostly follows that of the Sundials library, both for ease of reading the existing documentation and for adapting existing source code, but several changes have been made for programming convenience and to increase safety, namely:

solver sessions are mostly configured via algebraic data types rather than multiple function calls,

errors are signalled by exceptions not return codes (also from user-supplied callback routines),

user data is shared between callback routines via closures (partial applications of functions),

vectors are checked for compatibility (using a combination of static and dynamic checks), and

explicit free commands are not necessary since OCaml is a garbage-collected language.

Functional Description: Sundials/ML is an OCaml interface to the Sundials suite of numerical solvers (CVODE, CVODES, IDA, IDAS, KINSOL, ARKODE).

Release Contributions: Sundials/ML v6.0.0p0 adds support for v5.x and v6.x of the Sundials Suite of numerical solvers. This includes the latest Arkode features, many vectors, and nonlinear solvers.

URL: `http://inria-parkas.github.io/sundialsml/`

Publications: hal-01408230v1, hal-01967659v1

Contact: Timothy Bourke

Participants: Jun Inoue, Marc Pouzet, Timothy Bourke

6.1.3 Zelus

Keywords: Numerical simulations, Compilers, Embedded systems, Hybrid systems

Scientific Description: The Zélus implementation has two main parts: a compiler that transforms Zélus programs into OCaml programs and a runtime library that orchestrates compiled programs and numeric solvers. The runtime can use the Sundials numeric solver, or custom implementations of well-known algorithms for numerically approximating continuous dynamics.

Functional Description: Zélus is a new programming language for hybrid system modeling. It is based on a synchronous language but extends it with Ordinary Differential Equations (ODEs) to model continuous-time behaviors. It allows for combining arbitrarily data-flow equations, hierarchical automata and ODEs. The language keeps all the fundamental features of synchronous languages: the compiler statically ensure the absence of deadlocks and critical races, it is able to generate statically scheduled code running in bounded time and space and a type-system is used to distinguish discrete and logical-time signals from continuous-time ones. The ability to combines those features with ODEs made the language usable both for programming discrete controllers and their physical environment.

URL: `https://zelus.di.ens.fr`

Publications: hal-03051954v1, hal-02333603v1, hal-02426533v1, inria-00554271v1, hal-01242732v1, hal-00654113v1, hal-00909029v1, hal-01575621v4, hal-01575631v1, hal-00766726v1, hal-00938891v1, hal-00654112v1, hal-01879026v1, hal-01549183v2, hal-00938866v1

Contact: Marc Pouzet

Participants: Marc Pouzet, Timothy Bourke

Partner: ENS Paris

6.1.4 Vélus

Name: Verified Lustre Compiler

Keywords: Synchronous Language, Compilation, Software Verification, Coq, Ocaml

Functional Description: Vélus is a prototype compiler from a subset of Lustre to assembly code. It is written in a mix of Coq and OCaml and incorporates the CompCert verified C compiler. The compiler includes formal specifications of the semantics and type systems of Lustre, as well as the semantics of intermediate languages, and a proof of correctness that relates the high-level dataflow model to the values produced by iterating the generated assembly code.

Release Contributions: Vélus 3.0 introduces syntax and semantics for Lustre (previous versions only treated the normalized form of Lustre). It includes a verified normalization pass that transforms Lustre programs into NLustre programs.

URL: `https://velus.inria.fr`

Publications: hal-01817949, hal-03287572, hal-01512286, hal-01403830, tel-03068862, hal-02005639, hal-02426573, hal-03370264

Contact: Timothy Bourke

Participants: Timothy Bourke, Basile Pesin, Paul Jeanmaire, Marc Pouzet

6.1.5 MPPcodegen

Name: Source-to-source loop tiling based on MPP

Keywords: Source-to-source compiler, Polyhedral compilation

Functional Description: MPPcodegen applies a monoparametric tiling to a C program enriched with pragmas specifying the tiling and the scheduling function. The tiling can be generated by any convex polyhedron and translation functions, it is not necessarily a partition. The result is a C program depending on a scaling factor (the parameter). MPPcodegen relies on the MPP mathematical library to tile the iteration sets.

URL: `http://foobar.ens-lyon.fr/mppcodegen/`

Publication: hal-02493164

Authors: Christophe Alias, Guillaume Iooss, Sanjay Rajopadhye

Contact: Christophe Alias

Partner: Colorado State University

6.1.6 MPP

Name: MonoParametric Partitionning transformation

Keywords: Compilation, Polyhedral compilation

Functional Description: This library applies a monoparametric partitioning transformation to polyhedra and affine functions. This transformation is a subset of the parametric sized tiling transformation, specialized for the case where shapes depend only on a single parameter. Unlike in the general case, the resulting sets and functions remain in the polyhedral model.

URL: `https://github.com/guillaumeiooss/MPP`

Contact: Guillaume Iooss

6.1.7 ProbZelus

Keywords: Probabilistic Programming, Synchronous Language

Scientific Description: ProbZelus is a probabilistic reactive language which provides the facilities of a synchronous language to write control software, with probabilistic constructs to model uncertainties and perform inference-in-the-loop.

Functional Description: ProbZelus is built on top of Zelus a dataflow language à la Scade/Lustre and offers several streaming inference techniques including classic Sequential Monte Carlo (SMC) algorithms and semi-symbolic inference algorithm based on delayed sampling.

URL: `https://github.com/IBM/probzelus`

Authors: Guillaume Baudart, Louis Mandel, Eric Atkinson, Benjamin Sherman, Marc Pouzet, Michael Carbin

Contact: Guillaume Baudart

Partners: CSAIL, MIT, IBM

6.1.8 DeepStan

Keywords: Probabilistic Programming, Compilers, Stan, Pyro

Scientific Description: Stan is a probabilistic programming language that is popular in the statistics community, with a high-level syntax for expressing probabilistic models. Stan differs by nature from generative probabilistic programming languages like Pyro. DeepStan is a compiler from Stan to Pyro. Building on Pyro we can extend Stan with support for explicit variational inference guides, automatic guide generation, and deep probabilistic models.

Functional Description: The compiler is a fork of the Stanc3 compiler with two new backends for Pyro and NumPyro. The runtime is packaged as an independent Python library and contains the Stan standard library and thin wrapper for the Pyro/NumPyro runtime.

URL: `https://github.com/deepppl`

Contact: Guillaume Baudart

Participants: Guillaume Baudart, Louis Mandel

Partner: IBM

7 New results

7.1 Verified compilation of Lustre

Participants: Timothy Bourke, Paul Jeanmaire, Basile Pesin, Marc Pouzet.

Vélus is a compiler for a subset of LUSTRE and SCADE that is specified in the Coq [19] Interactive Theorem Prover (ITP). It integrates the CompCert C compiler [20, 18] to define the semantics of machine operations (integer addition, floating-point multiplication, etcetera) and to generate assembly code for different architectures. The research challenges are to

- to mechanize, i.e., put into Coq, the semantics of the programming constructs used in modern languages for Model-Based Development;
- to implement compilation passes and prove them correct;
- to interactively verify source programs and guarantee that the obtained invariants also hold of the generated code.

Work continued this year on this long-running project in two main directions: adding state machines, and developing constructive denotational models to facilitate interactive verification.

Adding higher-level constructs: This year, in the context of Basile Pesin's thesis project, we developed and refined a semantic model in Coq for hierarchical state machines, implemented algorithms to compile them into compositions of simpler constructions, and verified the algorithms correct. We generalized the definitions and algorithms for dependency analysis by adapting for use in Coq a preexisting technique based on *causality labels*. This technique permits, for example, to have different dependency relations in different branches of the `switch` construct. We completed a proof of determinism for the extended semantic model. This required the introduction of a novel intermediate model to simplify reasoning about variable dependencies and clock correctness.

We developed a small example to demonstrate these ideas, implemented a web interface to showcase the compiler, and submitted two articles on our new results.

Abstract Models and Program Verification: To date we have focused on proving the correctness of compilation passes. This involves specifying semantic models to define the input/output relation associated with a program, implementing compilation functions to transform the syntax of a program, and proving that the relation is unchanged by the functions. In addition to specifying compiler correctness, semantic models can also serve as a base for verifying individual programs. The challenge is to present and manipulate such detailed specifications in interactive proofs. The potential advantage is to be able to reason on abstract models and to obtain, via the compiler correctness theorem, proofs that apply to generated code. Making this idea work requires solving several scientific and technical challenges. It is the subject of Paul Jeanmaire's thesis.

This year we continued developing a Kahn-style semantics in Coq using C. Paulin-Mohring's library [21]. The model now includes most features from the dataflow core of Lustre, notably including node instances with subsampling. We have also treated the node reset construction in a limited setting (single input-single output nodes). Work continues to extend the model to account for multiple equations, rather than just a single equation with multiple expressions. We have formally shown that this new constructive model satisfies the relational semantic predicates used in the compiler correctness proof. These results required developing invariants and tools to reason about dependencies and fixpoints. One of the main challenges in developing a constructive model is to represent and reason explicitly about errors due to incorrect types, incorrect clocks, or partial operations from the underlying Clight language. We made good progress on definitions and proofs that treat these issues and work continues. We tried to eliminate the base clock argument in our semantic rules by working in a Present/Absent/Ready model where the new Ready tag indicates that a stream can be "pulled". Unfortunately, this does not work due to an underlying causality issue. This work was presented at TYPES 2022.

Glossary

Interactive Theorem Prover (ITP, also known as a *proof assistant*) Software for formal specification and proof, with features for generating and checking proofs, and extracting programs for later compilation

Model-Based Development (MBD) The specification of control software using block-diagrams, state machines, and other high-level constructions allowing programmers to focus on describing desired behaviour and to rely on automatic code generation to produce low-level executables.

7.2 Latency-based scheduling of synchronous programs

Participants: Timothy Bourke, Baptiste Pauget, Marc Pouzet.

External collaborators: Michel Angot, Vincent Bregeon, and Matthieu Boitrel, (Airbus).

It is sometimes desirable to compile a single synchronous language program into multiple tasks for execution by a real-time operating system. We have been investigating this question from three different perspectives.

Scheduling and code generation for periodic streams: In this approach, the top-level node of a Lustre program is distinguished from inner nodes. It may contain special annotations to specify the triggering and other details of node instances from which separate "tasks" are to be generated. Special operators are introduced to describe the buffering between top-level instances. Notably, different forms of the `when` and `current` operators are provided. Some of the operators are under-specified and a constraint solver is used to determine their exact meaning, that is, whether the signal is delayed by zero, one, or more cycles of the receiving clock, which depends on the scheduling of the source and destination nodes. Scheduling is formalized as a constraint solving problem based on latency constraints between some pairs of input/outputs that are specified by the designer.

This year we continued work on our prototype compiler, notably adding support for constraints on node sequencing and features for scheduling across two computation units. We developed an XML-RPC interface to the NEOS online ILP solvers to facilitate testing. We added features for finding conflicts between constraints and spent a lot of time debugging our compiler. We developed a new, far simpler encoding for end-to-end latency constraints and evaluated its efficacy. We formalized and refined the main code generation predicates and algorithms. We reimplemented the small ROSACE case study as a complement to our work on the full-scale industrial example. We wrote and submitted articles on our work.

This work is funded by a direct industrial contract with Airbus.

7.3 Sundials/ML: OCaml interface to Sundials Numeric Solvers

Participants: Timothy Bourke.

This year we made major updates to the Sundials/ML OCaml interface to support support v6.x of the Sundials Suite of numerical solvers. This required adding support for the new logging and profiling frameworks, continuing to adapt to changes in the ARKode solver, supporting new nvector and solver features, and adding new example programs. We fixed compilation and many bugs under OCaml 5.x. We also discovered and reported a problem in the new OCaml 5.x runtime. We presented our work at the OCaml Users in Paris (OUPS) meeting.

This library is being used in the Miking, OWL (OCaml Scientific Computing), and Zelus projects.

7.4 The Zelus Language

Participants: Guillaume Baudart, Marc Pouzet.

Zelus is our laboratory to experiment our research on programming languages for hybrid systems. It is devoted to the design and implementation of systems that may mix discrete-time/continuous-time signals and systems between those signals. It is first a synchronous language reminiscent of Lustre and Lucid Synchrone with the ability to define functions that manipulate continuous-time signals defined by Ordinary Differential Equations (ODEs) and zero-crossing events. The language is functional in the sense that a system is a function from signals to signals (not a relation). It provides some features from ML languages like higher-order and parametric polymorphism as well as dedicated static analyses.

Distribution of the language The language, its compiler and examples (release 2.1) are on GitHub. It is also available as an OPAM package. All the installation machinery has been greatly simplified.

The implementation of Zelus is now relatively mature. The language has been used in a collection of advances projects; the most important of the recent years being the design and implementation of ProbZelus on top of Zelus. This experiment called for several internal deep changes in the Zelus language.

One of the biggest troubles we faced when implementing Zelus was the lack of a tool to automatically test the compiler and to prototype language extensions before finding how to incorporate in the language and how to compile them. This is what motivated first our work on an *executable* semantics. The tool *zrun* works now quite well and, based on it, we have started a new implementation so that every pass of the compiler can be automatically tested.

7.4.1 Transparent Assertions for Hybrid Systems Models

Participants: Marc Pouzet.

External collaborator: Francois Bidet (Ecole Polytechnique).

Hybrid systems modeling languages and tools are used to write an executable model with mixed signals — discrete-time and continuous-time —, e.g., a model for the software and its physical environment. Given a numerical solver for differential equations, concrete simulation of a model computes an approximation of all the signals up to a given time horizon. While routinely used, concrete simulation raises an intrinsic difficulty when debugging a model: even a small change, e.g., adding a continuous-time block that only observe a signal, may change the simulation results of what is observed.

In 2022, we have worked on the following problem: how to express and implement *transparent observers* for a hybrid system modeling language so that the simulation with and without them give the same result and with no supplementary approximation at the frontier. We have defined an executable formal semantics of the concrete simulation loop first; and extended to the general case of a model that contains several, possibly nested, observers, each being approximated with its own numerical solver.

7.5 A Constructive Set-based synchronous semantics

Participants: Baptiste Pauget, Marc Pouzet.

We have pursued our work on the definition of an executable semantics for a synchronous language. The result is an interpreter (named `zrun`) implemented in purely functional style, in OCaml. A paper have been submitted for publication.

The software development is part of the working branch of Zelus (). During year 2022, we have started the implementation of new version of Zelus that uses `zrun` so that very compilation step is automatically tested.

7.6 A Memory Aware Synchronous data-flow language for the Compilation of Computer Intensive Applications

Participants: Baptiste Pauget, Marc Pouzet.

External collaborators: Jean-Louis Colaco (ANSYS, Toulouse).

We have pursed our work on the design and compilation of programming languages for safety-critical embedded software that combines complex control code and intensive computations using arrays. This year, we have focused on the compilation and more precisely the sequential code generation for programs that manipulate data-structures (mainly arrays) expressed in a purely functional synchronous language. The considered target for code generation is a sequential program, e.g., C, where all the memory must be known statically. Although extensively studied, the use of arrays is still challenging when coupled with safety critical real-time software constraints. This is because the functional programming style (with operations like `map`, `fold`, `concat`, etc.), that is recommended for verification, does not mix well with the static memory management imposed by targeted applications.

Here are two concrete examples that makes compilation challenging.

- Functional descriptions encourages the definition of data-structures by parts that are aggregated at some point (e.g. with array concatenation). Compilation should avoid constructing each part separately and then moving them into the resulting structure, because each of them create a copy.
- Functional updates, which semantically return a fresh array with only some of the element that change, should be implemented, as much as possible without any copies, by mutating the concrete data-structure, if data-dependences allow. Unfortunately, classical techniques for implementing functional update efficiently in general purpose functional languages are not applicable in the context of a language for real-time applications. Indeed, these techniques need dynamic allocation (an extra indirection is added to the original array for any update) and they change the WCET (worst-case execution time).

Taking care of memory has two main properties: (i) it reduces memory footprint and (ii) it avoids needless copies, that have a negative impact on the WCET, a central parameter for real-time application.

Instead of considering optimization passes to remove useless copies (such the built-in-place optimization of Sisal [1] and previous works done at PARKAS [5] we have designed a language that allows precise memory specification through compiler verified memory location hints, while remaining fully declarative. This builds on two key ideas:

- A static knowledge of all size of data-structures in the program. In order to allocate arrays with respect of others, their size must be known at compile time. The type-system with a *size polymorphism* mechanism proposed last year reveals to be very adequate: the type of every expression expresses its size, including for array expressions.

- A description of memory locations, that extends previous ideas done at PARKAS in 2012 [2]: each expression is associated to a *location* coupled with a description of how to access its parts. It is possible, for example, to express that the output of a function is located at the same place as one of its inputs.

- All copies must be explicit and are the only place where the compiler introduces a copy, with no change on the semantics (a copy is the identity function). In particular, building an array or a tuple that would normally need a copy do not. Instead, they produce a constraint on locations.

For example, array transformations such as reversal, transposition or concatenation do not need any copy. Rather than generating fresh arrays (that would need to be removed by a fragile optimization step), these operations only express changes on indexes to access them.

These ideas have been experimented in a prototype. It allows to generate sequential code with a precise control of memory while keeping a straightforward compilation process, with few transformations and no optimizations.

[1]: Gaudiot, J.-L., Bohm, W., Najjar, W., DeBoni, T., Feo, J., and Miller, P. The Sisal model of functional programming and its implementation. In Proceedings of IEEE International Symposium on Parallel Algorithms Architecture Synthesis (1997), IEEE, pp. 112–123.

[2] Gérard, L., Guatto, A., Pasteur, C., and Pouzet, M. A Modular Memory Optimization for Synchronous Data-Flow Languages. Application to Arrays in a Lustre Compiler. In Languages, Compilers and Tools for Embedded Systems (LCTES'12) (Beijing, June 12-13 2012), ACM. Best paper award.

7.7 Reactive Probabilistic Programming

Participants: Guillaume Baudart, Marc Pouzet.

External collaborators: Louis Mandel (IBM), Erik Atkinson, Michael Carbin and Charles Yuan (MIT), Waïss Azizian, Marc Lelarge, Reyyan Tekin (Inria).

Synchronous languages were introduced to design and implement real-time embedded systems with a (justified) enphasis on determinacy. Yet, they interact with a physical environment that is only partially known and are implemented on architectures subject to failures and noise (e.g., channels, variable communication delays or computation time). Dealing with uncertainties is useful for online monitoring, learning, statistical testing or to build simplified models for faster simulation. Actual synchronous and languages provide limited support for modeling the non-deterministic behaviors that are omnipresent in embedded systems. ProbZelus is a probabilistic extension of the synchronous language Zelus for the design of reactive probabilistic models in interaction with an environment.

This year we continued this project along three main directions: 1) automatic parallelization of sequential Monte Carlo methods, 2) a novel semi-symbolic streaming inference method, and 3) embedding ProbZelus ideas in Julia.

JAX-based parallel inference for reactive probabilistic programming (with L. Mandel and R. Tekin). Compared to classic synchronous programming where a program is a stream processor mapping a stream of inputs to a stream of outputs, a ProbZelus models computes a stream of distributions from a stream of inputs. Unfortunately, this inference problem is in general intractable. Monte Carlo inference techniques thus rely on many independent executions to compute accurate approximations. These methods are expensive but can be parallelized. We used JAX to parallelize ProbZelus reactive inference engine.

JAX is a recent library to compile Python code which can then be executed on massively parallel architectures such as GPUs or TPUs. We implemented a new reactive inference engine implemented in JAX and the new associated JAX backend for ProbZelus. We then showed on existing examples that our new parallel implementation outperforms the original sequential implementation for a high number of particles.

The main article *JAX-Based Parallel Inference for Reactive Probabilistic Programming* was presented at the ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES 2022) [13]. A preliminary version of this work in French was also presented at the Journées Francophones des Langages Applicatifs (JFLA 2022) [15].

Semi-symbolic Inference for Efficient Streaming Probabilistic Programming (with Erik Atkinson, Michael Carbin, L. Mandel, and Charles Yuan).

Efficient inference is often possible in a streaming context using Rao-Blackwellized particle filters (RBPFs), which exactly solve inference problems when possible and fall back on sampling approximations when necessary. While RBPFs can be implemented by hand to provide efficient inference, the goal of streaming probabilistic programming is to automatically generate such efficient inference implementations given input probabilistic programs.

We proposed semi-symbolic inference, a technique for executing probabilistic programs using a runtime inference system that automatically implements Rao-Blackwellized particle filtering. To perform exact and approximate inference together, the semi-symbolic inference system manipulates symbolic distributions to perform exact inference when possible and falls back on approximate sampling when necessary. This approach enables the system to implement the same RBPF a developer would write by hand. To ensure this, we identify closed families of distributions ś such as linear-Gaussian and finite discrete models on which the inference system guarantees exact inference. We have implemented the runtime inference system in the ProbZelus streaming probabilistic programming language. Despite an average 1.6x slowdown compared to the state of the art on existing benchmarks, our evaluation shows that speedups of 3x-87x are obtainable on a new set of challenging benchmarks we have designed to exploit closed families.

The main article *Semi-symbolic Inference for Efficient Streaming Probabilistic Programming* was presented at the Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2022) and published in the Proceedings of the ACM on Programming Languages (PACMPL) [12].

OnlineSampling.jl (with Marc Lelarge and Waïss Azizian). OnlineSampling.jl is an embedded reactive probabilistic language in Julia. Inspired by ProbZelus we designed a domain specific language for describing reactive probabilistic models using Julia macros. Following ProbZelus ideas, the inference method is a Rao-Blackwellised particle filter, a semi-symbolic algorithm which tries to analytically compute closed-form solutions, and falls back to a particle filter when symbolic computations fail. For Gaussian random variables with linear relations, we use belief propagation instead of delayed sampling if the factor graph is a tree. We can thus compute exact solutions for a broader class of models.

This project is an opensource Julia package available on GitHub and was presented at JuliaCon 2022.

7.8 Extracting Hyperparameters Constraints from Machine Learning Operators Code

Participants: Guillaume Baudart.

External collaborators: Ingkarat Rak-amnoukit, Ana Milanova (RPI), Martin Hirzel, Julian Dolby (IBM).

Machine-learning operators often have correctness constraints that cut across multiple hyperparameters and/or data. Violating these constraints causes the operator to raise runtime exceptions. We designed an interprocedural weakest-precondition analysis for Python to extract hyperparameter constraints. The analysis is mostly static, but to make it tractable for typical Python idioms in machine-learning libraries, it selectively switches to the concrete domain for some cases. We evaluated the analysis on 181 operators from existing ML libraries.

Building on last year preliminary results, we continued this work and the main article *The Raise of Machine Learning Hyperparameter Constraints in Python Code* was presented at the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2022) [14] where it received an *ACM SIGSOFT Distinguished Paper Award.*

8 Bilateral contracts and grants with industry

8.1 Bilateral contracts with industry

Collaboration with Airbus

Participants: Timothy Bourke, Marc Pouzet.

Our work on multi-clock Lustre programs is funded by a contract with Airbus.

9 Partnerships and cooperations

9.1 National initiatives

9.1.1 ANR

ANR JCJC FidelR

Participants: Timothy Bourke, Basile Pesin, Marc Pouzet, Paul Jeanmaire.

The ANR JCJC project "FidelR" led by T. Bourke began in 2020 and continues for four years.

9.1.2 FUI: Fonds unique interministériel

Modeliscale contract (AAP-24)

Participants: Timothy Bourke, Marc Pouzet.

Using Modelica at scale to model and simulate very large Cyber-Physical Systems. Principal industrial partner: Dassault-Systèmes. INRIA contacts are Benoit Caillaud (HYCOMES, Rennes) and Marc Pouzet (PARKAS, Paris).

9.1.3 Programme d'Investissements d'Avenir (PIA)

Participants: Timothy Bourke, Marc Pouzet, Baptiste Pauget.

ES3CAP collaborative project (Bpifrance) Develop a software and hardware platform for tomorrow's intelligent systems. PARKAS collaborates with the industrial participants ANSYS/Esterel Technologies, Kalray, and Safran Electronics & Defense. Inria contacts are Marc Pouzet (PARKAS, Paris) and Fabrice Rastello (CORSE, Grenoble).

9.1.4 Others

Inria Project Lab (IPL) Modeliscale

Participants: Timothy Bourke, Marc Pouzet.

This project treats the modelling and analysis of Cyber-Physical Systems at large scale. The PARKAS team contributes their expertise in programming language design for reactive and hybrid systems to this multi-team effort.

10 Dissemination

10.1 Promoting scientific activities

10.1.1 Scientific events: organisation

General chair, scientific chair

- Timothy Bourke served as vice-président (2022) and président (2023) of the Journées Francophones des Langages Applicatifs.

Member of the organizing committees

- Timothy Bourke served as the Real-Time Pitches chair for the 2022 Euromicro Conference on Real-Time Systems (ECRTS 2022).

10.1.2 Scientific events: selection

Member of the conference program committees

- Guillaume Baudart served on the program committee of the Workshop on Reactive and Event-Based Languages and Systems (REBLS 2022).
- Timothy Bourke served on the program committee of the 2022 Euromicro Conference on Real-Time Systems (ECRTS 2022).
- Timothy Bourke served on the program committee of the 2022 International Conference on Embedded Software (EMSOFT 2022).
- Timothy Bourke served on the program committees of the 2022 American and Asian Modelica Conferences.
- Timothy Bourke served on the program committee of the 2023 Journées Francophones des Langages Applicatifs (JFLA 2023)

Reviewer

- Timothy Bourke contributed expert reviews for the conferences:
  - 2022 ACM Principles of Programming Languages (POPL 2022)
  - 2022 Computer Aided Verification (CAV 2022)
  - 2022 IEEE Real-Time Systems Symposium (RTSS 2022)

10.1.3 Journal

Reviewer - reviewing activities

- Guillaume Baudart reviewed an article for the International Journal of Approximate Reasoning (IJAR)
- Timothy Bourke reviewed articles for the journals:
  - Springer Real-Time Systems (RTS)
  - ACM Transactions on Embedded Computer Systems (TECS)
  - Elsevier Theoretical Computer Science (TCS)

10.1.4 Invited talks

- Guillaume Baudart was an invited speaker at the Probabilistic Programming Workshop at College de France
- Marc Pouzet was an invited speaker at "Journées scientifiques de l'INRIA", for a retrospetive talk on synchronous programming, in November 2023.

10.1.5 Leadership within the scientific community

- Timothy Bourke contributed a blog post to the ACM SIGBED website, describing the EMSOFT 2021 conference.

10.2 Teaching - Supervision - Juries

10.2.1 Teaching

- Marc Pouzet is Director of Studies for the CS department, at ENS.
- Licence : Marc Pouzet & Timothy Bourke: "Operating Systems" (L3), Lectures and TDs, ENS, France.
- Master : Marc Pouzet, Guillaume Baudart, & Timothy Bourke, "Models and Languages for Programming Reactive Systems" (M1), Lectures and TDs, ENS, France.
- Master: Marc Pouzet & Timothy Bourke: "Synchronous Systems" (M2), Lectures and TDs, MPRI, France
- Master: Marc Pouzet: "Synchronous Reactive Languages" (M2), Lectures, Master CPS (Cyber-physical Systems, led by Giorgio Mover (École Polytechnique).
- Master: Marc Pouzet "The Elements of Computing Systems". Cycle pluridisciplinaire d'études supérieures (CPES), L2.
- Master: Timothy Bourke: "A Programmer's introduction to Computer Architectures and Operating Systems" (M1), École Polytechnique, France
- Master: Timothy Bourke and Basile Pesin presented two lectures and TPs on Synchronous Languages in Carlos Agon's course on concurrent models at Sorbonne Université.
- Master: Guillaume Baudart: "Synchronous Programming" (M2), TDs, Université de Paris, France
- Master: Guillaume Baudart: "Probabilistic Programming Languages" (M2), Lectures and TDs, MPRI, France
- Aggregation: Guillaume Baudart: "Introduction to Software Engineering" (préparation à l'aggrégation d'informatique), Lectures and TDs, France
- Bachelor: Timothy Bourke: "A Programmer's introduction to Computer Architectures and Operating Systems" (L2), École Polytechnique, France

10.2.2 Supervision

- PhD in progress: Paul Jeanmaire, 3rd year, supervised by Timothy Bourke and Marc Pouzet.
- PhD in progress: Baptiste Pauget, 3rd year, supervised by Marc Pouzet.
- PhD in progress: Basile Pesin, 3rd year, supervised by Timothy Bourke and Marc Pouzet.
- PhD in progress: Astyax Nourel , 2nd year, supervised by Adrien Guatto (IRIF, Univ. of Paris) and Marc Pouzet.
- Master: Gregoire Bussone, M2 research internship, supervised by Guillaume Baudart and Christine Tasson (March-August 2022).
- Bachelors internship: Carolina Nina de Matos, supervised by Timothy Bourke.
- Polytechnique Informatics project: Jonathan Arnoult, supervised by Timothy Bourke.
- Bachelors project: Vrushank Agrawal supervised by Timothy Bourke.

10.2.3 Juries

- Internships Timothy Bourke & Guillaume Baudart participated in reviewing the L3 and M1 internships of students at the ENS, France.
- Timothy Bourke was an examiner for the thesis defenses of:
  - François Bidet
  - Frédéric Fort
  - Olivier Nicole
  - Hugo Pompougnac

10.3 Popularization

10.3.1 Internal or external Inria responsibilities

- Timothy Bourke served on the jury for CRCN/ISFP recrutements at Inria Paris.
- Timothy Bourke participated in a number of thesis monitoring committees.

11 Scientific production

11.1 Major publications

[1] G. Baudart, L. Mandel, E. Atkinson, B. Sherman, M. Pouzet and M. Carbin. 'Reactive probabilistic programming'. In: *PLDI 2020 - 41th ACM SIGPLAN International Conference in Programming Language Design and Implementation*. London / Virtual, United Kingdom, June 2020. DOI: 10.1145/3385412.3386009. URL: https://hal.inria.fr/hal-03051954.

[2] T. Bourke, L. Brun, P.-E. Dagand, X. Leroy, M. Pouzet and L. Rieg. 'A Formally Verified Compiler for Lustre'. In: *PLDI 2017 - 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM. Barcelone, Spain, June 2017. URL: https://hal.inria.fr/hal-01512286.

[3] T. Bourke, F. Carcenac, J.-L. Colaço, B. Pagano, C. Pasteur and M. Pouzet. 'A Synchronous Look at the Simulink Standard Library'. In: *EMSOFT 2017 - 17th International Conference on Embedded Software*. Seoul, South Korea: ACM Press, Oct. 2017, p. 23. URL: https://hal.inria.fr/hal-01575631.

[4] T. Bourke, J.-L. Colaço, B. Pagano, C. Pasteur and M. Pouzet. ‘A Synchronous-based Code Generator For Explicit Hybrid Systems Languages’. In: *International Conference on Compiler Construction (CC)*. LNCS. London, United Kingdom, July 2015. URL: https://hal.inria.fr/hal-01242732.

[5] L. Gérard, A. Guatto, C. Pasteur and M. Pouzet. ‘A modular memory optimization for synchronous data-flow languages: application to arrays in a lustre compiler’. In: *Proceedings of the 13th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, Tools and Theory for Embedded Systems*. Beijing, China: ACM, June 2012, pp. 51–60. DOI: 10.1145/2248418.2248426. URL: https://hal.inria.fr/hal-00728527.

[6] J. C. Juega, S. Verdoolaege, A. Cohen, J. I. Gómez, C. Tenllado and F. Catthoor. ‘Patterns for parallel programming on GPUs’. In: *Patterns for parallel programming on GPUs*. Ed. by F. Magoulès. Vol. Evaluation of State-of-the-Art Parallelizing Compilers Generating CUDA Code for Heterogeneous CPU/GPU Computing. ISBN 978-1-874672-57-9. Saxe-Cobourg, 2013. URL: https://hal.archives-ouvertes.fr/hal-01257261.

[7] L. Mandel, F. Plateau and M. Pouzet. ‘Static Scheduling of Latency Insensitive Designs with Lucy-n’. In: *FMCAD 2011 - Formal Methods in Computer Aided Design*. Austin, TX, United States, Oct. 2011. URL: https://hal.inria.fr/hal-00654843.

[8] R. Morisset, P. Pawan and F. Zappa Nardelli. ‘Compiler testing via a theory of sound optimisations in the C11/C++11 memory model’. In: *PLDI 2013 - 34th ACM SIGPLAN conference on Programming language design and implementation*. Seattle, WA, United States: ACM, June 2013, pp. 187–196. DOI: 10.1145/2491956.2491967. URL: https://hal.inria.fr/hal-00909083.

[9] A. Pop and A. Cohen. ‘OpenStream: Expressiveness and Data-Flow Compilation of OpenMP Streaming Programs’. In: *ACM Transactions on Architecture and Code Optimization* 9.4 (2013). Selected for presentation at the HiPEAC 2013 Conf. DOI: 10.1145/2400682.2400712. URL: https://hal.inria.fr/hal-00786675.

[10] J. Sevcik, V. Vafeiadis, F. Zappa Nardelli, S. Jagannathan and P. Sewell. ‘CompCertTSO: A Verified Compiler for Relaxed-Memory Concurrency’. In: *Journal of the ACM (JACM)* 60.3 (2013), art. 22:1–50. DOI: 10.1145/2487241.2487248. URL: https://hal.inria.fr/hal-00909076.

[11] V. Vafeiadis, T. Balabonski, S. Chakraborty, R. Morisset and F. Zappa Nardelli. ‘Common compiler optimisations are invalid in the C11 memory model and what we can do about it’. In: *POPL 2015 - 42nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. Mumbai, India, Jan. 2015. URL: https://hal.inria.fr/hal-01089047.

11.2 Publications of the year

International journals

[12] E. Atkinson, C. Yuan, G. Baudart, L. Mandel and M. Carbin. ‘Semi-symbolic inference for efficient streaming probabilistic programming’. In: *Proceedings of the ACM on Programming Languages* 6 (31st Oct. 2022), pp. 1668–1696. DOI: 10.1145/3563347. URL: https://hal.archives-ouvertes.fr/hal-03891782.

International peer-reviewed conferences

[13] G. Baudart, L. Mandel and R. Tekin. ‘JAX based parallel inference for reactive probabilistic programming’. In: *International Conference on Languages, Compilers, and Tools for Embedded Systems*. LCTES’22 - 23rd ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems. San Diego CA, United States, 14th June 2022. DOI: 10.1145/3519941.3535066. URL: https://hal.archives-ouvertes.fr/hal-03891766.

[14] I. Rak-Amnouykit, A. Milanova, G. Baudart, M. Hirzel and J. Dolby. ‘The raise of machine learning hyperparameter constraints in Python code’. In: *International Symposium on Software Testing and Analysis*. ISSTA 2022 - 31st ACM SIGSOFT International Symposium on Software Testing and Analysis. Virtual, South Korea, 18th July 2022. DOI: 10.1145/3533767.3534400. URL: https://hal.archives-ouvertes.fr/hal-03891774.

National peer-reviewed Conferences

[15] G. Baudart, L. Mandel, M. Pouzet and R. Tekin. 'Inférence parallèle pour un langage réactif probabiliste'. In: *Journées Francophones des Langages Applicatifs*. 33èmes Journées Francophones des Langages Applicatifs. 6-23. Saint-Médard-d'Excideuil, France, 2nd Feb. 2022. URL: https://hal.inria.fr/hal-03626762.

[16] J.-L. Colaço, B. Pauget and M. Pouzet. 'Inférer et vérifier les tailles de tableaux avec des types polymorphes'. In: *Journées Francophones des Langages Applicatifs*. 33èmes Journées Francophones des Langages Applicatifs. Saint-Médard-d'Excideuil, France, 2nd Feb. 2022, pp. 140–164. URL: https://hal.inria.fr/hal-03626802.

Scientific books

[17] C. Keller, T. Bourke, S. Blazy, F. Bour, G. Bury, S. Dumbrava, D. Gallois-Wong, A. Guatto, D. Janin, M. Kerjean, L. Pellissier, M. Pereira, A. Trieu and Y. Zakowski. *33èmes Journées Francophones des Langages Applicatifs*. Saint-Médard-d'Excideuil, France, 28th June 2022, pp. 1–292. URL: https://hal.inria.fr/hal-03689075.

11.3 Cited publications

[18] S. Blazy, Z. Dargaye and X. Leroy. 'Formal Verification of a C Compiler Front-End'. In: *FM 2006: Int. Symp. on Formal Methods*. Vol. 4085. Lecture Notes in Computer Science. Springer-Verlag, 2006, pp. 460–475. URL: http://gallium.inria.fr/~xleroy/publi/cfront.pdf.

[19] *The Coq proof Assistant*. http://coq.inria.fr. 2019.

[20] X. Leroy. *The Compcert verified compiler*. 2009. URL: http://compcert.inria.fr/doc/index.html.

[21] C. Paulin-Mohring. 'A constructive denotational semantics for Kahn networks in Coq'. In: *From Semantics to Computer Science: Essays in Honour of Gilles Kahn*. Ed. by Y. Bertot, G. Huet, J.-J. Lévy and G. Plotkin. Cambridge, UK: Cambridge University Press, 2009, pp. 383–413. URL: https://hal.inria.fr/inria-00431806/document.