RESEARCH CENTRE

Paris

IN PARTNERSHIP WITH:

CNRS, Ecole normale supérieure de Paris

2021 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, Starting Faculty Position]
- Timothy Bourke [Inria, Researcher]

Faculty Members

- Marc Pouzet [Team leader, Sorbonne Université, Professor, HDR]
- Paul Feautrier [Université de Lyon, Emeritus]

PhD Students

- Paul Jeanmaire [Inria]
- Ismail Lahkim Bennani [Inria, until May 2021]
- Astyax Nourel [Inria, from Apr 2021 until Sep 2021]
- Baptiste Pauget [ANSYS, CIFRE]
- Basile Pesin [Inria]

Interns and Apprentices

- Antonin Reitz [Inria, from Apr 2021 until Aug 2021]
- Reyyan Tekin [Inria, from Mar 2021 until Aug 2021]

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

The PARKAS team organized the 28th International Open Workshop on Synchronous Programming (SYNCHRON 2021).

5.1 Awards

Timothy Bourke, Basile Pesin, Paul Jeanmaire, and Marc Pouzet received the *best paper award* for “Verified Lustre Normalization with Node Subsampling” [13] at the ACM SIGBED International Conference on Embedded Software (EMSOFT) in October 2021.

6 New software and platforms

Software developed in the PARKAS team.

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 [32] Interactive Theorem Prover (ITP). It integrates the CompCert C compiler [37, 23] 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 three main directions: finalizing and presenting the *normalization* pass, progressively supporting higher-level constructions, and developing increasingly abstract models to facilitate interactive verification.

Normalizing Lustre: We continued the work from last year on translating Lustre programs into the normalized form required by existing compilation passes. Proofs about normalized Lustre programs proceed by induction on a list of equations ordered so that variables are defined before being used. This is not possible in the unrestricted form so we developed a new approach based on induction over an acyclic dependency graph. As part of this work, and to better prepare for future compilation passes, we reimplemented the generation of identifiers and simplified the axioms that specify the underlying OCaml routines that are shared with the CompCert verified C compiler. We wrote and submitted an article on this work to the EMSOFT conference [13]. The article was presented at a virtual event in October and received the best paper award.

Adding higher-level constructs: Our next major goal in terms of verified Lustre compilation is to specify the semantics of hierarchical state machines and implement verified algorithms to compile them into compositions of simpler constructions. This is the Basile Pesins thesis topic. Over the last year, we made solid progress toward this goal.

The syntax and semantics of the Vélus compiler was updated with the notion of a "block" to group sets of equations. We added passes to flatten the block structure and proved semantic preservation for these passes. We then generalized the `reset` operator to blocks. This was surprisingly difficult. It was necessary to adapt the existing treatment for node instances [24], to study existing formalisations and implementations of clock typing, and to treat `fby`-equations within a block. Next, we augmented blocks with local variable declarations. This involved updating the semantic model, the typing definitions, implementing a compilation pass to rename and lift local variables, and extending the proofs of semantics preservation. During this work we encountered many problems with the "anonymous variables" [25] introduced to treat node subsampling. We thus replaced this mechanism by a simpler solution.

State machines are compiled to `switch` statements over an enumerated type that encodes possible states [31]. We added this construct to the syntax of V?lus and extended the semantic model. The challenge was to adapt the existing relational predicates to define an overall behavior on a set of streams as the conjunction of constraints, one for each branch, over complementary intervals of those streams. It was necessary to impose a global constraint for intervals when the switch statement is inactive (i.e., when the streams it defines must be absent). The resulting model seemed reasonable but we were worried that it would be difficult to treat `last` variables [31], since they require that branch definitions interact over time. It turns out that this problem can be solved by using a distinct environment and applying `last` definitions from the top down. The compilation algorithms for the `switch` and `last` features have been verified, and the various proofs about typing, clock alignment, and determinism have been restored.

We have defined a subset of state machine features to treat over the next few months and we have made progress on the semantic model.

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.

We began the year by examining the related literature [40, 28, 26] and performing practical experiments on simple program fragments. We looked at different techniques to improve the readability of program terms in the proof assistant and experimented with dependent types for expressing a programs' type and clock constraints. For programs with subsampling, there are two standard approaches for treating filtered values: add explicit absent markers or reason on unsynchronized streams *à la Kahn* [36]. Our existing semantic models use explicit absent markers but prior work [26] suggests that a Kahn-style model permits simpler and more readable proofs. We thus started to develop a Kahn-style semantics in Coq using C. Paulin-Mohring's library [41]. The goal is to reason by rewriting on stream equations in a denotational model and to link this model with the one used in the compiler specifications. We have made solid progress in understanding the library and using it to express and reason about a subset of Lustre operators. Current work is focused on relating this model to the existing one.

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.

Harmonic clocks: We studied the extension of a synchronous language with periodic harmonic clocks based on the work of Mandel et al. [29, 42, 30, 38, 39] on n-synchrony and the extension proposed by Forget et al. [35]

Mandel et al. considered a language with periodic clocks expressed as ultimately periodic binary sequences. The decision procedures (equality, inclusion, precedence) for such an expressive language can be very costly. It is thus sometimes useful to apply an envelope-based abstraction, that is, one where sets of clocks are represented by a rational slope and an interval. Forget considered simpler "harmonic" clocks. His decision procedures conincide with those for the envelope-based abstraction but without any loss of information. During his M2 internship, B. Pauget continued this line of work by extending the input language of the Vélus Lustre compiler with harmonic clocks. This work was the starting point for the proposal of a new intermediate language for a synchronous compiler that is capable of exploiting clock information to apply agressive optimizations and generate parallel code.

New Intermediate Language MObc (Multi Object Code): This intermediate language is reminiscent of the intermediate Obc language used in the Vélus and Heptagon compiler, but with some important differences and new features. MObc permits a synchronous function to be represented as a set of named state variables and possibly nested blocks with a partial ordering which express the way blocks can and must be called. In comparison, Obc represents a synchronous function as a set of state variables and a transition function that is itself written in a sequential language. Each block comprises a set of equations in Single Static Assignment (SSA) form, that is, exactly one equation per variable, so as to simplify the implementation of a number of classic optimizations (for example, constant propagation, inlining, common sub-expression elimination, code specialisation). Then, every block is translated into a step function (e.g., a C function). This intermediate language has been designed to facilitate the generation of code for a real-time OS and a multi-core target. This work exploits two older results: the article of Caspi et al. [27] that introduces an object representation for synchronous nodes and a "scheduling policy" that specifies how their methods may be called, and; the work of Pouzet et al. [43] on the calculation of input/output relations to merge calculations. We are preparing and article on this subject.

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 a new prototype compiler for a synchronous language with clock rates and a model where "synchronous" does not necessarily mean "simultaneous". This year, we made progress on constraining latency across chains of components, treating cycles in the dataflow graph, and generating imperative code.

Embedded controllers often contain calculation sequences whose end-to-end latency is critical to application performance. This is notably the case for control calculations that occur between input acquisition and output emission. We devised and implemented an algorithm for expressing and constraining this latency as part of the overall scheduling problem. The difficulty is to treat the causality between and within instants, and to properly handle rate changes across sequences. This algorithm has been implemented in our prototype compiler. We also wrote a visualization tool for validating and explaining the algorithm. This tool helped us to find many bugs in the original implementation. It also appears to be a useful means of analyzing application behavior. We also adapted our prototype with support for atomic sequences.

Our algorithm worked well on a mid-sized case study, but upon attempting to apply it to a production problem, we had many problems with cyclic dependencies. In an attempt to visualize the problem we contributed algorithms to the ocamlgraph library and developed some simple tools (undotter and subdotter). We then surveyed the literature on heuristics for removing cycles by calculating *feedback arc sets*. Good heuristics are essential since the problem is NP-complete and our graphs are quite large. We adapted the "FASH" algorithm [34, 33] by extending it with weights and irreversible arcs and implemented it in OCaml. This approach seems to work quite well, but we are still experimenting with it.

We extended our compiler by implementing standard techniques for constraining and load-balancing resources. We extended the standard code generation scheme [22] to treat rate-based components.

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 v5.x and v6.x of the Sundials Suite of numerical solvers. This involved a significant reworking of the interface to the ARKode MRIStep solver. A major rewriting of the interface to nonlinear solvers, notably to permit callbacks from OCaml, through a C solver, and back to a custom OCaml solver. This rewriting uncovered a subtle inter-heap cycle that caused a difficult-to-diagnose memory leak and resulting performance problems. We added support for "many vectors" which provide arrays of heterogeneous vector types. The challenge here was to provide such a heterogeneous collection in OCaml. As usual, much time was spent on adding and updating example programs, fixing bugs uncovered in the process, and also analyzing and improving overall performance. 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 essentially a synchronous language reminiscent of Lustre and Lucid Synchrone but with the ability to define functions that manipulate continuous-time signals defined by Ordinary Differential Equations (ODEs). 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 now on GitHub. It is also available as an OPAM package. All the installation machinery has been greatly simplified.

Set-based simulation of Zelus programs In collaboration with Francois Bidet (PhD. student under the supervision of Sylvie Putot and Eric Goubault from Ecole polytechnique), we are developing a method to perform set-based simulation of Zelus program. Set-based simulation goes beyond concrete simulation (the default simulation mode of all existing hybrid system modeling languages). Instead of computing one trajectory, it computes a set of trajectories or *flowpipes* at once, replacing a possibly unbounded number of concrete simulations. It is also able to deal with models with partially known parameters and inputs.

Very little tools currently deal with models expressed modularily (as the parallel and hierarchical composition of subsystems, with function application and the mix between a software model and ODEs, for example). A prototype is under way. Set based simulation is done on the intermediate language generated by Zelus, that is a collection of tarnsition functions acting on a state.

Property Based Testing of Hybrid Programs Property-based program testing involves checking an executable specification by running many tests. We build on the work of Georgios Fainekos and Alexandre Donzé, and take inspiration from earlier work by Nicolas Halbwachs, to write a Zélus library of synchronous observers with a quantitative semantics that can be used to specify properties of a system under test. We implemented several optimization algorithms for producing test cases, some of which are gradient-based. This year, we have studied the use SUNDIALS CVODEs (sensitivity analysis) to find more falsification examples and faster.

7.5 An executable reference semantics for Zelus

During year 2021, we have worked on the definition of a comprehensive semantics for Zelus language, including all language constructs, that is executable and can lead to a reference interpreter.

The scientific objective is to use it to test an existing compiler, to prove the correctness of compile-time checks (e.g., that a well typed/causal/initialized program does not lead to an error); to prove the semantics preservation of compiler transformations (e.g., static scheduling, compilation of automata); to execute unfinished programs or programs that are semantically correct but are statically rejected by the compiler. Examples are cyclic circuits accepted by an Esterel compiler (the so-called "constructively causal" programs) but are rejected by Lustre, Lucid Synchrone, Scade, Zelus compilers that impose stronger causality constraints; finally to prototype new language constructs.

The existing semantics for rich languages like Scade is defined by its translation into a small data-flow language; we expect instead to have a semantics that apply directly to the source, before any rewriting or check is made.

The current prototype we have developed only deal with the synchronous subset only. It builds on the works 1/ “A Coiterative Characterization of Synchronous Stream Functions”, by Caspi and Pouzet, CMCS, 1998 (VERIMAG tech. report, 1997) and 2/ “The semantics and execution of a synchronous block-diagram language”, by Edwards and Lee, Science of Computer Programming 2006.

7.6 Array Size Checking and Inference with an ML Type System

Participants: Baptiste Pauget, Marc Pouzet.

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

We are interested here in the programming, with a high-level language, of real-time embedded applications that are submitted to strong safety requirements, such as those found in avionics, railway and automotive (eg, flight control, braking, electrical engine). Modern real-time applications combine complex control code, with a high level of nesting of hierarchical automata, and intensive computations using arrays. This work focuses on the latter aspect. We seek to express array computations within the framework of a purely functional language such as Scade, by offering a sufficient expressiveness for

typical applications, and whose safety can be ensured at compile-time by relatively inexpensive and modular means.

During year 2001, we have worked on a compile-time analysis for checking and inferring the size of arrays in a statically typed and strict functional language. 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 type and size. Inference allows a lighter writing of the classical signal processing operations — point-to-point application, accumulation, projection, transposition, convolution, a restricted form of recursion over sizes — and their composition. The automatic inference of types is a key feature of the proposed solution. To obtain a good compromise between the expressiveness of the type language, the decidability of the verification and automatic inference, the solution relies on two elements: (i) a language of types where sizes in types are multivariate polynomials; (ii) the possible insertion of explicit coercions between sizes in the source program. When the program is well-typed, it executes without any size errors outside of these coercion points. Two uses of the proposed solution can be considered: (i) the generation of defensive code at coercion points or, (ii) their static verification by restricting them to be expressions that can be evaluated at compile-time — a frequent situation in safety critical applications — or by other formal verification means for the remaining cases.

The article defines a core functional language that is sufficient to express array operations and to capture size constraints in types; in particular, arrays are simply functions on a finite domain. The article presents the dynamic semantics of the language, the type system and inference algorithm, and its correctness. Then, it presents a surface language for the programmer, with the classical notations for arrays, that elaborates to the core language. All the presented material is supported by an implementation in OCaml, whose source code is available.

A prelinary work, written in French, is accepted for publication at *Journés Francophones des langages applicatifs (JFLA)*, June 2022.

7.7 Probabilistic Programming

Participants: Guillaume Baudart, Marc Pouzet, Reyyan Tekin.

7.7.1 Reactive Probabilistic Programming

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.

We previously designed ProbZelus, an extension of Zelus with probabilistic constructs to model uncertainties and perform inference-in-the-loop. Importanty, we introduced a novel streaming delayed sampling implementation which enables partial exact inference over infinite streams in bounded memory for a large class of models.

Continuing the collaboration with Louis Mandel (IBM), Erik Atkinson, Michael Carbin and Charles Yuan (MIT), we found conditions on a reactive probabilistic model's execution under which delayed sampling will execute in bounded memory. The two conditions are dataflow properties of the core operations of delayed sampling: the m-consumed property and the unseparated paths property. A showed that a program executes in bounded memory under delayed sampling if, and only if, it satises the m-consumed and unseparated paths properties. We proposed a static analysis that abstracts over these properties to soundly ensure that any program that passes the analysis satises these properties, and thus executes in bounded memory under delayed sampling.

The main article *Statically Bounded-Memory Delayed Sampling for Probabilistic Streams* was presented at the Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2021) and published in the Proceedings of the ACM on Programming Languages (PACMPL) [12]

. A short version was also presented at the international conference on Probabilistic Programming (PROGPROB 2021) [19].

7.7.2 Compiling Stan to Generative Probabilistic Languages

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 Church,Anglican, or Pyro. We proposed a comprehensive compilation scheme to compile any Stan model to a generative language and proved its correctness. We use our compilation scheme to build two new backends for the Stanc3 compiler targeting Pyro and NumPyro. Experimental results show that the NumPyro backend yields significant speedup compared to Stan on existing benchmarks.

Our compiler leverages the rich set of Pyro and Numpyro features for Stan users. Building on Pyro we thus extended Stan with support for explicit variational inference guides and deep probabilistic models, i.e., probabilistic models involving neural networks. Leveraging NumPyro runtime we show that using our recently proposed compiler from Stan to Pyro, Stan users can easily try the set of algorithms implemented in Pyro for black-box variational inference.

The compiler is available on GitHub and the main article *Compiling Stan to Generative Probabilistic Languages and Extension to Deep Probabilistic Programming* was presented at the Conference on Programming Language Design and Implementation (PLDI 2021) [14] . The article *Automatic Guide Generation for Stan via NumPyro* was accepted as an oral presentation at the international conference on Probabilistic Programming (PROGPROB 2021) [19].

7.8 Automated Machine Learning

Participants: Guillaume Baudart.

7.8.1 Lale: Gradual Automation

Automated machine learning (AutoML) can make data scientists more productive. But if machine learning is totally automated, that leaves no room for data scientists to apply their intuition. Hence, data scientists often prefer not total but gradual automation, where they control certain choices and AutoML explores the rest. We thus proposed Lale, a sklearn-compatible AutoML library based on a small set of orthogonal combinators for composing machine- learning operators into pipelines. Lale then compiles pipelines and associated hyperparameter schemas to search spaces for AutoML optimizers.

Lale is gradual, letting users specify only what they want while reusing and automating the rest. For instance, Lale comes with an extensive library of reusable hyperparameter schemas for many popular operators, so users rarely need to write their own schemas; but it also makes it easy to customize schemas when desired. There are Lale optimizer backends for multiple optimizers: Hyperopt; sklearn's GridSearchCV and HalvingGridSearchCV; ADMM; SMAC; and Hyberband.

Lale is available on GitHub and the main article *Pipeline Combinators for Gradual AutoML* was presented at the Conference on Neural Information Processing Systems (NeurIPS'21) [15]

7.8.2 Extracting Hyperparameters Constraints from Code

In Lale, each machine learning operator is associated to a schema that captures its hyperparameters and correctness constraints that cut across multiple hyperparameters and/or data. Violating these constraints causes runtime exceptions, but they are usually documented only informally or not at all. We proposed an interprocedural weakest-precondition analysis for Python code 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.

The paper *Extracting Hyperparameter Constraints from Code* was presented at the ICLR 2021 Workshop on Security and Safety in Machine Learning Systems [21].

7.9 Application: Learning GraphQL Query Cost

GraphQL is a query language for APIs and a runtime for executing those queries. Its expressiveness and its flexibility have made it an attractive candidate for API providers in many industries, especially through the web. A major drawback to blindly servicing a client's query in GraphQL is that the cost of a query can be unexpectedly large. To mitigate these drawbacks, it is necessary to efficiently estimate the cost of a query before executing it. We proposed a machine-learning approach to efficiently and accurately estimate the query cost.

There are many well-known operators that implement regression algorithms and feature preprocessing. A library like scikit-learn implements many of these operators, but picking the right operators and configuring their hyperparameters is a tedious task and depends on the dataset. We thus used Lale to select the best operators and tune the hyperparameters given a query/response dataset. We demonstrated the power of this approach by testing it on query-response data from publicly available commercial APIs.

The paper *Learning GraphQL Query Cost* was presented as an industry showcase at the International Conference on Automated Software Engineering (ASE) [20]

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 International initiatives

9.1.1 Participation in other International Programs

MIT-IBM

Participants: Guillaume Baudart.

Title: Probabilistic Programming

Partner Institution(s):
- IBM, United States of America
- MIT, United States of America

Partners:
- Louis Mandel (IBM)
- Michael Carbin (MIT)
- Eric Atkinson (MIT)
- Charles Yuan (MIT)

Inria contact: Guillaume Baudart

Summary: Collaboration started when G. Baudart was at IBM Research. This project focuses on reactive probabilistic programming, in particular the development of ProbZelus and associated inference algorithms and static analyses.

RPI-IBM

Participants: Guillaume Baudart.

Title: Constraints from Machine Learning Code

Partner Institution(s):
- IBM, United States of America
- Rensselaer Polytechnic Institute, United States of America

Partners:
- Martin Hirzel (IBM)
- Julian Dolby (IBM)
- Ana Milanova (RPI)
- Ingkarat Rak-amnouykit (RPI)

Inria contact: Guillaume Baudart

Summary: Collaboration started when G. Baudart was at IBM Research. This project focuses on static analysis techniques to extract constraints from machine learning operator codes. These constraints can then be used by the Lale project for automated machine learning.

9.2 European initiatives

9.2.1 FP7 & H2020 projects

TETRAMAX (582)

Title: TEchnology TRAnsfer via Multinational Application eXperiments

Duration: 9/2017 - 12/2021

Coordinator: Rainer Leupers

Partners:

- AMG TECHNOLOGY OOD (Bulgaria)
- BUDAPESTI MUSZAKI ES GAZDASAGTUDOMANYI EGYETEM (Hungary)
- INSTITUT JOZEF STEFAN (Slovenia)
- RHEINISCH-WESTFAELISCHE TECHNISCHE HOCHSCHULE AACHEN (Germany)
- RUHR-UNIVERSITAET BOCHUM (Germany)
- SVEUCILISTE U ZAGREBU FAKULTET ELEKTROTEHNIKE I RACUNARSTVA (Croatia)
- TALLINNA TEHNIKAULIKOOL (Estonia)
- TECHNISCHE UNIVERSITAET MUENCHEN (Germany)
- TECHNISCHE UNIVERSITEIT DELFT (Netherlands)
- THE UNIVERSITY OF EDINBURGH (United Kingdom)
- THINK SILICON EREYNA KAI TECHNOLOGIA ANONYMI ETAIRIA (Greece)
- TTY-SAATIO (Finland)
- UNIVERSITA DI PISA (Italy)
- UNIVERSITAT POLITECNICA DE CATALUNYA (Spain)
- UNIVERSITEIT GENT (Belgium)
- VYSOKA SKOLA BANSKA - TECHNICKA UNIVERZITA OSTRAVA (Czech Republic)
- VYSOKE UCENI TECHNICKE V BRNE (Czech Republic)

- ZENIT ZENTRUM FUR INNOVATION UND TECHNIK IN NORDRHEIN-WESTFALEN GMBH (Germany)

Inria contact: Timothy Bourke

Summary: TETRAMAX, *Technology Transfer via Multinational Application Experiments*, is funded by the H2020 "Smart Anything Everywhere (SAE)" initiative. The overall ambition is to build and leverage a European Competence Center Network in customized low-energy computing, providing easy access for SMEs and mid-caps to novel CLEC technologies via local contact points. This is a bidirectional interaction: SMEs can demand CLEC technologies and solutions via the network, and vice versa academic research institutions can actively and effectively offer their new technologies to European industries. Furthermore, TETRAMAX wants to support 50+ industry clients and 3rd parties with innovative technologies, using different kinds of Technology Transfer Experiments (TTX) to accelerate innovation within European industries and to create a competitive advantage in the global economy.

MNEMOSENE

Title: Computation-in-memory architecture based on resistive devices

Duration: 1/2018 - 6/2021

Coordinator: Said Hamdioui

Partners:

- ARM LIMITED (UK)
- EIDGENOESSISCHE TECHNISCHE HOCHSCHULE ZUERICH (Switzerland)
- IBM RESEARCH GMBH (Switzerland)
- INTELLIGENTSIA CONSULTANTS SARL (Luxembourg)
- RHEINISCH-WESTFAELISCHE TECHNISCHE HOCHSCHULE AACHEN (Germany)
- STICHTING IMEC NEDERLAND (Netherlands)
- TECHNISCHE UNIVERSITEIT DELFT (Netherlands)
- TECHNISCHE UNIVERSITEIT EINDHOVEN (Netherlands)

Inria contact: Andi Drebes

Summary: MNEMOSENE aims at demonstrating a new computation-in-memory (CIM) computer architecture based on resistive devices, together with its required programming flow and interface. MNEMOSENE targets advanced explorative technology development at TRL 2 (technology concept formulation) and TRL3 (experimental proof-of-concept) and represents a first step towards the development of a fully operational CIM based computer, which MNEMOSENE consortium partners believe will require 9 to 12 years of further research after project completion.

9.3 National initiatives

9.3.1 ANR

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

9.3.2 FUI: Fonds unique interministériel

Modeliscale contract (AAP-24) 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).

9.3.3 Programme d'Investissements d'Avenir (PIA)

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).

9.3.4 Others

Inria Project Lab (IPL) Modeliscale 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

Member of the organizing committees

- Timothy Bourke and Marc Pouzet were coorganizers, with Thérèse Hardin, of the 28th International Open Workshop on Synchronous Programming (SYNCHRON 2021).
- Timothy Bourke was tutorial chair for Embedded Systems Week 2021 (ESWEEK 2021).

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 2021).
- Guillaume Baudart served on the program committee of the Industry Track of the ACM International Conference on Distributed and Event-Based System (DEBS 2021).
- Guillaume Baudart served on the program committee of the ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES 2021)
- Guillaume Baudart served on the program committee of the ACM/IEEE International conference on Embedded Software (EMSOFT 2021).
- Guillaume Baudart served on the program committee of the ACM SIGPLAN International Conference on Compiler Construction (CC 2021).
- Timothy Bourke served on the program committee of the Design Automation Conference (DAC 2021, track ESS1: Embedded Software).
- Timothy Bourke served on the program committee of the *Journées Francophones des Langages Applicatifs* (JFLA 2021).
- Timothy Bourke served on the program committee of the 24th International Workshop on Software and Compilers for Embedded Systems (SCOPES 2021).
- Timothy Bourke served on the program committee of the International Modelica Conference (MODELICA 2021).
- Marc Pouzet served on the program committee of the 24th International Workshop on Software and Compilers for Embedded Systems (SCOPES 2021).
- Marc Pouzet served on the program committee of the International Forum on specification & Design Languages (FDL 2021).

Reviewer

- Timothy Bourke reviewed articles for the *2021 Symposium on Principles of Programming Languages*.

10.1.3 Journal

Reviewer - reviewing activities

- Timothy Bourke reviewed articles for the *Journal of Logical and Algebraic Methods in Programming*.

10.1.4 Invited talks

- Guillaume Baudart was an invited speaker at the Workshop on Probabilistic Interactive and Higher-Order Computation (PIHOC 2021).
- Guillaume Baudart was an invited speaker at the Inria Paris, demi-heure de science seminar.
- Timothy Bourke gave the keynote talk at the Workshop on Reactive and Event-Based Languages and Systems (REBLS 2021).
- Marc Pouzet was invited speaker by the formal method group of NASA Langley, January 2021.

10.1.5 Research administration

- Timothy Bourke was a jury member for the Paris Centre CRCN/ISFP concours.
- Timothy Bourke participated in several thesis monitoring committees.
- Marc Pouzet was jury member of the Phd. thesis of Vincent Lampietro (Univl. Montpellier, December 2021.

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 COMASIC (École Polytechnique) and FIL (Université Paris-Sud, Saclay), France
- 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 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
- Internships Timothy Bourke & Guillaume Baudart participated in reviewing the L3 and M1 internships of students at the ENS, France.

10.2.2 Supervision

- PhD in progress: Paul Jeanmaire, 2nd year, supervised by Timothy Bourke and Marc Pouzet.
- PhD in progress: Baptiste Pauget, 2nd year, supervised by Marc Pouzet.
- PhD in progress: Basile Pesin, 2nd year, supervised by Timothy Bourke and Marc Pouzet.
- PhD in progress: Astyax Nourel, 1st year, supervised by Adrien Guatto (IRIF, Univ. of Paris) and Marc Pouzet.
- Master: Antonin Reitz, M2 research internship, supervised by Marc Pouzet (March-August 2021).
- Master: Reyan Tekin, M2 research internship, supervised by Guillaume Baudart and Marc Pouzet (March-August 2021).

10.3 Popularization

10.3.1 Education

- Timothy Bourke presented at a meeting of the IEEE Student Chapter at the École polytechnique.

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] ***Best Paper***
E. Atkinson, G. Baudart, L. Mandel, C. Yuan and M. Carbin. 'Statically bounded-memory delayed sampling for probabilistic streams'. In: *Proceedings of the ACM on Programming Languages* 5.OOPSLA (20th Oct. 2021), pp. 1–28. DOI: 10.1145/3485492. URL: https://hal.archives-ouvertes.fr/hal-03401752.

[13] T. Bourke, P. Jeanmaire, B. Pesin and M. Pouzet. 'Verified Lustre Normalization with Node Subsampling'. In: *ACM Transactions on Embedded Computing Systems (TECS)* 20.5s (1st Oct. 2021), pp. 1–25. DOI: 10.1145/3477041. URL: https://hal.inria.fr/hal-03370264.

International peer-reviewed conferences

[14] ***Best Paper***
G. Baudart, J. Burroni, M. Hirzel, L. Mandel and A. 2. Shinnar. 'Compiling Stan to generative probabilistic languages and extension to deep probabilistic programming'. In: PLDI '21 - 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation. Virtual, Canada: ACM, 18th June 2021, pp. 497–510. DOI: 10.1145/3453483.3454058. URL: https://hal.archives-ouvertes.fr/hal-03401742.

[15] G. Baudart, M. Hirzel, K. Kate, P. Ram, A. Shinnar and J. Tsay. 'Pipeline Combinators for Gradual AutoML'. In: NeurIPS 2021 - Thirty-fifth Conference on Neural Information Processing Systems. Virtual, France, 6th Dec. 2021. URL: https://hal.archives-ouvertes.fr/hal-03464012.

[16] L. Chelini, A. Drebes, O. Zinenko, A. Cohen, N. Vasilache, T. Grosser and H. Corporaal. 'Progressive Raising in Multi-level IR'. In: CGO 2021 : International Symposium on Code Generation and Optimization. International Conference on Code Generation and Optimization (CGO). Seoul / Virtual, South Korea, 27th Feb. 2021. URL: https://hal.inria.fr/hal-03139764.

National peer-reviewed Conferences

[17] T. Bourke, P. Jeanmaire, B. Pesin and M. Pouzet. 'Verified normalization of the Lustre language'. In: *JFLA 2021 - 32ème Journées Francophones des Langages Applicatifs*. JFLA 2021 - 32ème Journées Francophones des Langages Applicatifs. En ligne, France, 7th Apr. 2021, pp. 117–133. URL: https://hal.inria.fr/hal-03287572.

Conferences without proceedings

[18] E. Atkinson, G. Baudart, L. Mandel, C. Yuan and M. Carbin. 'Checking Bounded-Memory Execution for Delayed Sampling on Probabilistic Streams'. In: PROBPROG 2021 - Third International Conference on Probabilistic Programming. Virtual, United States, 20th Oct. 2021. URL: https://hal.archives-ouvertes.fr/hal-03401720.

[19] G. Baudart and L. Mandel. 'Automatic Guide Generation for Stan via NumPyro'. In: PROBPROG 2021 - Third International Conference on Probabilistic Programming. Virtual, United States, 20th Oct. 2021. URL: https://hal.archives-ouvertes.fr/hal-03401708.

[20] G. Mavroudeas, G. Baudart, A. Cha, M. Hirzel, J. A. Laredo, M. Magdon-Ismail, L. Mandel and E. Wittern. 'Learning GraphQL Query Cost'. In: ASE 2021 - IEEE/ACM International Conference on Automated Software Engineering – Industry Showcase. Melbourne / Virtuel, Australia, 14th Nov. 2021. URL: https://hal.archives-ouvertes.fr/hal-03469475.

[21] I. Rak-Amnouykit, A. Milanova, G. Baudart, M. Hirzel and J. Dolby. 'Extracting Hyperparameter Constraints from Code'. In: ICLR Workshop on Security and Safety in Machine Learning Systems. Virtual, United States, 7th May 2021. URL: https://hal.archives-ouvertes.fr/hal-03401683.

11.3 Cited publications

[22] D. Biernacki, J.-L. Colaço, G. Hamon and M. Pouzet. 'Clock-directed Modular Code Generation of Synchronous Data-flow Languages'. In: *ACM International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*. Tucson, Arizona, June 2008.

[23] 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.

[24] T. Bourke, L. Brun and M. Pouzet. 'Towards a verified Lustre compiler with modular reset'. In: *21st International Workshop on Software and Compilers for Embedded Systems (SCOPES 2018)*. Proceedings of the 21st International Workshop on Software and Compilers for Embedded Systems (SCOPES 2018). Sankt Goar, Germany: ACM Press, May 2018, p. 4. DOI: 10.1145/3207719.3207732. URL: https://hal.inria.fr/hal-01817949.

[25] T. Bourke and M. Pouzet. 'Clocked arguments in a verified Lustre compiler'. In: *JFLA 2019 - Les Trentièmes Journées Francophones des Langages Applicatifs*. Les actes des trentièmes Journées Francophones des Langages Applicatifs (JFLA 2019). Les Rousses, France, Jan. 2019, p. 16. URL: https://hal.inria.fr/hal-02005639.

[26] C. D. Canovas. 'Méthodes déductives pour la preuve de programmes Lustre'. PhD thesis. Université Joseph Fourier (Grenoble), Nov. 2000.

[27] P. Caspi, J.-L. Colaço, L. Gérard, M. Pouzet and P. Raymond. 'Synchronous Objects with Scheduling Policies: Introducing safe shared memory in Lustre'. In: *ACM International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*. Dublin, June 2009.

[28] P. Caspi and C. Dumas. 'A PVS Proof Obligation Generator for Lustre Programs'. In: *2nd International Conference on Logic for Programming and Reasonning, LPAR2000*. Vol. 1955. La Réunion: Lecture Notes in Artificial Intelligence, Nov. 2000.

[29] A. Cohen, M. Duranton, C. Eisenbeis, C. Pagetti, F. Plateau and M. Pouzet. 'N-Synchronous Kahn Networks: a Relaxed Model of Synchrony for Real-Time Systems'. In: *ACM International Conference on Principles of Programming Languages (POPL'06)*. Charleston, South Carolina, USA, Jan. 2006.

[30] A. Cohen, L. Mandel, F. Plateau and M. Pouzet. 'Abstraction of Clocks in Synchronous Data-flow Systems'. In: *The Sixth ASIAN Symposium on Programming Languages and Systems (APLAS)*. Bangalore, India, Dec. 2008.

[31] J.-L. Colaço, B. Pagano and M. Pouzet. 'A Conservative Extension of Synchronous Data-flow with State Machines'. In: *ACM International Conference on Embedded Software (EMSOFT'05)*. Jersey city, New Jersey, USA, Sept. 2005.

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

[33] P. Eades and X. Lin. 'A Heuristic for the Feedback Arc Set Problem'. In: *Australasian Journal of Combinatorics* 12 (Sept. 1995), pp. 15–25. URL: https://ajc.maths.uq.edu.au/pdf/12/ocr-ajc-v12-p15.pdf.

[34] P. Eades, X. Lin and W. Smyth. 'A Fast & Effective Heuristic for the Feedback Arc Set Problem'. In: *Information Processing Letters* 47.6 (Oct. 1993), pp. 319–323. DOI: 10.1016/0020-0190(93)90079-O.

[35] J. Forget. 'Un Langage Synchrone pour les Systèmes Embarqués Critiques Soumis à des Contraintes Temps Réel Multiples'. PhD thesis. Université de Toulouse, Nov. 2009.

[36] G. Kahn. 'The semantics of a simple language for parallel programming'. In: *IFIP 74 Congress*. North Holland, Amsterdam, 1974.

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

[38] L. Mandel, F. Plateau and M. Pouzet. 'Lucy-n: a n-Synchronous Extension of Lustre'. In: *Tenth International Conference on Mathematics of Program Construction (MPC 2010)*. Québec, Canada, June 2010. URL: http://www.lri.fr/~mandel/papiers/MandelPlateauPouzet-MPC-10.pdf.

[39] L. Mandel, F. Plateau and M. Pouzet. 'Static Scheduling of Latency Insensitive Designs with Lucy-n'. In: *International Conference on Formal Methods in Computer-Aided Design (FMCAD)*. Austin, Texas, USA, Oct. 2011.

[40] Z. Manna and A. Pnueli. *Temporal Verifications of Reactive Systems – safety*. Spinger, 1995.

[41] 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.

[42] F. Plateau. 'Modèle n-synchrone pour la programmation de réseaux de Kahn à mémoire bornée'. PhD thesis. Orsay, France: Université Paris-Sud 11, June 2010. URL: https://www.lri.fr/~mandel/lucy-n/~plateau/these/.

[43] M. Pouzet and P. Raymond. 'Modular Static Scheduling of Synchronous Data-flow Networks: An efficient symbolic representation'. In: *ACM International Conference on Embedded Software (EMSOFT'09)*. Grenoble, France, Oct. 2009.