RESEARCH CENTRE

Inria Paris Centre

IN PARTNERSHIP WITH:

CNRS, Ecole normale supérieure de Paris

2023

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

Inria

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]

Faculty Member

- Marc Pouzet [Team leader, Ecole normale supérieure, Professor, HDR]

PhD Students

- Gregoire Bussone [ENS Paris]
- Paul Jeanmaire [ENS PARIS, from Sep 2023]
- Paul Jeanmaire [INRIA, until Aug 2023]
- Baptiste Pauget [Ansys, until Oct 2023]
- Basile Pesin [INRIA, until Oct 2023]

Interns and Apprentices

- Vrushank Agrawal [INRIA, Intern, until Mar 2023]
- Victor Deng [INRIA, Intern, from Feb 2023 until Jul 2023]
- Antoine Grimod [ENS PARIS, from Apr 2023 until Aug 2023]
- Paul Robert [INRIA, Intern, from Mar 2023 until Aug 2023]

Administrative Assistants

- Christine Anocq [INRIA, until Jan 2023]
- Laurence Bourcier [INRIA]
- Nelly Maloisel [INRIA]

External Collaborator

- Paul Feautrier [ENS Lyon, Emeritus]

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 New software, platforms, open data

5.1 New software

5.1.1 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

5.1.2 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

5.1.3 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, IBM

5.1.4 presseail

Name: All-in-Lustre Compiler

Keywords: Embedded systems, Compilers, Synchronous Language, Real-time application

Functional Description: The input to the compiler is the rate-synchronous language described in our ECRTS 2023 article. The compiler generates and Integer Linear Programming (ILP) problem that includes data dependency and resource constraints. The problem is solved using an external solver and the resulting schedule is used by the compiler to generate sequential code using a generalization of the modular clock-driven compilation scheme used in modern Lustre/Scade compilers. The compiler implements special features for analyzing and eliminating cyclic data dependencies.

Release Contributions: First version described in the ECRTS 2023 publication.

Contact: Timothy Bourke

5.1.5 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

5.1.6 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

5.1.7 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

5.1.8 ZRun

Name: The ZRun Synchronous Language Interpreter

Keywords: Formal semantics, Interpreter, Ocaml, Reactive programming

Functional Description: ZRun is an executable semantics of a synchronous data-flow language. It takes the form of a purely functional interpreter and is implemented in OCaml. The input of Zrun is a large subset of the language Zélus, but only its discrete-time (synchronous) subset. The basic primitives are those of Lustre: a unit non-initialized delay (pre), the initialization operator (->), the initialized delay (fby), and streams can be defined by mutually recursive definitions. It also provides richer programming constructs that were introduced in Lucid Synchrone and Scade 6, but are not in Lustre: the by-case definition of streams, the last computed value of a signal, hierarchical automata with parameters, stream functions with static parameters that are either know at compile time or at instanciation time, and two forms of iteratiors on arrays: the "forward" to perform an iteration in time, the "foreach" to perform an iteration on space.

The objective of this prototype is to give a reference executable semantics that is independent of a compiler. It can be used, e.g., as an oracle for compiler testing, to execute unfinished programs or programs that are semantically correct but are statically rejected by the compiler.

Release Contributions: Branch Master (2000) - v1.x. - first-order language, streams, hierarchical automata, by-case definition of streams, operator last.

Branch Works (2023): - v2.x - static higher-order, hierarchical automata with parameters, valued signals. - arrays, - "forward" and "foreach" iterations.

URL: https://github.com/marcpouzet/zrun

Contact: Marc Pouzet

6 New results

6.1 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 [27] Interactive Theorem Prover (ITP). It integrates the CompCert C compiler [28, 25] 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: improving the compilation of shared variables, and developing constructive denotational models to facilitate interactive verification.

Compiling shared variables: This year, in the context of Basile Pesin's thesis project, we completed the work on the semantics and compilation of hierarchical state machines and related control structures. Shared variables, defined with a `last` value, are a useful enhancment of state machines. They allow access to the previous value of a variable relative to the whole state machine, rather than just the current state. Importantly, they permit implicit completion: for a shared variable `x`, if no explicit definition is given, then `x = last x`. This construct was already added to the Coq-based semantics, but it was compiled away early in the compiler, which often results in unnecessary copying in the generated code. This year, we modified the compiler so that `last` variables are carried right through to the back-end passes which faciliates the elimination of redundant assignments and the associated correctness proofs. This improvement required changes to all the intermediate languages, semantic definitions, and compilation algorithms. In particuarly, we found it necessary to modify the Stc intermediate language to include both 'next' and 'last' definitions. In terms of expressivity, only one such form is necesary, but having both facilitates optimizations and their correctness proofs, at the expense of more complicated semantics and scheduling. Basile Pesin presented this work at EMSOFT 2023 [17] and defended his thesis in October 2023 [31].

Denotational semantics for 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 [30]. The model now treats the dataflow core of Lustre as presented in our EMSOFT 2021 article [26] with the generalization to enumerated types. We show that, under specific conditions, the denotational model satisfies the relational predicates used in the compiler correctness proof. This allows us to strengthen the overall compiler correctness theorem. Rather than state "If a semantics exists for a program, then it is preseved by the generated code", we show that "Under specific conditions, a semantics exists and it is preserved by the generated code". The "specific conditions" are, as usual, that the source program satisfies typing and clock typing rules, but also, that it is not subject to run-time errors. Run-time errors cannot be ignored in our context of end-to-end proof. The CompCert definitions for several arithmetic and logical operators are partial, for example, integer division by zero is not defined. Such partiality simply propogates to the the Vélus relational model, but the denotational model is a total function and operator failures must thus be modeled explicitly. We expressed the absence of run-time errors as a predicate over the dynamic behavior of a program. We implemented a simple static analysis, that nevertheless suffices for many practical programs, and showed that it is a sufficient condition for the absence of run-time errors. The next version of the Vélus compiler will now print warning messages if the source program uses features not treated in the denotational model or if the simple static analysis cannot guarantee the absence of errors. In this case, it becomes the user's responsability to show that run-time errors cannot occur. Our denotational model clarifies several points about the clock typing and Kahn semantics of the function reset operator. We are currently formalizing an alternative model for the function reset operator to further improve our understanding of this topic. We have started drafting an article on these results.

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.

6.2 Latency-based scheduling of synchronous programs

Participants: Timothy Bourke, 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 presented our previous results at ECRTS 2023 [16]. We also looked more closely at the possibilty of eliminating inter-period instantaneous cycles by adding constraints to the ILP scheduling problem. This problem is related to the detection of feedback arc sets for which there are two well-known encodings. Unfortunately, they can both induce a very large number of additional variables and constraints in the ILP encoding. This is not surprising since the base problem is NP-hard. We thus worked on mitigating heuristics. On the positive side, it turns out that our existing data-dependency and end-to-end latency constraints are readily generalized to allow for "variable concomitance" which may sometimes be useful for breaking instantanous cycles. In particular, we can require that the end-to-end latency along a cycle of data dependencies be strictly greater than zero. The ILP solver is then free to break dependencies by either scheduling the components in different phases or choosing concomitance values to prevent cycles during microscheduling. We presented these preliminary results at the Synchron 2023 workshop. In the collaboration with Airbus we extended the prototype compiler with a *hyper-period expansion* pass to permit an integration with the Lopht compiler. In the collaboration with Airbus we extended the prototype compiler with a *hyper-period expansion* pass to permit an integration with the Lopht compiler.

This work is funded by direct industrial contracts with Airbus.

6.3 The Zelus Language

Participants: Timothy Bourke, Guillaume Baudart, Marc Pouzet, Gregoire Bussone.

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 Zélus 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 well now. It is detailled in the Section below. Based on it, we have started a new implementation of Zélus with the objective that every pass of the compiler can be tested, using *Zrun* as an oracle.

6.4 A Constructive Synchronous Semantics

Participants: Baptiste Pauget, Marc Pouzet.

External collaborators: Jean-Louis Colaco (ANSYS, Toulouse, France); Michael Mendler (Univ. of Bamberg, Germany).

In 2023, we have finished an experiment that stated right after the COVID, during the preparation of a Master course given at University of Bamberg, in July 2020 (M. Pouzet, as an invited Professor). This work has been presented at the EMSOFT conference this year, in September 2023 and at the SYNCHRON Workshop, in November 2023, in Kiel (Gernany). It is published in the ACM TECS journal.

The purpose of this work is the definition of a formal and executable semantics for a reactive language that can be used as an oracle for compiler testing and the formal verification of compiler steps. We have considered a comprehensive synchronous language with programming constructs that exist in several compilers (developed at PARKAS and elsewhere): its core is a language subset reminiscent of Lustre, including the definition of stream functions and streams defined by mutually recursive defitions, the point-wise application of combinational operations, the delay operator. It is extended with constructs that do not exist in Lustre, like the by-case definitions of streams, hierarchical automata and the modular reset. Those construct a part of Vélus, Scade and Zélus (developed at PARKAS), for example, and LustreC (developed at ENSEEIHT, Toulouse). Two main approaches have been considered for defining the semantics of a language with such constructs in the litterature: (i) an indirect collapsing semantics based on a source-to-source translation of high-level constructs into a data-flow core language whose semantics is precisely specified and is the entry for code generation; (ii) a relational synchronous semantics, either state-based or stream-based, that applies directly to the source. It defines what is a valid synchronous reaction but hides, on purpose, if a semantics exists, is unique and can be computed. Hence, it is not executable and can thus not be used for compiler testing.

In this work, we define an executable semantics for a language that has all the above programming constructs all together. It applies directly to the source language before static checks and compilation steps. It is constructive in the sense that the language in which the semantics is defined is a statically typed functional language with call-by-value and strong normalization, e.g., it is expressible in a proof-assistant where all functions terminate. It leads to a reference, purely functional, interpreter. This semantics is modular and can account for possible errors, allowing to establish what property is ensured by each static verification performed by the compiler. It also clarifies how causality is treated in Scade compared with Esterel.

This semantics can serve as an oracle for compiler testing and validation; to prototype novel language constructs before they are implemented, to execute possibly unfinished models or that are correct but rejected by the compiler; to prove the correctness of compilation steps.

In term of expressiveness, we went a little further with the treatment of array operators and two forms of iterations, the iteration on time, named `forward` and the iteration in space, named `foreach`. The former has been studied by B. Pauget in his PhD. thesis: it constists in iterating a stream function on an array, interpreted as a finite stream and is reminiscent of "time refinement" (Caspi et Mikac, 2005; Mandel, Pouzet and Pasteur, 2015). We also added static parameters and a limited form of higher-order (functions of functions but no streams of functions). Those extensions are part of the source code distribution.

Our long term objective is to define an executable semantics for the Zélus language, dealing with both discrete and continuous-time constructs. For the moment, only the discrete-time subset of Zélus language is considered. Treating the whole language would lead to the very first operational semantics for a hybrid systems modeling language.

The semantics is implemented as an interpreter in a purely functional style, in OCaml. The source code of this development is available at the Zrun repository.

6.5 Design, Semantics and Implementation of a Memory Aware Synchronous data-flow language for Computer Intensive Reactive Applications

Participants: Baptiste Pauget, Marc Pouzet, Grégoire Bussone.

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

In his PhD., Baptiste Pauget studied the design, semantics and compilation of a reactive language extended with array operators. Those operations are used in classical control systems applications (e.g., Kalman filtering, linear algebra operations) and more recent ones that involves optimization algorithms and machine learning algorithems (e.g., neural networks). Existing languages, e.g., Lustre and Scade but more widely all the existing block-diagram languages used for model-based design, e.g, Simulink, are too limited in term of expressiveness and modularity. But more problematically, the generated code is not as efficient as it should be. The consequence is that designer may have to model its system in one language (e.g., Simulink, Scade) and to re-implement it into C code. One difficulty is that the generated code contains many useless copies for arrays that are difficult to remove. This problem exist in all purely functional language: how to generate code for functional arrays with in-place modifications and a compile-time static allocation of memory.

Baptiste Pauget has addressed three aspects, with the support of a compiler prototype. (i) He developed a Hindley-Milner type system specifying sizes in the form of multivariate polynomials. This proposal makes it possible to verify and infer most sizes in a modular way. (ii) He explored an alternative compilation method, based on a memory-aware declarative language named MADL. It aims to reconcile the data flow style with precise specification of memory locations. The modular size description is a key element of this. In this language, copies must be explicit. Several programming constructs (e.g., concat, append, reverse, transpose, etc.) do not generate any code. They define a special view of a memory location. MADL comes with a original type system that associate a location to every expression. Type checking ensure that programs can be statically scheduled. (iii) Finally, he proposed an iteration construction inspired by Sisal which complements current iterators. By treating tables as finite sequences, it gives access to Scade's sequential constructions (automata) during iterations. In addition, it makes it possible to describe in a declarative manner efficient implementations of algorithms such as the Cholesky decomposition. This controllable compilation is a necessary first step for compiling to GPUs.

In his PhD. thesis started in 2023, Grégoire Bussone pursues this work on the design, semantics and implentation of a synchronous language, dealing with aggressive optimization techniques.

6.6 Polymorphic Types with Polynomial Sizes

Participants: Baptiste Pauget, Marc Pouzet.

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

In this work, we present a compile-time analysis for tracking the size of data-structures in a statically typed and strict functional language. This information is valuable for static checking and code generation. Rather than relying on dependent types, we propose a type-system close to that of ML: polymorphism is used to define functions that are generic in types and sizes; both can be inferred. This approach is convenient, in particular for a language used to program critical embedded systems, where sizes are

indeed known at compile-time. By using sizes that are multivariate polynomials, we obtain a good compromise between the expressiveness of the size language and its properties (verification, inference).

We define a minimal functional language that is sufficient to capture size constraints in types, present its dynamic semantics, the type system and inference algorithm. Last, we sketch some practical extensions that matter for a more realistic language.

This work has been presented at the conference ARRAY (associated with PLDI) in June 2023, the international workshop SYNCHRON, in December 2023; at a seminar of the GDR GPL (group compilation). It is published by ACM (ARRAY'23). This work is part of the PhD. thesis of B. Pauget defended in December 2023 [29].

6.7 Translation Validation Techniques for a Synchronous Language Compilers

Participants: Timoty Bourke, Grégoire Bussone, Marc Pouzet.

Grégoire Bussone stated his PhD. in April 2023. He studies the use of translation validation techniques applied to a realistic synchronous language compiler. The objective is to deal with the compilation of array operations and, more generally, memory location. Arrays are not supported in Vélus for the moment. The problem is difficult and occurs in two situations: avoid copies for functional iterators (e.g., map, fold, transpose, concat, reverse); optimize the representation of the state in the final target code (e.g., C) and avoid useless copies for states whose lifetime never intersect (a classical situation that comes for a Scade-like hierarchical automaton where all states are entered by reset). For this work, we follow a translation validation approach, relying on an untrusted compiler and an independent but trustable validation step. We also target a richer and type-safe language back-end (here Rust) instead of C to transmit some of the invariants from the source. In the longer term, the purpose is to be able to implement and to machine-check the correctness of compilation techniques for a synchronous language with arrays and their efficient compilation.

During year 2023, several compilation steps that are implemented in the Zélus compiler have been implemented as translation validation functions proved correct in Coq, notably the inlining, renaming, scheduling, normalization. Internally, the technique employs the "locally nameless representation" introduced by Charguéraud. The input language is, for the moment, a simple subset of Zélus. The treatment of MADL is under way.

6.8 Reactive Probabilistic Programming

Participants: Guillaume Baudart, Marc Pouzet, Grégoire Bussone.

External collaborators: Louis Mandel (IBM), Erik Atkinson, Michael Carbin and Ellie Y. Cheng (MIT), Waïss Azizian, Marc Lelarge (Inria), Christine Tasson (ISAE-Supaero).

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) new semantics models 2) static analysis for semi-symbolic inference, and 3) embedding ProbZelus ideas in Julia.

Schedule agnostic semantics for reactive probabilistic programming In ProbZelus, the semantics of probabilistic models is only defined for scheduled equations. This is a significant limitation compared to

synchronous dataflow languages where sets of mutually recursive equations are not ordered. This is a key requirement for commercial synchronous data-flow languages where programs are written using a block diagram graphical interface. Scheduling should not depend on the placement of the blocks which motivate their definition as mutually recursive equations. Besides, the compiler implements a series of source-to-source transformations which often introduces new variables in arbitrary order. Scheduling local declarations is one of the very last compilation passes. The original semantics of ProbZelus is thus far from what is exposed to the programmer and prevents reasoning about most program transformations and compilation passes.

Building on existing semantics for deterministic synchronous languages, we proposed two schedule agnostic semantics for ProbZelus. The key idea is to interpret probabilistic expressions as a stream of un-normalized density functions which maps random variable values to a result and positive score. The co-iterative semantics extends the original semantics to interpret mutually recursive equations using a fixpoint operator. The relational semantics directly manipulates streams and is thus a better fit to reason about program equivalence. We use the relational semantics to prove the correctness of a program transformation required to run the Assumed Parameter Filter (APF) an optimized inference algorithm for state-space models with constant parameters.

A preliminary version of this work is available online [24]. The work on the APF-based inference engine (static analysis, compilation, runtime) was presented By G. Bussone at the Journées Francophones des Langages Applicatifs (JFLA) 2023 [18].

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

Advanced probabilistic inference algorithms combine exact and approximate inference to improve performance in probabilistic programs, and often use various heuristics and optimizations. The inference engine tries to compute exact solution as much as possible and falls back to approximate sampling when symbolic computations fail. The dynamic nature of these systems comes at a cost: 1) The heuristics are not guaranteed to be globally optimal, and 2) inference behavior is unpredictable.

We propose a new probabilistic language with a semi-symbolic inference engine based on our previous work on Delayed Sampling and Semi-Symbolic inference. In this language the user can annotate the program with constraints on the random variable representation (e.g., sampled or symbolic). A specialized static analysis then checks at compile time if these constraints are satisfiable.

A short version of this work was presented at the VeriProP workshop at the International Conference on Computer Aided Verification (CAV) 2023 [23].

OnlineSampling.jl (with Marc Lelarge and Waïss Azizian).

We continued our work on OnlineSampling.jl. 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 work was accepted at the SPIGM workshop at the International Conference on Machine Learning (ICML) [20]

7 Bilateral contracts and grants with industry

7.1 Bilateral contracts with industry

Collaboration with Airbus

Participants: Timothy Bourke, Marc Pouzet.

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

8 Partnerships and cooperations

8.1 National initiatives

8.1.1 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 ended in December 2023.

9 Dissemination

9.1 Promoting scientific activities

9.1.1 Scientific events: organisation

- Timothy Bourke presided and organized, with D. Demange as vice-president, the Journées Francophones des Langages Applicatifs (JFLA) 2023.

General chair, scientific chair

- Guillaume Baudart was co-chair with B. Greenman of the Artifact Evaluation Committee of the OOPSLA 2023 conference.

Member of the conference program committees

- Guillaume Baudart served on the Languages for Inference workshop (LAFI) 2023 program committee.
- Guillaume Baudart served on the Languages, Compilers, Tools and Theory of Embedded Systems conference (LCTES) 2023 program committee.
- Guillaume Baudart served on the Forum on specification and Design Languages (FDL) 2023 program committee.
- Timothy Bourke served on the International Conference on Embedded Software (EMSOFT) 2023 program committee.
- Timothy Bourke served on the Euromicro Conference on Real-Time Systems (ECRTS) 2023 program committee.
- Timothy Bourke served on the Workshop on Reactive and Event-Based Languages and Systems (REBLS) 2023 program committee.

Reviewer

- Timothy Bourke reviewed an article for the International Conference on Interactive Theorem Proving (ITP) 2023.
- Timothy Bourke reviewed an article for the International Conference on the Principles of Programming Languages (POPL) 2024.

Reviewer - reviewing activities

- Guillaume Baudart reviewed an article for the ACM Transactions on Software Engineering and Methodology.
- Timothy Bourke reviewed an article for the Leibniz Transactions on Embedded Systems.
- Timothy Bourke reviewed articles for the ACM Transactions on Embedded Computing Systems.

9.1.2 Scientific expertise

- Timothy Bourke was an expert reviewer for the ANR AAPG 2023 call.

9.2 Teaching - Supervision - Juries

9.2.1 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

9.2.2 Supervision

- Timothy Bourke supervised the L3 Internship of Vrushank Agrawal on “Model-Based Engineering of Quadcopter Control Software”.
- Timothy Bourke supervised the M2 (MPRI) Internship of Paul Robert (MPRI)on “"Delay Sensitive Static Scheduling of Periodic Synchronous Systems".
- Marc Pouzet supervised the M2 (MPRI)Internship of Antoine Grimod on "A clock-calculus for Zélus".

9.2.3 Juries

- Timothy Bourke was examiner for PhD defense of Jayanth Krishnamurthy (Université Côté d'Azur)
- Timothy Bourke was examiner for PhD defense of Nicolas Nalpon (INSA Toulouse)
- Timothy Bourke was examiner for PhD defense of Baptiste Pollien (ISAE SUPAERO)
- Timothy Bourke was examiner for PhD defense of Fabien Siron (Université Côté d'Azur)
- Marc Pouzet was reviewer of the HDR of Claire Maiza (Université Grenoble-Alpes)
- Marc Pouzet was reviewer of the HDR of Julien Forget (Université de Lille)
- Marc Pouzet was examiner for the HDR of Jérôme Feret (Ecole normale supérieure and PSL University)
- Marc Pouzet was examiner for PhD defense of Léo Gourdin (Université Grenble-Alpe). Dir: Sylvain Boulmé et David Monniaux.
- Marc Pouzet was examiner for the PhD of Amaury Maille, Ecole normale supérieure de Lyon. Dir: Ludovic Henrio and Matthieu Moy.
- Marc Pouzet was examiner for the PhD of Albin Salazar. Ecole normale supérieure and PSL University. Dir: Jérôme Feret.

9.2.4 PhD. Defenses at PARKAS

- Basile Pesin defended his PhD. the 13th of October, 2023 (Jury: Florence Maraninchi (President); Magnus Myreen, Robert de Simone (Reviewers); Carlos Agon (IRCAM and Sorbonne Université; Julien Forget, Xavier Leroy (Examiner).
- Baptiste Pauget defended his PhD. the 8th of December, 2023 (Jury: Jean-Louis Giavitto (President); Albert Cohen, Francois Pottier (Reviewers); Yamine Ait-Ameur, Laure Gonnor (Examiner); Jean-Louis Colaco, Marc Pouzet (director)

9.2.5 Internal or external Inria responsibilities

- Timothy Bourke served on the Jury for the Inria PARIS CRCN/ISFP Concours.

9.2.6 Education

- Timothy Bourke gave a "Chiche" presentation to the groups at the Lycée Montaigne, Paris 6e.

10 Scientific production

10.1 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] T. Bourke, B. Pesin and M. Pouzet. 'Verified Compilation of Synchronous Dataflow with State Machines'. In: *ACM Transactions on Embedded Computing Systems*. EMSOFT 2023: 23rd International Conference on Embedded Software. Vol. 22. 5s. Hamburg, Germany, 30th Sept. 2023, 137:1–137:26. DOI: 10.1145/3608102. URL: https://inria.hal.science/hal-04201401.

[6] 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.

[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.

10.2 Publications of the year

International journals

[12] J.-L. Colaço, M. Mendler, B. Pauget and M. Pouzet. 'A Constructive State-based Semantics and Interpreter for a Synchronous Data-flow Language with State Machines: Application to the Language Scade'. In: *ACM Transactions on Embedded Computing Systems (TECS)* 22.5s (9th Sept. 2023), Article 152: 1–26. DOI: 10.1145/3609131. URL: https://hal.science/hal-04491219.

[13] I. Rak-amnouykit, A. Milanova, G. Baudart, M. Hirzel and J. Dolby. 'Principled and practical static analysis for Python: Weakest precondition inference of hyperparameter constraints'. In: *Software: Practice and Experience* 54.3 (2024), pp. 363–393. DOI: 10.1002/spe.3279. URL: https://hal.science/hal-04489590.

[14] S. Varoumas, B. Pesin, B. Vaugon and E. Chailloux. 'Programming microcontrollers through high-level abstractions: The OMicroB project'. In: *Journal of Computer Languages* 77 (Nov. 2023), p. 101228. DOI: 10.1016/j.cola.2023.101228. URL: https://hal.sorbonne-universite.fr/hal-04279767.

Invited conferences

[15] G. Baudart and C. Tasson. 'Programmation réactive probabiliste'. In: 35es Journées Francophones des Langages Applicatifs (JFLA 2024). Saint-Jacut-de-la-Mer, France, 30th Jan. 2024. URL: https://inria.hal.science/hal-04407154.

International peer-reviewed conferences

[16] T. Bourke, V. Bregeon and M. Pouzet. 'Scheduling and Compiling Rate-Synchronous Programs with End-To-End Latency Constraints'. In: *Leibniz International Proceedings in Informatics*. 35th Euromicro Conference on Real-Time Systems (ECRTS 2023). Vol. 262. 35th Euromicro Conference on Real-Time Systems (ECRTS 2023). Vienna, Austria, 3rd July 2023, 1:1–1:22. DOI: 10.4230/LIPIcs.ECRTS.2023.1. URL: https://inria.hal.science/hal-04149828.

[17] T. Bourke, B. Pesin and M. Pouzet. 'Verified Compilation of Synchronous Dataflow with State Machines'. In: *ACM Transactions on Embedded Computing Systems*. EMSOFT 2023: 23rd International Conference on Embedded Software. Vol. 22. 5s. Hamburg, Germany, 30th Sept. 2023, 137:1–137:26. DOI: 10.1145/3608102. URL: https://inria.hal.science/hal-04201401.

National peer-reviewed Conferences

[18] G. Baudart, G. Bussone, L. Mandel and C. Tasson. 'Filtrer sans s'appauvrir : inférer les paramètres constants des modèles réactifs probabilistes'. In: *Journées Francophones des Langages Applicatifs*. JFLA 2023 - 34èmes Journées Francophones des Langages Applicatifs. Praz-sur-Arly, France, 16th Jan. 2023, pp. 24–42. URL: https://inria.hal.science/hal-03936566.

[19] T. Bourke, B. Pesin and M. Pouzet. 'Analyse de dépendance vérifiée pour un langage synchrone à flot de données'. In: *Journées Francophones des Langages Applicatifs*. JFLA 2023 - 34èmes Journées Francophones des Langages Applicatifs. Praz-sur-Arly, France, 16th Jan. 2023, pp. 101–120. URL: https://inria.hal.science/hal-03936656.

Conferences without proceedings

[20] W. Azizian, G. Baudart and M. Lelarge. 'Automatic Rao-Blackwellization for Sequential Monte Carlo with Belief Propagation'. In: SPIGM@ICML. Honolulu, United States, 28th July 2023. URL: https://hal.science/hal-04488225.

[21] J.-L. Colaço, B. Pauget and M. Pouzet. 'Polymorphic Types with Polynomial Sizes'. In: *Proceedings 9th ACM SIGPLAN International Workshop on Libraries, Languages and Compilers for Array Programming (ARRAY 2023)*. 9th ACM SIGPLAN International Workshop on Libraries, Languages and Compilers for Array Programming (ARRAY 2023). Orlando, United States: ACM, 6th June 2023, pp. 36–49. DOI: 10.1145/3589246.3595372. URL: https://hal.science/hal-04491216.

Edition (books, proceedings, special issue of a journal)

[22] T. Bourke and D. Demange, eds. *JFLA 2023 - 34èmes Journées Francophones des Langages Applicatifs*. JFLA 2023 - 34èmes Journées Francophones des Langages Applicatifs. Journées Francophones des Langages Applicatifs. 31st Jan. 2023, pp. 1–308. URL: https://inria.hal.science/hal-03962188.

Reports & preprints

[23] E. Atkinson, E. Y. Cheng, G. Baudart, L. Mandel and M. Carbin. *Verifying Performance Properties of Probabilistic Inference*. 14th July 2023. URL: https://hal.science/hal-04488233.

[24] G. Baudart, L. Mandel and C. Tasson. *Density-Based Semantics for Reactive Probabilistic Programming*. 7th Sept. 2023. URL: https://hal.science/hal-04488216.

10.3 Cited publications

[25] 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.

[26] T. Bourke, P. Jeanmaire, B. Pesin and M. Pouzet. 'Verified normalization of the Lustre language'. In: *JFLA 2021 - 32ème Journées Francophones des Langages Applicatifs*. Yann Régis-Gianas et Chantal Keller. En ligne, France, Apr. 2021, pp. 117–133. URL: https://inria.hal.science/hal-03287572.

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

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

[29] B. Pauget. 'Memory Specification in a Data-flow Synchronous Language with Statically Sized Arrays'. PhD thesis. Paris, France: PSL Université, Dec. 2023.

[30] 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.

[31] B. Pesin. 'Verified Compilation of a Synchronous Dataflow Language with State Machines'. PhD thesis. PSL Research University, Oct. 2023. URL: https://velus.inria.fr/phd-pesin/thesis.pdf.