RESEARCH CENTRE

Inria Paris Centre

IN PARTNERSHIP WITH:

CNRS, Ecole normale supérieure de Paris

2024

ACTIVITY REPORT

Team

PARKAS

Parallélisme de Kahn Synchrone

Inria teams are typically groups of researchers working on the definition of a common project, and objectives, with the goal to arrive at the creation of a project-team. Such project-teams may include other partners (universities or research institutions)

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

Team PARKAS

Creation of the Team: 2024 January 01

Keywords

Computer sciences and digital sciences

A1.1.1. – Multicore, Manycore

A1.1.2. – Hardware accelerators (GPGPU, FPGA, etc.)

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

A6.2.1. – Numerical analysis of PDE and ODE

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 Scientist

- Timothy Bourke [INRIA, Researcher]

Faculty Members

- Marc Pouzet [Team leader, Ecole normale supérieure, Professor]
- Paul Feautrier [DI-ENS, Emeritus]

Post-Doctoral Fellow

- Paul Jeanmaire [INRIA, from Oct 2024]

PhD Students

- Gregoire Bussone [ENS PARIS]
- Paul Jeanmaire [ENS PARIS, until Aug 2024]
- Charles de Haro [‘Ecole normale supérieure, from Aug 2024, Cosupervised with the ANTIQUE Team]

Technical Staff

- Remy Citerin [INRIA, Engineer, from Aug 2024 until Oct 2024]
- Loic Sylvestre [INRIA, Engineer, from Dec 2024]

Interns and Apprentices

- Lubin Bailly [Ecole normale supérieure, Intern, from Mar 2024 until Jul 2024]
- Hector Denis-Blanchardon [INRIA, Intern, from Apr 2024 until Sep 2024]
- Tita Philine Rosemeyer [INRIA, Intern, until Mar 2024]
- Charles de Haro [Ecole normale supérieure de Rennes, Intern, from Feb 2024 until Aug 2024, Cosupervised with the ANTIQUE team.]

Administrative Assistants

- Laurence Bourcier [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, 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 and compilers (Vélus). 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

Embedded control software is at the heart of many critical applications (medical devices, cars, planes, satellites, trains, energy factories). This complex software interacts in real time with a partially known physical environment and it must ensure statically strong safety constraints (e.g., determinacy, deadlock freedom, execution in bounded time and memory). Model-based design, in its most formal interpretation, is a way to increase confidence that the system and its software behave as expected. It is based on dedicated languages to write executable mathematical specifications (from control theory, physics) which are simulated, tested, formally verified and compiled to executable code. This approach is recognized today by certification authorities and supported by industrial tools and languages like SCADE and SIMULINK.

The compiler plays a central role in these languages: it performs static verification checks on models, source-to-source transformations, optimizations; it generates intermediate representations for formal verification, automatic testing; and finally, executable code for fast simulation and the implementation on an embedded target. Moreover, for hybrid systems modeling languages (e.g., Simulink, Modelica), the compiler does complex static analyses and transformations (e.g., index reduction, differentiation, computation of the Jacobian) in order to produce simulation code. The run-time system — the so-called "simulation engine" — which involves numeric solvers for differential equations and zero-crossing detection is itself a complex software whose correctness is critical to get confidence in simulation results. How to guarantee that all these compilation steps and the run-time system are correct? That what is simulated, tested and verified corresponds to what is written in the source program and remains true of the generated code? In other words, that we have a *high fidelity chain going from a mathematically precise specification to its faithful reproduction on a target architecture* or, to paraphrase Gérard Berry, that "what you simulate/prove on the model is what you execute" [19].

The scientific objective of PARKAS is to tackle this fundamental question, making concrete proposals in the form of language and compiler prototypes, to improve the best industrial practices for certified applications where SCADE is the state-of-the-art. We aim at extending the theory and practice on the design, semantics and implementation of *domain-specific languages for reactive systems*, allowing to go from a *precise specification down to trustable code*, with a *precise and traceable compilation chain* that gives strong and explainable evidence of correctness.

Our research draws its inspiration and focus from the simplicity and complementarity of the data-flow and deterministic model of parallelism introduced by Kahn and Mac Queen [28, 29], the theory and practice of synchronous programming,[25, 18, 23] typed functional programming [33] and the fruitful relationships between the two.[34, 35, 22, 31, 17] To reach our goal, we leverage a large body of formal principles: language and compiler design, semantics, dedicated type systems, proof engineering for compilers, synchronous circuits, and compilation algorithms.

Our research program has recently been organized along two axes.

1. The definition of languages for cyber-physical system allowing to write hybrid (discrete/continuous) models (e.g., the interaction between software and a physical environment). We have investigated how to express and exploit relaxed model of synchrony such as "communication by sampling" through bounded buffers; models that mix reactive control and array-based computer intensive applications; probabilistic programming constructs to model uncertainty; and random testing techniques for finding bugs.

2. An important and closely interconnected challenge is the ability to preserve the trustworthiness of a compiler chain, giving the greatest confidence in its correctness, from the model to the code. We

develop precise specifications (on paper) for all the language extensions we propose. When the solution is mature enough, we develop computer-aided and machine-checked formal semantics and compilation techniques. We have treated important language extensions (e.g., the mix of data-flow and hierarchical automata). We have also explored two executable denotational semantics. One is stream-based (or "Kahnian"); the other is state-based and lead to an effective interpreter.

In our research, we adopt and follow a *language-centric approach*, focusing our efforts on the development of concrete languages proposals, dedicated type systems, compile-time static analyses, compilation algorithms, etc. These softwares constitute essential "laboratories": they ground our scientific contributions, guide and validate our research through experimentation, and they are an important vehicle for teaching and long-standing collaborations with industry. ZELUS, VÉLUS, ZRUN, PRESSEAIL are visible results of our approach.

4 Application domains

4.1 Embedded Control Software

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

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

4.2 Hybrid Systems Design and Simulation

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

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

5 Highlights of the year

5.1 Awards

- Timothy Bourke received an *Outstanding Reviewer Award* at the 20th ACM/IEEE Embedded Systems Week in Raleigh, NC, USA.

6 New software, platforms, open data

6.1 New software

6.1.1 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

6.1.2 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

6.1.3 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

6.1.4 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

6.1.5 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 Gerard, Marc Pouzet

Partners: UGA, ENS Paris, Inria, LIG

6.1.6 ZRun

Name: The ZRun Synchronous Language Interpreter

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

7 New results

7.1 Verified compilation of Lustre

Participants: Timothy Bourke, Paul Jeanmaire, Marc Pouzet.

Vélus is a compiler for a subset of LUSTRE and SCADE that is specified in the Coq [24] Interactive Theorem Prover (ITP). It integrates the CompCert C compiler [30, 20] 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 by developing constructive denotational models to facilitate interactive verification.

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 completed the first phase of work to develop a Kahn-style semantics in Coq using C. Paulin-Mohring's library [32]. The model treats the dataflow core of Lustre with resettable node instances as presented in our EMSOFT 2021 article [21] 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 preserved 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. We proved that the relational definition of resettable node instances, used in the compiler correctness proof, corresponds with two previously proposed functional definitions [27, 26]. Clock typing constraints are necessary on a source program to maintain the possibility of a Kahn-style semantics. However, they can be relaxed within the compiler as successive passes rewrite the program into a normalized form, since the intermediate programs are assigned a synchronous semantics that defines the behaviour cycle-by-cycle. An article on these results has been submitted. Paul Jeanmaire defended his thesis on this topic in December 2024 [15].

The PhD work of Paul Jeanmaire specifies when the semantic predicates of Vélus admit at least one solution. By building on a determinancy theorem within CompCert, we strengthened the Vélus correctness theorem to show that the generated code only has one behavior — the one that corresponds to the dataflow source semantics.

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, Marc Pouzet.

External collaborators: Dumitru Potop Butucaru (Inria) and; Matthieu Boitrel, Marc Brundler, Matthieu David, Victor Jegu, Sylvain Sauvant, and Jean Souyris (Airbus).

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

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

This year we continued working on the possibility 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 results at the workshop on Time-Centric Reactive Software as part of ESWEEK 2024 in Raleigh, NC, USA [13].

We also continued working on integration with the LoPHt compiler developed by Dumitru Potop Butucaru. In this approach, our compiler assigns tasks to phases and LoPHt parallelizes the tasks within a phase by allocating them to cores and adding additional synchronization instructions. We tested the combined toolchain on case studies provided by Airbus.

In parallel, we started experimenting with an alternative approach where the ILP constraints also encode the allocation of tasks to cores. In our approach, inter-core communications within a single cycle are forbidden, and additional synchronization instructions are thus not needed. Results to date have been mixed. For small examples where no equations execute at the base rate, the approach works well. For large examples, the solution process still takes an unreasonable amount of time. We tried reducing the problem size using graph partitioning techniques, but the size reduction is limited and feasible solutions are often eliminated unless many partitions are used, somewhat defeating the purpose of this approach. We are currently working on alternative encodings that we hope will reduce the solution time. New ideas are still required to handle applications containing equations that execute at the base rate.

Together with our collaborators at Airbus and support from the transfer (Estelle Gaspard), legal (Manon Coger), and financial (Odette Dabire) services of Inria Paris, we contributed to a project proposal on mixing control algorithms and matrix manipulations within a dataflow synchronous language.

This work is funded by direct industrial contracts with Airbus.

Glossary

Integer Linear Programming (ILP): A problem is encoded as a list of linear constraints on real and integer variables, together with a linear goal expression. Solver software attempts to find values for the variables such that the constraints are satisfied while minimizing or maximizing the value of the goal expression.

7.3 The Zelus Language

Participants: Timothy Bourke, 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.

In 2024, we have started a new implementation of Zélus and its compiler (see the current development at the Zélus repository. The main new features are the following:

- The input langage now include new programming constructs: functions parameterized by (integer) sizes that are ultimately known at compile-time. Size expressions can appear in the type of arrays (e.g., iterators) and to define recursive functions on sizes (e.g., FFT, recursive search by dichotomy). Two important new constructs have been added : the forward loop construct which iterate a stream function on an array (cf. PhD of Baptiste Pauget) and the foreach loop constructs which corresponds to running several stream functions in parallel (e.g., also called "multi-instanciation).
- The type system and compilation has been extended to deal with those new constructs. Through the techniques are classical and well understood (essentially, a simple variant of HM(X)), it have demanded quite an important work.
- The compiler is now paired with ZRun which is used as an oracle for (black-box, random) testing of the compiler.

We simplified and reorganised several parts of the compiler; e.g., a new implementation of the typing phrase, a new way of implementing the sequence of source-to-source transformations for generating sequential code; new static analyses; and the pairing with ZRun for testing the compiler. We also wanted that the compiler organisation to be easier for other to use it for their research and experiment with new ideas.

One promising extension is driven by Prof. Jean-Baptiste Jeannin (Univ. Michigan, Ann Arbor), who is spending a sabbatical year at Parkas (June 2024-June 2025). With his students, he develops MarVeLus, an

extension of Zélus with refinement types. We collaborate on the extension of Zélus to integrate a new typing pass for formally verifying safety properties as types and expressed by synchronous observers.

7.4 A Constructive Synchronous Semantics

Participants: Baptiste Pauget, Marc Pouzet.

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

In 2024, we have continued the development of ZRun. We develop it in parallel with Zélus such that every new programming construct (and more generally, any language features) is implemented in both the compiler and the interpreter.

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.

7.5 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.

7.6 Verification by Abstract Interpretation of Synchronous Programs

Participants: Marc Pouzet, Charles De Haro, Xavier Rival.

In Sept. 2024, Charles de Haro have started his PhD. thesis (Dir. Xavier Rival, INRIA project team Antique; Marc Pouzet, INRIA project team Parkas) on the formal verification of synchronous programs. The objective is to verify safety properties for programs that mix data-flow equations, hierarchical automata and arrays. Those constructs are present in the industrial language Scade and the languages and compilers developed at Parkas, namely Zélus and the Vélus compiler.

While previous works (in particular, Bertrand Jeannet PhD. thesis, in 2000) have considered a subset of the language Lustre (a purely data-flow subset, without clocks), the objective here is to deal with the mix with control structures, like the by-case definition of streams and hierarchical automata, exploiting

structural informations that is present in the source. During first year, Charles have defined a new abstract semantics, reminiscent of the concrete, state-based semantics presented at Emsoft'23 and implemented in the ZRun interpreter.

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 contracts with Airbus.

9 Partnerships and cooperations

9.1 International research visitors

9.1.1 Visits of international scientists

- Jean-Baptiste Jeannin, Assistant Professor, University of Michigan — Ann Arbor, is on sabbatical in the team from 9/2024 until 8/2025.

Other international visits to the team

- Mary Sheeran, Professor, Chalmers University of Technology, visited the team in December 2024 with support from the Swedish Institute.

10 Dissemination

10.1 Promoting scientific activities

10.1.1 Scientific events: organisation

Member of the organizing committees

- Timothy Bourke was Web Chair for the 20th ACM/IEEE Embedded Systems Week in Raleigh, NC, USA.

10.1.2 Scientific events: selection

Chair of conference program committees

- Timothy Bourke co-chaired the PC of SETTA 2024, the Symposium on Dependable Software Engineering: Theories, Tools and Applications, held in Hong Kong, China.

Member of the conference program committees

- Timothy Bourke served on the PC of ECRTS 2024, the 36th Euromicro Conference on Real-Time Systems.
- Timothy Bourke served on the PC of EMSOFT 2024, the 24th International Conference on Embedded Software.
- Timothy Bourke served on the PC of DATE 2025 (track E1: Embedded software architecture, compilers and tool chains), Design, Automation and Test in Europe Conference.

- Timothy Bourke served on the PC of RTSOPS Workshop 2024, the 12th International Real-Time Scheduling Open Problems Seminar.

Reviewer

- Timothy Bourke reviewed articles for ESOP 2024, the 33rd European Symposium on Programming.

10.1.3 Journal

Reviewer - reviewing activities

- Timothy Bourke reviewed articles for Science of Computer Programming.

10.1.4 Invited talks

- Timothy Bourke gave a keynote talk at FDL 2024, Forum on Specification and Design Languages.

10.1.5 Research administration

- Timothy Bourke reviewed project proposals for the ANR.

10.2 Teaching - Supervision - Juries

10.2.1 Teaching

- Marc Pouzet is Director of Studies for the CS department, at ENS.
- Licence : Timothy Bourke: “Operating Systems” (L3), Lectures and TDs, ENS, France.
- Master : Marc Pouzet & Timothy Bourke, with presentations by Jean-Baptiste Jeannin, “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 Sergio 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 presented two lectures and TPs on Synchronous Languages in Carlos Agon’s course on concurrent models at Sorbonne Université.
- Bachelor: Timothy Bourke: “A Programmer’s introduction to Computer Architectures and Operating Systems" (L2), École Polytechnique, France
- Summer School: Timothy Bourke presented a course at the École Jeunes Chercheuses et Jeunes Chercheurs en Programmation, l’école du GDR GPL.

10.2.2 Supervision

- Marc Pouzet and Timothy Bourke supervised the PhD thesis of Gregoire Bussone.
- Timothy Bourke and Marc Pouzet supervised the PhD thesis of Paul Jeanmaire.
- Timothy Bourke supervised the L3 internship of Tita Philine Rosemeyer.
- Timothy Bourke and Marc Pouzet supervised the masters internship of Hector Denis-Blanchardon.
- Timothy Bourke and Marc Pouzet supervised the internship project of Remy Citerin.

10.2.3 Juries

- Timothy Bourke was an examiner for the “viva” of George Kaye at the University of Birmingham.

10.3 Popularization

10.3.1 Others science outreach relevant activities

- Timothy Bourke gave “Chiche” presentations at Livry Gargan, Rocquencourt, and Saint-Denis.

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] 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] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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

[13] T. Bourke and M. Pouzet. 'Lustre, Fast First and Fresh'. In: *IEEE Embedded Systems Letters* (2024), pp. 1–1. DOI: 10.1109/LES.2024.3498932. URL: https://inria.hal.science/hal-04815075 (cit. on p. 9).

Invited conferences

[14] 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.

Doctoral dissertations and habilitation theses

[15] P. Jeanmaire. 'A denotational semantics for a verified synchronous compiler'. Université PSL (Paris Sciences & Lettres), 20th Dec. 2024. URL: https://hal.science/tel-04885682 (cit. on p. 8).

Reports & preprints

[16] G. Baudart, L. Mandel and C. Tasson. *Density-Based Semantics for Reactive Probabilistic Programming*. 4th Mar. 2024. URL: https://hal.science/hal-04488216.

11.3 Cited publications

[17] A. Benveniste, T. Bourke, B. Caillaud, J.-L. Colaço, C. Pasteur and M. Pouzet. 'Building a Hybrid Systems Modeler on Synchronous Languages Principles'. In: *Proceedings of the IEEE*. Design Automation for Cyber-Physical Systems 106.9 (Sept. 2018), pp. 1568–1592. DOI: 10.1109/JPROC.2018.2858016. URL: https://hal.inria.fr/hal-01879026 (cit. on p. 3).

[18] A. Benveniste, P. Caspi, S. Edwards, N. Halbwachs, P. Le Guernic and R. de Simone. 'The synchronous languages 12 years later'. In: *Proceedings of the IEEE* 91.1 (Jan. 2003) (cit. on p. 3).

[19] G. Berry. 'Real Time programming: Special purpose or general purpose languages'. In: *IFIP Congress*. Elsevier Science Publishers, 1989, pp. 11–17 (cit. on p. 3).

[20] 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 (cit. on p. 7).

[21] 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 (cit. on p. 8).

[22] P. Caspi and M. Pouzet. 'Synchronous Kahn Networks'. In: *ACM SIGPLAN International Conference on Functional Programming (ICFP)*. Philadelphia, Pensylvania, May 1996 (cit. on p. 3).

[23] J.-L. Colaço, B. Pagano and M. Pouzet. 'Scade 6: A Formal Language for Embedded Critical Software Development'. In: *TASE 2017 - 11th International Symposium on Theoretical Aspects of Software Engineering*. Nice, France, Sept. 2017, pp. 1–10. URL: https://hal.inria.fr/hal-01666470 (cit. on p. 3).

[24] *The Coq proof Assistant*. http://coq.inria.fr. 2019 (cit. on p. 7).

[25] G. Berry. 'Real time programming: Special purpose or general purpose languages'. In: *Information Processing* 89 (1989), pp. 11–17 (cit. on p. 3).

[26] L. Gérard. 'Programmer le parallélisme avec des futures en Heptagon un langage synchrone flot de données et étude des réseaux de Kahn en vue d'une compilation synchrone'. PhD thesis. Paris, France: Université Paris-Sud, Sept. 2013. URL: https://theses.hal.science/tel-00929932 (cit. on p. 8).

[27] G. Hamon and M. Pouzet. 'Modular Resetting of Synchronous Data-Flow Programs'. In: *Proc. 2nd ACM SIGPLAN Int. Conf. on Principles and Practice of Declarative Programming (PPDP 2000)*. Ed. by F. Pfenning. Montreal, Canada: ACM, Sept. 2000, pp. 289–300. DOI: 10.1145/351268.351300. URL: https://www.di.ens.fr/~pouzet/bib/ppdp00.ps.gz (cit. on p. 8).

[28] G. Kahn. 'The semantics of a simple language for parallel programming'. In: *IFIP 74 Congress*. North Holland, Amsterdam, 1974 (cit. on p. 3).

[29] G. Kahn and D. B. MacQueen. 'Coroutines and Networks of Parallel Processes'. In: *IFIP Congress*. 1977, pp. 993–998 (cit. on p. 3).

[30] X. Leroy. *The Compcert verified compiler*. 2009. URL: http://compcert.inria.fr/doc/index.html (cit. on p. 7).

[31] L. Mandel and M. Pouzet. 'ReactiveML, a Reactive Extension to ML'. In: *ACM International Conference on Principles and Practice of Declarative Programming (PPDP)*. Lisboa, July 2005 (cit. on p. 3).

[32] 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 (cit. on p. 8).

[33] B. C. Pierce. *Types and Programming Languages*. MIT Press, 2002 (cit. on p. 3).

[34] M. Sheeran. 'muFP, a language for VLSI design'. In: *ACM Conference on LISP and Functional Programming*. Austin, Texas, 1984, pp. 104–112 (cit. on p. 3).

[35] Z. Wan and P. Hudak. 'Functional Reactive Programming from First Principles'. In: *International Conference on Programming Language, Design and Implementation (PLDI)*. 2000 (cit. on p. 3).