



RESEARCH CENTER

FIELD

**Algorithmics, Programming, Software and Architecture**

Activity Report 2016

# Section Scientific Foundations

Edition: 2017-08-25



## ALGORITHMICS, COMPUTER ALGEBRA AND CRYPTOLOGY

1. ARIC Project-Team	5
2. AROMATH Project-Team	10
3. CARAMBA Project-Team	12
4. CASCADE Project-Team	14
5. DATASHAPE Team	17
6. GRACE Project-Team	19
7. LFANT Project-Team	22
8. POLSYS Project-Team	25
9. SECRET Project-Team	29
10. SPECFUN Project-Team	31
11. VEGAS Project-Team (section vide)	36

## ARCHITECTURE, LANGUAGES AND COMPILATION

12. CAIRN Project-Team	37
13. CAMUS Team	40
14. COMPSYS Team	43
15. CORSE Project-Team	51
16. DREAMPAL Project-Team	52
17. PACAP Project-Team	55
18. TASC Project-Team	63

## EMBEDDED AND REAL-TIME SYSTEMS

19. AOSTE Project-Team	66
20. CONVECS Project-Team	70
21. HYCOMES Project-Team	74
22. MUTANT Project-Team	78
23. PARKAS Project-Team	82
24. POSET Team	85
25. SPADES Project-Team	86
26. TEA Project-Team	89

## PROOFS AND VERIFICATION

27. ANTIQUE Project-Team	93
28. CELTIQUE Project-Team (section vide)	95
29. DEDUCTEAM Team	96
30. GALLIUM Project-Team	98
31. MARELLE Project-Team	102
32. MEXICO Project-Team	103
33. PARSIFAL Project-Team	111
34. PIR2 Project-Team	114
35. SUMO Project-Team	120
36. TOCCATA Project-Team	122
37. VERIDIS Project-Team	132

## SECURITY AND CONFIDENTIALITY

38. CARTE Team .....	134
39. COMETE Project-Team .....	136
40. DICE Team .....	138
41. PESTO Project-Team .....	141
42. PRIVATICS Project-Team (section vide) .....	143
43. PROSECCO Project-Team .....	144
44. TAMIS Team .....	148

## ARIC Project-Team

### 3. Research Program

#### 3.1. Efficient approximation methods

##### 3.1.1. *Computer algebra generation of certified approximations*

We plan to focus on the generation of certified and efficient approximations for solutions of linear differential equations. These functions cover many classical mathematical functions and many more can be built by combining them. One classical target area is the numerical evaluation of elementary or special functions. This is currently performed by code specifically handcrafted for each function. The computation of approximations and the error analysis are major steps of this process that we want to automate, in order to reduce the probability of errors, to allow one to implement “rare functions”, to quickly adapt a function library to a new context: new processor, new requirements – either in terms of speed or accuracy.

In order to significantly extend the current range of functions under consideration, several methods originating from approximation theory have to be considered (divergent asymptotic expansions; Chebyshev or generalized Fourier expansions; Padé approximants; fixed point iterations for integral operators). We have done preliminary work on some of them. Our plan is to revisit them all from the points of view of effectivity, computational complexity (exploiting linear differential equations to obtain efficient algorithms), as well as in their ability to produce provable error bounds. This work is to constitute a major progress towards the automatic generation of code for moderate or arbitrary precision evaluation with good efficiency. Other useful, if not critical, applications are certified quadrature, the determination of certified trajectories of spatial objects and many more important questions in optimal control theory.

##### 3.1.2. *Digital Signal Processing*

As computer arithmeticians, a wide and important target for us is the design of efficient and certified linear filters in digital signal processing (DSP). Actually, following the advent of MATLAB as the major tool for filter design, the DSP experts now systematically delegate to MATLAB all the part of the design related to numerical issues. And yet, various key MATLAB routines are neither optimized, nor certified. Therefore, there is a lot of room for enhancing numerous DSP numerical implementations and there exist several promising approaches to do so.

The main challenge that we want to address over the next period is the development and the implementation of optimal methods for rounding the coefficients involved in the design of the filter. If done in a naive way, this rounding may lead to a significant loss of performance. We will study in particular FIR and IIR filters.

##### 3.1.3. *Table Maker’s Dilemma (TMD)*

There is a clear demand for hardest-to-round cases, and several computer manufacturers recently contacted us to obtain new cases. These hardest-to-round cases are a precious help for building libraries of correctly rounded mathematical functions. The current code, based on Lefèvre’s algorithm, will be rewritten and formal proofs will be done.

We plan to use uniform polynomial approximation and diophantine techniques in order to tackle the case of the IEEE quad precision, and analytic number theory techniques (exponential sums estimates) for counting the hardest-to-round cases.

## 3.2. Lattices: algorithms and cryptology

Lattice-based cryptography (LBC) is an utterly promising, attractive (and competitive) research ground in cryptography, thanks to a combination of unmatched properties:

- **Improved performance.** LBC primitives have low asymptotic costs, but remain cumbersome in practice (e.g., for parameters achieving security against computations of up to 2100 bit operations). To address this limitation, a whole branch of LBC has evolved where security relies on the restriction of lattice problems to a family of more structured lattices called *ideal lattices*. Primitives based on such lattices can have quasi-optimal costs (i.e., quasi-constant amortized complexities), outperforming all contemporary primitives. This asymptotic performance sometimes translates into practice, as exemplified by NTRUEncrypt.
- **Improved security.** First, lattice problems seem to remain hard even for quantum computers. Moreover, the security of most of LBC holds under the assumption that standard lattice problems are hard in the worst case. Oppositely, contemporary cryptography assumes that specific problems are hard with high probability, for some precise input distributions. Many of these problems were artificially introduced for serving as a security foundation of new primitives.
- **Improved flexibility.** The master primitives (encryption, signature) can all be realized based on worst-case (ideal) lattice assumptions. More evolved primitives such as ID-based encryption (where the public key of a recipient can be publicly derived from its identity) and group signatures, that were the playing-ground of pairing-based cryptography (a subfield of elliptic curve cryptography), can also be realized in the LBC framework, although less efficiently and with restricted security properties. More intriguingly, lattices have enabled long-wished-for primitives. The most notable example is homomorphic encryption, enabling computations on encrypted data. It is the appropriate tool to securely outsource computations, and will help overcome the privacy concerns that are slowing down the rise of the cloud.

We work on three directions, detailed now.

### 3.2.1. Lattice algorithms

All known lattice reduction algorithms follow the same design principle: perform a sequence of small elementary steps transforming a current basis of the input lattice, where these steps are driven by the Gram-Schmidt orthogonalisation of the current basis.

In the short term, we will fully exploit this paradigm, and hopefully lower the cost of reduction algorithms with respect to the lattice dimension. We aim at asymptotically fast algorithms with complexity bounds closer to those of basic and normal form problems (matrix multiplication, Hermite normal form). In the same vein, we plan to investigate the parallelism potential of these algorithms.

Our long term goal is to go beyond the current design paradigm, to reach better trade-offs between run-time and shortness of the output bases. To reach this objective, we first plan to strengthen our understanding of the interplay between lattice reduction and numerical linear algebra (how far can we push the idea of working on approximations of a basis?), to assess the necessity of using the Gram-Schmidt orthogonalisation (e.g., to obtain a weakening of LLL-reduction that would work up to some stage, and save computations), and to determine whether working on generating sets can lead to more efficient algorithms than manipulating bases. We will also study algorithms for finding shortest non-zero vectors in lattices, and in particular look for quantum accelerations.

We will implement and distribute all algorithmic improvements, e.g., within the `fpLLL` library. We are interested in high performance lattice reduction computations (see application domains below), in particular in connection with/continuation of the HPAC ANR project (algebraic computing and high performance consortium).

### 3.2.2. Lattice-based cryptography

Our long term goal is to demonstrate the superiority of lattice-based cryptography over contemporary public-key cryptographic approaches. For this, we will 1- Strengthen its security foundations, 2- Drastically improve the performance of its primitives, and 3- Show that lattices allow to devise advanced and elaborate primitives.

The practical security foundations will be strengthened by the improved understanding of the limits of lattice reduction algorithms (see above). On the theoretical side, we plan to attack two major open problems: Are ideal lattices (lattices corresponding to ideals in rings of integers of number fields) computationally as hard to handle as arbitrary lattices? What is the quantum hardness of lattice problems?

Lattice-based primitives involve two types of operations: sampling from discrete Gaussian distributions (with lattice supports), and arithmetic in polynomial rings such as  $(\mathbb{Z}/q\mathbb{Z})[x]/(x^n + 1)$  with  $n$  a power of 2. When such polynomials are used (which is the case in all primitives that have the potential to be practical), then the underlying algorithmic problem that is assumed hard involves ideal lattices. This is why it is crucial to precisely understand the hardness of lattice problems for this family. We will work on improving both types of operations, both in software and in hardware, concentrating on values of  $q$  and  $n$  providing security. As these problems are very arithmetic in nature, this will naturally be a source of collaboration with the other themes of the AriC team.

Our main objective in terms of cryptographic functionality will be to determine the extent to which lattices can help securing cloud services. For example, is there a way for users to delegate computations on their outsourced dataset while minimizing what the server eventually learns about their data? Can servers compute on encrypted data in an efficiently verifiable manner? Can users retrieve their files and query remote databases anonymously provided they hold appropriate credentials? Lattice-based cryptography is the only approach so far that has allowed to make progress into those directions. We will investigate the practicality of the current constructions, the extension of their properties, and the design of more powerful primitives, such as functional encryption (allowing the recipient to learn only a function of the plaintext message). To achieve these goals, we will in particular focus on cryptographic multilinear maps.

This research axis of AriC is gaining strength thanks to the recruitment of Benoit Libert. We will be particularly interested in the practical and operational impacts, and for this reason we envision a collaboration with an industrial partner.

### 3.2.3. Application domains

- Diophantine equations. Lattice reduction algorithms can be used to solve diophantine equations, and in particular to find simultaneous rational approximations to real numbers. We plan to investigate the interplay between this algorithmic task, the task of finding integer relations between real numbers, and lattice reduction. A related question is to devise LLL-reduction algorithms that exploit specific shapes of input bases. This will be done within the ANR DynA3S project.
- Communications. We will continue our collaboration with Cong Ling (Imperial College) on the use of lattices in communications. We plan to work on the wiretap channel over a fading channel (modeling cell phone communications in a fast moving environment). The current approaches rely on ideal lattices, and we hope to be able to find new approaches thanks to our expertise on them due to their use in lattice-based cryptography. We will also tackle the problem of sampling vectors from Gaussian distributions with lattice support, for a very small standard deviation parameter. This would significantly improve current schemes for communication schemes based on lattices, as well as several cryptographic primitives.
- Cryptanalysis of variants of RSA. Lattices have been used extensively to break variants of the RSA encryption scheme, via Coppersmith's method to find small roots of polynomials. We plan to work with Nadia Heninger (U. of Pennsylvania) on improving these attacks, to make them more practical. This is an excellent test case for testing the practicality of LLL-type algorithm. Nadia Heninger has a strong experience in large scale cryptanalysis based on Coppersmith's method (<http://smartfacts.cr.yp.to/>)

## 3.3. Algebraic computing and high performance kernels

The main theme here is the study of fundamental operations (“kernels”) on a hierarchy of symbolic or numeric data types spanning integers, floating-point numbers, polynomials, power series, as well as matrices of all these. Fundamental operations include basic arithmetic (e.g., how to multiply or how to invert) common to all

such data, as well as more specific ones (change of representation/conversions, GCDs, determinants, etc.). For such operations, which are ubiquitous and at the very core of computing (be it numerical, symbolic, or hybrid numeric-symbolic), our goal is to ensure both high performance and reliability.

### **3.3.1. Algorithms**

On the symbolic side, we will focus on the design and complexity analysis of algorithms for matrices over various domains (fields, polynomials, integers) and possibly with specific properties (structure). So far, our algorithmic improvements for polynomial matrices and structured matrices have been obtained in a rather independent way. Both types are well known to have much in common, but this is sometimes not reflected by the complexities obtained, especially for applications in cryptology and coding theory. Our goal in this area is thus to explore these connections further, to provide a more unified treatment, and eventually bridge these complexity gaps. A first step towards this goal will be the design of enhanced algorithms for various generalizations of Hermite-Padé approximation; in the context of list decoding, this should in particular make it possible to match or even improve over the structured-matrix approach, which is so far the fastest known.

On the other hand we will focus on the design of algorithms for certified computing. We will study the use of various representations, such as mid-rad for classical interval arithmetic, or affine arithmetic. We will explore the impact of precision tuning in intermediate computations, possibly dynamically, on the accuracy of the results (e.g. for iterative refinement and Newton iterations). We will continue to revisit and improve the classical error bounds of numerical linear algebra in the light of the subtleties of IEEE floating-point arithmetic.

Our goals in linear algebra and lattice basis reduction that have been detailed above in Section 3.2 will be achieved in the light of a hybrid symbolic-numeric approach.

### **3.3.2. Computer arithmetic**

Our work on certified computing and especially on the analysis of algorithms in floating-point arithmetic leads us to manipulate floating-point data in their greatest generality, that is, as symbolic expressions in the base and the precision. Our aim here is thus to develop theorems as well as efficient data structures and algorithms for handling such quantities by computer rather than by hand as we do now. The main outcome would be a “symbolic floating-point toolbox” which provides a way to check automatically the certificates of optimality we have obtained on the error bounds of various numerical algorithms.

We will also work on the interplay between floating-point and integer arithmetics. Currently, small numerical kernels like an exponential or a  $2 \times 2$  determinant are typically written using exclusively one of these two kinds of arithmetic. However, modern processors now have hardware support for both floating-point and integer arithmetics, often with vector (SIMD) extensions, and an important question is how to make the best use of all such capabilities to optimize for both accuracy and efficiency.

A third direction will be to work on algorithms for performing correctly-rounded arithmetic operations in medium precision as efficiently and reliably as possible. Indeed, many numerical problems require higher precision than the conventional floating-point (single, double) formats. One solution is to use multiple precision libraries, such as GNU MPFR, which allow the manipulation of very high precision numbers, but their generality (they are able to handle numbers with millions of digits) is a quite heavy alternative when high performance is needed. Our objective here is thus to design a multiple precision arithmetic library that would allow to tackle problems where a precision of a few hundred bits is sufficient, but which have strong performance requirements. Applications include the process of long-term iteration of chaotic dynamical systems ranging from the classical Henon map to calculations of planetary orbits. The designed algorithms will be formally proved.

Finally, our work on the IEEE 1788 standard leads naturally to the development of associated reference libraries for interval arithmetic. A first direction will be to implement IEEE 1788 interval arithmetic within MPFI, our library for interval arithmetic using the arbitrary precision floating-point arithmetic provided by MPFR: indeed, MPFI has been originally developed with definitions and handling of exceptions which are not compliant with IEEE 1788. Another one will be to provide efficient support for multiple-precision intervals,



in mid-rad representation and by developing MPFR-based code-generation tools aimed at handling families of functions.

### ***3.3.3. High-performance algorithms and software***

The algorithmic developments for medium precision floating-point arithmetic discussed above will lead to high performance implementations on GPUs. As a follow-up of the HPAC project (which ended in December 2015) we shall pursue the design and implementation of high performance linear algebra primitives and algorithms.

## AROMATH Project-Team

### 3. Research Program

#### 3.1. High order geometric modeling

The accurate description of shapes is a long standing problem in mathematics, with an important impact in many domains, inducing strong interactions between geometry and computation. Developing precise geometric modeling techniques is a critical issue in CAD-CAM. Constructing accurate models, that can be exploited in geometric applications, from digital data produced by cameras, laser scanners, observations or simulations is also a major issue in geometry processing. A main challenge is to construct models that can capture the geometry of complex shapes, using few parameters while being precise.

Our first objective is to develop methods, which are able to describe accurately and in an efficient way, objects or phenomena of geometric nature, using algebraic representations.

The approach followed in CAGD, to describe complex geometry is based on parametric representations called NURBS (Non Uniform Rational B-Spline). The models are constructed by trimming and gluing together high order patches of algebraic surfaces. These models are built from the so-called B-Spline functions that encode a piecewise algebraic function with a prescribed regularity at the seams. Although these models have many advantages and have become the standard for designing nowadays CAD models, they also have important drawbacks. Among them, the difficulty to locally refine a NURBS surface and also the topological rigidity of NURBS patches that imposes to use many such patches with trims for designing complex models, with the consequence of the appearing of cracks at the seams. To overcome these difficulties, an active area of research is to look for new blending functions for the representation of CAD models. Some examples are the so-called T-Splines, LR-Spline blending functions, or hierarchical splines, that have been recently devised in order to perform efficiently local refinement. An important problem is to analyze spline spaces associated to general subdivisions, which is of particular interest in higher order Finite Element Methods. Another challenge in geometric modeling is the efficient representation and/or reconstruction of complex objects, and the description of computational domains in numerical simulation. To construct models that can represent efficiently the geometry of complex shapes, we are interested in developing modeling methods, based on alternative constructions such as skeleton-based representations. The change of representation, in particular between parametric and implicit representations, is of particular interest in geometric computations and in its applications in CAGD.

We also plan to investigate adaptive hierarchical techniques, which can locally improve the approximation of a shape or a function. They shall be exploited to transform digital data produced by cameras, laser scanners, observations or simulations into accurate and structured algebraic models.

The precise and efficient representation of shapes also leads to the problem of extracting and exploiting characteristic properties of shapes such as symmetry, which is very frequent in geometry. Reflecting the symmetry of the intended shape in the representation appears as a natural requirement for visual quality, but also as a possible source of sparsity of the representation. Recognizing, encoding and exploiting symmetry requires new paradigms of representation and further algebraic developments. Algebraic foundations for the exploitation of symmetry in the context of non linear differential and polynomial equations are addressed. The intent is to bring this expertise with symmetry to the geometric models and computations developed by AROMATH.

#### 3.2. Robust algebraic-geometric computation

In many problems, digital data are approximated and cannot just be used as if they were exact. In the context of geometric modeling, polynomial equations appear naturally, as a way to describe constraints between the unknown variables of a problem. *An important challenge is to take into account the input error in order to*

*develop robust methods for solving these algebraic constraints.* Robustness means that a small perturbation of the input should produce a controlled variation of the output, that is forward stability, when the input-output map is regular. In non-regular cases, robustness also means that the output is an exact solution, or the most coherent solution, of a problem with input data in a given neighborhood, that is backward stability.

Our second long term objective is to develop methods to robustly and efficiently solve algebraic problems that occur in geometric modeling.

Robustness is a major issue in geometric modeling and algebraic computation. Classical methods in computer algebra, based on the paradigm of exact computation, cannot be applied directly in this context. They are not designed for stability against input perturbations. New investigations are needed to develop methods, which integrate this additional dimension of the problem. Several approaches are investigated to tackle these difficulties.

One is based on linearization of algebraic problems based on “elimination of variables” or projection into a space of smaller dimension. Resultant theory provides strong foundation for these methods, connecting the geometric properties of the solutions with explicit linear algebra on polynomial vector spaces, for families of polynomial systems (e.g., homogeneous, multi-homogeneous, sparse). Important progresses have been made in the last two decades to extend this theory to new families of problems with specific geometric properties. Additional advances have been achieved more recently to exploit the syzygies between the input equations. This approach provides matrix based representations, which are particularly powerful for approximate geometric computation on parametrized curves and surfaces. They are tuned to certain classes of problems and an important issue is to detect and analyze degeneracies and to adapt them to these cases.

A more adaptive approach involves linear algebra computation in a hierarchy of polynomial vector spaces. It produces a description of quotient algebra structures, from which the solutions of polynomial systems can be recovered. This family of methods includes Gröbner Basis, which provides general tools for solving polynomial equations. Border Basis is an alternative approach, offering numerically stable methods for solving polynomial equations with approximate coefficients. An important issue is to understand and control the numerical behavior of these methods as well as their complexity and to exploit the structure of the input system.

In order to compute “only” the (real) solutions of a polynomial system in a given domain, duality techniques can also be employed. They consist in analyzing and adding constraints on the space of linear forms which vanish on the polynomial equations. Combined with semi-definite programming techniques, they provide efficient methods to compute the real solutions of algebraic equations or to solve polynomial optimization problems. The main issues are the completeness of the approach, their scalability with the degree and dimension and the certification of bounds.

Singular solutions of polynomial systems can be analyzed by computing differentials, which vanish at these points. This leads to efficient deflation techniques, which transform a singular solution of a given problem into a regular solution of the transformed problem. These local methods need to be combined with more global root localisation methods.

Subdivision methods are another type of methods which are interesting for robust geometric computation. They are based on exclusion tests which certify that no solution exists in a domain and inclusion tests, which certify the uniqueness of a solution in a domain. They have shown their strength in addressing many algebraic problems, such as isolating real roots of polynomial equations or computing the topology of algebraic curves and surfaces. The main issues in these approaches is to deal with singularities and degenerate solutions.

## CARAMBA Project-Team

### 3. Research Program

#### 3.1. The Extended Family of the Number Field Sieve

The Number Field Sieve (NFS) has been the leading algorithm for factoring integers for more than 20 years, and its variants have been used to set records for discrete logarithms in finite fields. It is reasonable to understand NFS as a framework that can be used to solve various sorts of problems. Factoring integers and computing discrete logarithms are the most prominent for the cryptographic observer, but the same framework can also be applied to the computation of class groups.

The state of the art with NFS is built from numerous improvements of its inner steps. In terms of algorithmic improvements, the recent research activity on the NFS family has been rather intense. Several new algorithms have been discovered in over the 2014–2016 period, and their practical reach has been demonstrated by actual experiments.

The algorithmic contributions of the CARAMBA members to NFS would hardly be possible without access to a dependable software implementation. To this end, members of the CARAMBA team have been developing the Cado-NFS software suite since 2007. Cado-NFS is now the most widely visible open source implementation of NFS, and is a crucial platform for developing prototype implementations for new ideas for the many sub-algorithms of NFS. Cado-NFS is free software (LGPL) and follows an open development model, with publicly accessible development repository and regular software releases. Competing free software implementations exist, such as `msieve`, developed by J. Papadopoulos. In Lausanne, T. Kleinjung develops his own code base, which is unfortunately not public.

The workplan of CARAMBA on the topic of the Number Field Sieve algorithm and its cousins includes the following aspects:

- Pursue the work on NFS, which entails in particular making it ready to tackle larger challenges. Several of the important computational steps of NFS that are currently identified as stumbling blocks will require algorithmic advances and implementation improvements. We will illustrate the importance of this work by computational records.
- Work on the specific aspects of the computation of discrete logarithms in finite fields.
- As a side topic, the application of the broad methodology of NFS to the treatment of “ideal lattices” and their use in cryptographic proposals based on Euclidean lattices is also relevant.

#### 3.2. Algebraic Curves in Cryptology

The challenges associated to algebraic curves in cryptology are diverse, because of the variety of mathematical objects to be considered. These challenges are also connected to each other. On the cryptographic side, efficiency matters. As of 2016, the most widely used set of elliptic curves, the so-called NIST curves, are in the process of being replaced by a new set of candidate elliptic curves for future standardization. This is the topic of RFC 7748 [34].

On the cryptanalytic side, the discrete logarithm problem on (Jacobians of) curves has resisted all attempts for many years. Among the currently active topics, the decomposition algorithms raise interesting problems related to polynomial system solving, as do attempts to solve the discrete logarithm problem on curves defined over binary fields. In particular, while it is generally accepted that the so-called Koblitz curves (base field extensions of curves defined over  $\text{GF}(2)$ ) are likely to be a weak class among the various curve choices, no concrete attack supports this claim fully.

The research objectives of CARAMBA on the topic of algebraic curves for cryptology are as follows:

- Work on the practical realization of some of the rich mathematical theory behind algebraic curves. In particular, some of the fundamental mathematical objects have potentially important connections to the broad topic of cryptology: Abel-Jacobi map, Theta functions, computation of isogenies, computation of endomorphisms, complex multiplication.
- Improve the point counting algorithms so as to be able to tackle larger problems. This includes significant work connected to polynomial systems.
- Seek improvements on the computation of discrete logarithms on curves, including by identifying weak instances of this problem.

### 3.3. Computer Arithmetic

Computer arithmetic is part of the common background of all team members, and is naturally ubiquitous in the two previous application domains mentioned. However involved the mathematical objects considered may be, dealing with them first requires to master more basic objects: integers, finite fields, polynomials, and real and complex floating-point numbers. Libraries such as GNU MP, GNU MPFR, GNU MPC do an excellent job for these, both for small and large sizes (we rarely, if ever, focus on small-precision floating-point data, which explains our lack of mention of libraries relevant to it).

Most of our involvement in subjects related to computer arithmetic is to be understood in connection to our applications to the Number Field Sieve and to abelian varieties. As such, much of the research work we envision will appear as side-effects of developments in these contexts. On the topic of arithmetic work *per se*:

- We will seek algorithmic and practical improvements to the most basic algorithms. That includes for example the study of advances algorithms for integer multiplication, and their practical reach.
- We will continue to work on the arithmetic libraries in which we have crucial involvement, such as GNU MPFR, GNU MPC, GF2X, MPFQ, and also GMP-ECM.

### 3.4. Polynomial Systems

Systems of polynomial equations have been part of the cryptographic landscape for quite some time, with applications to the cryptanalysis of block and stream ciphers, as well as multivariate cryptographic primitives.

Polynomial systems arising from cryptology are usually not generic, in the sense that they have some distinct structural properties, such as symmetries, or bi-linearity for example. During the last decades, several results have shown that identifying and exploiting these structures can lead to dedicated Gröbner bases algorithms that can achieve large speedups compared to generic implementations [27], [26].

Solving polynomial systems is well done by existing software, and duplicating this effort is not relevant. However we develop test-bed open-source software for ideas relevant to the specific polynomial systems that arise in the context of our applications. The TinyGB software, that we describe further in 6.3, is our platform to test new ideas.

We aim to work on the topic of polynomial system solving in connection with our involvement in the aforementioned topics.

- We have high expertise on Elliptic Curve Discrete Logarithm Problem on small characteristic finite fields, because it also involves highly structured polynomial systems. While so far we have not contributed to this hot topic, this could of course change in the future.
- Recent hirings (Minier) are likely to lead the team to study particular polynomial systems in context which are more related to symmetric key cryptography.
- More centered on polynomial systems *per se*, we will mainly pursue the study of the specificities of the polynomial systems that are strongly linked to our targeted applications, and for which we have significant expertise [27], [26]. We also want to see these recent results provide practical benefits compared to existing software, in particular for systems relevant for cryptanalysis.

## **CASCADE Project-Team**

### **3. Research Program**

#### **3.1. Randomness in Cryptography**

Randomness is a key ingredient for cryptography. Random bits are necessary not only for generating cryptographic keys, but are also often an important part of cryptographic algorithms. In some cases, probabilistic protocols make it possible to perform tasks that are impossible deterministically. In other cases, probabilistic algorithms are faster, more space efficient or simpler than known deterministic algorithms. Cryptographers usually assume that parties have access to perfect randomness but in practice this assumption is often violated and a large body of research is concerned with obtaining such a sequence of random or pseudorandom bits.

One of the project-team research goals is to get a better understanding of the interplay between randomness and cryptography and to study the security of various cryptographic protocols at different levels (information-theoretic and computational security, number-theoretic assumptions, design and provable security of new and existing constructions).

Cryptographic literature usually pays no attention to the fact that in practice randomness is quite difficult to generate and that it should be considered as a resource like space and time. Moreover since the perfect randomness abstraction is not physically realizable, it is interesting to determine whether imperfect randomness is “good enough” for certain cryptographic algorithms and to design algorithms that are robust with respect to deviations of the random sources from true randomness.

The power of randomness in computation is a central problem in complexity theory and in cryptography. Cryptographers should definitely take these considerations into account when proposing new cryptographic schemes: there exist computational tasks that we only know how to perform efficiently using randomness but conversely it is sometimes possible to remove randomness from probabilistic algorithms to obtain efficient deterministic counterparts. Since these constructions may hinder the security of cryptographic schemes, it is of high interest to study the efficiency/security tradeoff provided by randomness in cryptography.

Quite often in practice, the random bits in cryptographic protocols are generated by a pseudorandom number generation process. When this is done, the security of the scheme of course depends in a crucial way on the quality of the random bits produced by the generator. Despite the importance, many protocols used in practice often leave unspecified what pseudorandom number generation to use. It is well-known that pseudorandom generators exist if and only if one-way functions exist and there exist efficient constructions based on various number-theoretic assumptions. Unfortunately, these constructions are too inefficient and many protocols used in practice rely on “ad-hoc” constructions. It is therefore interesting to propose more efficient constructions, to analyze the security of existing ones and of specific cryptographic constructions that use weak pseudorandom number generators.

The project-team undertakes research in these three aspects. The approach adopted is both theoretical and practical, since we provide security results in a mathematical frameworks (information theoretic or computational) with the aim to design protocols among the most efficient known.

#### **3.2. Lattice Cryptography**

The security of almost all public-key cryptographic protocols in use today relies on the presumed hardness of problems from number theory such as factoring and discrete log. This is somewhat problematic because these problems have very similar underlying structure, and its unforeseen exploit can render all currently used public key cryptography insecure. This structure was in fact exploited by Shor to construct efficient quantum algorithms that break all hardness assumptions from number theory that are currently in use. And so naturally, an important area of research is to build provably-secure protocols based on mathematical problems that are unrelated to factoring and discrete log. One of the most promising directions in this line of research is using lattice problems as a source of computational hardness —in particular since they also offer features that other alternative public-key cryptosystems (such as MQ-based, code-based or hash-based schemes) cannot provide.

At its very core, secure communication rests on two foundations: authenticity and secrecy. Authenticity assures the communicating parties that they are indeed communicating with each other and not with some potentially malicious outside party. Secrecy is necessary so that no one except the intended recipient of a message is able to deduce anything about its contents.

Lattice cryptography might find applications towards constructing practical schemes for resolving essential cryptographic problems—in particular, guaranteeing authenticity. On this front, our team is actively involved in pursuing the following two objectives:

1. Construct, implement, and standardize a practical public key digital signature scheme that is secure against quantum adversaries.
2. Construct, implement, and standardize a symmetric key authentication scheme that is secure against side channel attacks and is more efficient than the basic scheme using AES with masking.

Despite the great progress in constructing fairly practical lattice-based encryption and signature schemes, efficiency still remains a very large obstacle for advanced lattice primitives. While constructions of identity-based encryption schemes, group signature schemes, functional encryption schemes, and even fully-homomorphic encryption schemes are known, the implementations of these schemes are extremely inefficient.

Fully Homomorphic Encryption (FHE) is a very active research area. Let us just give one example illustrating the usefulness of computing on encrypted data: Consider an on-line patent database on which firms perform complex novelty queries before filing patents. With current technologies, the database owner might analyze the queries, infer the invention and apply for a patent before the genuine inventor. While such frauds were not reported so far, similar incidents happen during domain name registration. Several websites propose “registration services” preceded by “availability searches”. These queries trigger the automated registration of the searched domain names which are then proposed for sale. Algorithms allowing arbitrary computations without disclosing their inputs (and/or their results) are hence of immediate usefulness.

In 2009, IBM announced the discovery of a FHE scheme by Craig Gentry. The security of this algorithm relies on worst-case problems over ideal lattices and on the hardness of the sparse subset sum problem. Gentry’s construction is an ingenious combination of two ideas: a somewhat homomorphic scheme (capable of supporting many “logical or” operations but very few “ands”) and a procedure that refreshes the homomorphically processed ciphertexts. Gentry’s main conceptual achievement is a “bootstrapping” process in which the somewhat homomorphic scheme evaluates its own decryption circuit (self-reference) to refresh (recrypt) ciphertexts.

Unfortunately, it is safe to surmise that if the state of affairs remains as it is in the present, then despite all the theoretical efforts that went into their constructions, these schemes will never be used in practical applications.

Our team is looking at the foundations of these primitives with the hope of achieving a breakthrough that will allow them to be practical in the near future.

### **3.3. Security amidst Concurrency on the Internet**

Cryptographic protocols that are secure when executed in isolation, can be completely insecure when multiple such instances are executed concurrently (as is unavoidable on the Internet) or when used as a part of a larger protocol. For instance, a man-in-the-middle attacker participating in two simultaneous executions of a cryptographic protocol might use messages from one of the executions in order to compromise the security of the second – Lowe’s attack on the Needham-Schroeder authentication protocol and Bleichenbacher’s attack on SSL work this way. Our research addresses security amidst concurrent executions in secure computation and key exchange protocols.

Secure computation allows several mutually distrustful parties to collaboratively compute a public function of their inputs, while providing the same security guarantees as if a trusted party had performed the computation. Potential applications for secure computation include anonymous voting as well as privacy-preserving auctions and data-mining. Our recent contributions on this topic include

1. new protocols for secure computation in a model where each party interacts only once, with a single centralized server; this model captures communication patterns that arise in many practical settings, such as that of Internet users on a website,

2. and efficient constructions of universally composable commitments and oblivious transfer protocols, which are the main building blocks for general secure computation.

In key exchange protocols, we are actively involved in designing new password-authenticated key exchange protocols, as well as the analysis of the widely-used SSL/TLS protocols.

### 3.4. Electronic Currencies

Electronic cash (e-cash) was first proposed in the 1980s but despite extensive research it has never been deployed on a large scale. Other means of digital payments have instead largely replaced cash and other “analog” payments. Common to all digital payments offered by banks and other payment providers is that they do not respect the citizens’ right to privacy, which for legitimate purchases and moderate sums also includes their right of anonymous payments.

Recently the rise of so-called decentralized currencies, such as Bitcoin and the numerous “alt-coins” inspired by it, have established a third way of payments in addition to physical cash, which offers privacy, and card and other electronic payments, which are traceable by its providers. The continuous growth of popularity and usage of this new kind of currencies, also called “cryptocurrencies” as their security and stability crucially relies on the use of cryptography, have triggered a renewed interest in cryptographic e-cash.

Our group investigates “centralized” e-cash, which respects the current economic model where money is issued by (central) banks, as opposed to cryptocurrencies, which use money distribution to incentivize widespread participation in the system, required for stability. Of particular interest among centralized e-cash schemes is transferable e-cash, which allows users to transfer coins between each other without any interaction with a third party. Currently all efficient e-cash schemes require coins to be deposited at the bank once received; they are thus not transferable. Our goal is to propose efficient transferable e-cash schemes.

Another direction concerns cryptocurrencies whose adoption is continuously growing so that now even central banks, like the Swedish *Riksbank*, are considering issuing their own currency as a cryptocurrency. While systems like Bitcoin are perceived as offering anonymous payments, a line of research has shown that this is not the case. One of the major research challenges in this area is thus to devise schemes that offer an anonymity level comparable to that of physical cash. The currently proposed schemes either lack formal security analyses or they are inefficient due to the heavy-duty cryptography used. Our group works towards practical cryptocurrencies with formally analyzed privacy guarantees.



## **DATASHAPE Team**

### **3. Research Program**

#### **3.1. Algorithmic aspects of topological and geometric data analysis**

TDA requires to construct and manipulate appropriate representations of complex and high dimensional shapes. A major difficulty comes from the fact that the complexity of data structures and algorithms used to approximate shapes rapidly grows as the dimensionality increases, which makes them intractable in high dimensions. We focus our research on simplicial complexes which offer a convenient representation of general shapes and generalize graphs and triangulations. Our work includes the study of simplicial complexes with good approximation properties and the design of compact data structures to represent them.

In low dimensions, effective shape reconstruction techniques exist that can provide precise geometric approximations very efficiently and under reasonable sampling conditions. Extending those techniques to higher dimensions as is required in the context of TDA is problematic since almost all methods in low dimensions rely on the computation of a subdivision of the ambient space. A direct extension of those methods would immediately lead to algorithms whose complexities depend exponentially on the ambient dimension, which is prohibitive in most applications. A first direction to by-pass the curse of dimensionality is to develop algorithms whose complexities depend on the intrinsic dimension of the data (which most of the time is small although unknown) rather than on the dimension of the ambient space. Another direction is to resort to cruder approximations that only captures the homotopy type or the homology of the sampled shape. The recent theory of persistent homology provides a powerful and robust tool to study the homology of sampled spaces in a stable way.

#### **3.2. Statistical aspects of topological and geometric data analysis**

The wide variety of larger and larger available data - often corrupted by noise and outliers - requires to consider the statistical properties of their topological and geometric features and to propose new relevant statistical models for their study.

There exist various statistical and machine learning methods intending to uncover the geometric structure of data. Beyond manifold learning and dimensionality reduction approaches that generally do not allow to assert the relevance of the inferred topological and geometric features and are not well-suited for the analysis of complex topological structures, set estimation methods intend to estimate, from random samples, a set around which the data is concentrated. In these methods, that include support and manifold estimation, principal curves/manifolds and their various generalizations to name a few, the estimation problems are usually considered under losses, such as Hausdorff distance or symmetric difference, that are not sensitive to the topology of the estimated sets, preventing these tools to directly infer topological or geometric information.

Regarding purely topological features, the statistical estimation of homology or homotopy type of compact subsets of Euclidean spaces, has only been considered recently, most of the time under the quite restrictive assumption that the data are randomly sampled from smooth manifolds.

In a more general setting, with the emergence of new geometric inference tools based on the study of distance functions and algebraic topology tools such as persistent homology, computational topology has recently seen an important development offering a new set of methods to infer relevant topological and geometric features of data sampled in general metric spaces. The use of these tools remains widely heuristic and until recently there were only a few preliminary results establishing connections between geometric inference, persistent homology and statistics. However, this direction has attracted a lot of attention over the last three years. In particular, stability properties and new representations of persistent homology information have led to very promising results to which the DATASHAPE members have significantly contributed. These preliminary results open many perspectives and research directions that need to be explored.

Our goal is to build on our first statistical results in TDA to develop the mathematical foundations of Statistical Topological and Geometric Data Analysis. Combined with the other objectives, our ultimate goal is to provide a well-founded and effective statistical toolbox for the understanding of topology and geometry of data.

### **3.3. Topological approach for multimodal data processing**

Due to their geometric nature, multimodal data (images, video, 3D shapes, etc.) are of particular interest for the techniques we develop. Our goal is to establish a rigorous framework in which data having different representations can all be processed, mapped and exploited jointly. This requires adapting our tools and sometimes developing entirely new or specialized approaches.

The choice of multimedia data is motivated primarily by the fact that the amount of such data is steadily growing (with e.g. video streaming accounting for nearly two thirds of peak North-American Internet traffic, and almost half a billion images being posted on social networks each day), while at the same time it poses significant challenges in designing informative notions of (dis)-similarity as standard metrics (e.g. Euclidean distances between points) are not relevant.

### **3.4. Experimental research and software development**

We develop a high quality open source software platform called GUDHI which is becoming a reference in geometric and topological data analysis in high dimensions. The goal is not to provide code tailored to the numerous potential applications but rather to provide the central data structures and algorithms that underly applications in geometric and topological data analysis.

The development of the GUDHI platform also serves to benchmark and optimize new algorithmic solutions resulting from our theoretical work. Such development necessitates a whole line of research on software architecture and interface design, heuristics and fine-tuning optimization, robustness and arithmetic issues, and visualization. We aim at providing a full programming environment following the same recipes that made up the success story of the CGAL library, the reference library in computational geometry.

Some of the algorithms implemented on the platform will also be interfaced to other software platform, such as the R software<sup>0</sup> for statistical computing, and languages such as Python in order to make them usable in combination with other data analysis and machine learning tools. A first attempt in this direction has been done with the creation of an R package called TDA in collaboration with the group of Larry Wasserman at Carnegie Mellon University (Inria Associated team CATS) that already includes some functionalities of the GUDHI library and implements some joint results between our team and the CMU team. A similar interface with the Python language is also considered a priority. To go even further towards helping users, we will provide utilities that perform the most common tasks without requiring any programming at all.

---

<sup>0</sup><https://www.r-project.org/>

## GRACE Project-Team

### 3. Research Program

#### 3.1. Algorithmic Number Theory

Algorithmic Number Theory is concerned with replacing special cases with general algorithms to solve problems in number theory. In the Grace project, it appears in three main threads:

- fundamental algorithms for integers and polynomials (including primality and factorization);
- algorithms for finite fields (including discrete logarithms); and
- algorithms for algebraic curves.

Clearly, we use computer algebra in many ways. Research in cryptology has motivated a renewed interest in Algorithmic Number Theory in recent decades—but the fundamental problems still exist *per se*. Indeed, while algorithmic number theory application in cryptanalysis is epitomized by applying factorization to breaking RSA public key, many other problems, are relevant to various area of computer science. Roughly speaking, the problems of the cryptological world are of bounded size, whereas Algorithmic Number Theory is also concerned with asymptotic results.

#### 3.2. Arithmetic Geometry: Curves and their Jacobians

Theme: Arithmetic Geometry: Curves and their Jacobians *Arithmetic Geometry* is the meeting point of algebraic geometry and number theory: that is, the study of geometric objects defined over arithmetic number systems (such as the integers and finite fields). The fundamental objects for our applications in both coding theory and cryptology are curves and their Jacobians over finite fields.

An algebraic *plane curve*  $\mathcal{X}$  over a field  $\mathbf{K}$  is defined by an equation

$$\mathcal{X} : F_{\mathcal{X}}(x, y) = 0 \quad \text{where } F_{\mathcal{X}} \in \mathbf{K}[x, y].$$

(Not every curve is planar—we may have more variables, and more defining equations—but from an algorithmic point of view, we can always reduce to the plane setting.) The *genus*  $g_{\mathcal{X}}$  of  $\mathcal{X}$  is a non-negative integer classifying the essential geometric complexity of  $\mathcal{X}$ ; it depends on the degree of  $F_{\mathcal{X}}$  and on the number of singularities of  $\mathcal{X}$ . The curve  $\mathcal{X}$  is associated in a functorial way with an algebraic group  $J_{\mathcal{X}}$ , called the *Jacobian* of  $\mathcal{X}$ . The group  $J_{\mathcal{X}}$  has a geometric structure: its elements correspond to points on a  $g_{\mathcal{X}}$ -dimensional projective algebraic group variety. Typically, we do not compute with the equations defining this projective variety: there are too many of them, in too many variables, for this to be convenient. Instead, we use fast algorithms based on the representation in terms of classes of formal sums of points on  $\mathcal{X}$ .

The simplest curves with nontrivial Jacobians are curves of genus 1, known as *elliptic curves*; they are typically defined by equations of the form  $y^2 = x^3 + Ax + B$ . Elliptic curves are particularly important given their central role in public-key cryptography over the past two decades. Curves of higher genus are important in both cryptography and coding theory.

#### 3.3. Curve-Based cryptology

Theme: Curve-Based Cryptology

Jacobians of curves are excellent candidates for cryptographic groups when constructing efficient instances of public-key cryptosystems. Diffie–Hellman key exchange is an instructive example.

Suppose Alice and Bob want to establish a secure communication channel. Essentially, this means establishing a common secret *key*, which they will then use for encryption and decryption. Some decades ago, they would have exchanged this key in person, or through some trusted intermediary; in the modern, networked world, this is typically impossible, and in any case completely unscalable. Alice and Bob may be anonymous parties who want to do e-business, for example, in which case they cannot securely meet, and they have no way to be sure of each other's identities. Diffie–Hellman key exchange solves this problem. First, Alice and Bob publicly agree on a cryptographic group  $G$  with a generator  $P$  (of order  $N$ ); then Alice secretly chooses an integer  $a$  from  $[1..N]$ , and sends  $aP$  to Bob. In the meantime, Bob secretly chooses an integer  $b$  from  $[1..N]$ , and sends  $bP$  to Alice. Alice then computes  $a(bP)$ , while Bob computes  $b(aP)$ ; both have now computed  $abP$ , which becomes their shared secret key. The security of this key depends on the difficulty of computing  $abP$  given  $P$ ,  $aP$ , and  $bP$ ; this is the Computational Diffie–Hellman Problem (CDHP). In practice, the CDHP corresponds to the Discrete Logarithm Problem (DLP), which is to determine  $a$  given  $P$  and  $aP$ .

This simple protocol has been in use, with only minor modifications, since the 1970s. The challenge is to create examples of groups  $G$  with a relatively compact representation and an efficiently computable group law, and such that the DLP in  $G$  is hard (ideally approaching the exponential difficulty of the DLP in an abstract group). The Pohlig–Hellman reduction shows that the DLP in  $G$  is essentially only as hard as the DLP in its largest prime-order subgroup. We therefore look for compact and efficient groups of prime order.

The classic example of a group suitable for the Diffie–Hellman protocol is the multiplicative group of a finite field  $\mathbf{F}_q$ . There are two problems that render its usage somewhat less than ideal. First, it has too much structure: we have a subexponential Index Calculus attack on the DLP in this group, so while it is very hard, the DLP falls a long way short of the exponential difficulty of the DLP in an abstract group. Second, there is only one such group for each  $q$ : its subgroup treillis depends only on the factorization of  $q - 1$ , and requiring  $q - 1$  to have a large prime factor eliminates many convenient choices of  $q$ .

This is where Jacobians of algebraic curves come into their own. First, elliptic curves and Jacobians of genus 2 curves do not have a subexponential index calculus algorithm: in particular, from the point of view of the DLP, a generic elliptic curve is currently *as strong as* a generic group of the same size. Second, they provide some diversity: we have many degrees of freedom in choosing curves over a fixed  $\mathbf{F}_q$ , with a consequent diversity of possible cryptographic group orders. Furthermore, an attack which leaves one curve vulnerable may not necessarily apply to other curves. Third, viewing a Jacobian as a geometric object rather than a pure group allows us to take advantage of a number of special features of Jacobians. These features include efficiently computable pairings, geometric transformations for optimised group laws, and the availability of efficiently computable non-integer endomorphisms for accelerated encryption and decryption.

### 3.4. Algebraic Coding Theory

Theme: Coding theory

Coding Theory studies originated with the idea of using redundancy in messages to protect against noise and errors. The last decade of the 20th century has seen the success of so-called iterative decoding methods, which enable us to get very close to the Shannon capacity. The capacity of a given channel is the best achievable transmission rate for reliable transmission. The consensus in the community is that this capacity is more easily reached with these iterative and probabilistic methods than with algebraic codes (such as Reed–Solomon codes).

However, algebraic coding is useful in settings other than the Shannon context. Indeed, the Shannon setting is a random case setting, and promises only a vanishing error probability. In contrast, the algebraic Hamming approach is a worst case approach: under combinatorial restrictions on the noise, the noise can be adversarial, with strictly zero errors.

These considerations are renewed by the topic of list decoding after the breakthrough of Guruswami and Sudan at the end of the nineties. List decoding relaxes the uniqueness requirement of decoding, allowing a small list of candidates to be returned instead of a single codeword. List decoding can reach a capacity close to the Shannon capacity, with zero failure, with small lists, in the adversarial case. The method of

Guruswami and Sudan enabled list decoding of most of the main algebraic codes: Reed–Solomon codes and Algebraic–Geometry (AG) codes and new related constructions “capacity-achieving list decodable codes”. These results open the way to applications against adversarial channels, which correspond to worst case settings in the classical computer science language.

Another avenue of our studies is AG codes over various geometric objects. Although Reed–Solomon codes are the best possible codes for a given alphabet, they are very limited in their length, which cannot exceed the size of the alphabet. AG codes circumvent this limitation, using the theory of algebraic curves over finite fields to construct long codes over a fixed alphabet. The striking result of Tsfasman–Vladut–Zink showed that codes better than random codes can be built this way, for medium to large alphabets. Disregarding the asymptotic aspects and considering only finite length, AG codes can be used either for longer codes with the same alphabet, or for codes with the same length with a smaller alphabet (and thus faster underlying arithmetic).

From a broader point of view, wherever Reed–Solomon codes are used, we can substitute AG codes with some benefits: either beating random constructions, or beating Reed–Solomon codes which are of bounded length for a given alphabet.

Another area of Algebraic Coding Theory with which we are more recently concerned is the one of Locally Decodable Codes. After having been first theoretically introduced, those codes now begin to find practical applications, most notably in cloud-based remote storage systems.

## LFANT Project-Team

### 3. Research Program

#### 3.1. Number fields, class groups and other invariants

**Participants:** Bill Allombert, Karim Belabas, Cyril Bouvier, Jean-Paul Cerri, Iuliana Ciocanea-Teodorescu, Henri Cohen, Jean-Marc Couveignes, Andreas Enge, Fredrik Johansson, Pınar Kılıçer.

Modern number theory has been introduced in the second half of the 19th century by Dedekind, Kummer, Kronecker, Weber and others, motivated by Fermat’s conjecture: There is no non-trivial solution in integers to the equation  $x^n + y^n = z^n$  for  $n \geq 3$ . For recent textbooks, see [5]. Kummer’s idea for solving Fermat’s problem was to rewrite the equation as  $(x + y)(x + \zeta y)(x + \zeta^2 y) \cdots (x + \zeta^{n-1} y) = z^n$  for a primitive  $n$ -th root of unity  $\zeta$ , which seems to imply that each factor on the left hand side is an  $n$ -th power, from which a contradiction can be derived.

The solution requires to augment the integers by *algebraic numbers*, that are roots of polynomials in  $\mathbb{Z}[X]$ . For instance,  $\zeta$  is a root of  $X^n - 1$ ,  $\sqrt[3]{2}$  is a root of  $X^3 - 2$  and  $\sqrt[5]{3}$  is a root of  $25X^2 - 3$ . A *number field* consists of the rationals to which have been added finitely many algebraic numbers together with their sums, differences, products and quotients. It turns out that actually one generator suffices, and any number field  $K$  is isomorphic to  $\mathbb{Q}[X]/(f(X))$ , where  $f(X)$  is the minimal polynomial of the generator. Of special interest are *algebraic integers*, “numbers without denominators”, that are roots of a monic polynomial. For instance,  $\zeta$  and  $\sqrt[3]{2}$  are integers, while  $\sqrt[5]{3}$  is not. The *ring of integers* of  $K$  is denoted by  $\mathcal{O}_K$ ; it plays the same role in  $K$  as  $\mathbb{Z}$  in  $\mathbb{Q}$ .

Unfortunately, elements in  $\mathcal{O}_K$  may factor in different ways, which invalidates Kummer’s argumentation. Unique factorisation may be recovered by switching to *ideals*, subsets of  $\mathcal{O}_K$  that are closed under addition and under multiplication by elements of  $\mathcal{O}_K$ . In  $\mathbb{Z}$ , for instance, any ideal is *principal*, that is, generated by one element, so that ideals and numbers are essentially the same. In particular, the unique factorisation of ideals then implies the unique factorisation of numbers. In general, this is not the case, and the *class group*  $\text{Cl}_K$  of ideals of  $\mathcal{O}_K$  modulo principal ideals and its *class number*  $h_K = |\text{Cl}_K|$  measure how far  $\mathcal{O}_K$  is from behaving like  $\mathbb{Z}$ .

Using ideals introduces the additional difficulty of having to deal with *units*, the invertible elements of  $\mathcal{O}_K$ : Even when  $h_K = 1$ , a factorisation of ideals does not immediately yield a factorisation of numbers, since ideal generators are only defined up to units. For instance, the ideal factorisation  $(6) = (2) \cdot (3)$  corresponds to the two factorisations  $6 = 2 \cdot 3$  and  $6 = (-2) \cdot (-3)$ . While in  $\mathbb{Z}$ , the only units are 1 and  $-1$ , the unit structure in general is that of a finitely generated  $\mathbb{Z}$ -module, whose generators are the *fundamental units*. The *regulator*  $R_K$  measures the “size” of the fundamental units as the volume of an associated lattice.

One of the main concerns of algorithmic algebraic number theory is to explicitly compute these invariants ( $\text{Cl}_K$  and  $h_K$ , fundamental units and  $R_K$ ), as well as to provide the data allowing to efficiently compute with numbers and ideals of  $\mathcal{O}_K$ ; see [28] for a recent account.

The *analytic class number formula* links the invariants  $h_K$  and  $R_K$  (unfortunately, only their product) to the  $\zeta$ -function of  $K$ ,  $\zeta_K(s) := \prod_{\mathfrak{p} \text{ prime ideal of } \mathcal{O}_K} (1 - N\mathfrak{p}^{-s})^{-1}$ , which is meaningful when  $\Re(s) > 1$ , but which may be extended to arbitrary complex  $s \neq 1$ . Introducing characters on the class group yields a generalisation of  $\zeta$ - to  $L$ -functions. The *generalised Riemann hypothesis (GRH)*, which remains unproved even over the rationals, states that any such  $L$ -function does not vanish in the right half-plane  $\Re(s) > 1/2$ . The validity of the GRH has a dramatic impact on the performance of number theoretic algorithms. For instance, under GRH, the class group admits a system of generators of polynomial size; without GRH, only exponential bounds are known. Consequently, an algorithm to compute  $\text{Cl}_K$  via generators and relations (currently the only viable practical approach) either has to assume that GRH is true or immediately becomes exponential.

When  $h_K = 1$  the number field  $K$  may be norm-Euclidean, endowing  $\mathcal{O}_K$  with a Euclidean division algorithm. This question leads to the notions of the Euclidean minimum and spectrum of  $K$ , and another task in algorithmic number theory is to compute explicitly this minimum and the upper part of this spectrum, yielding for instance generalised Euclidean gcd algorithms.

### 3.2. Function fields, algebraic curves and cryptology

**Participants:** Karim Belabas, Guilhem Castagnos, Jean-Marc Couveignes, Andreas Enge, Enea Milio, Damien Robert, Emmanouil Tzortzakis.

Algebraic curves over finite fields are used to build the currently most competitive public key cryptosystems. Such a curve is given by a bivariate equation  $\mathcal{C}(X, Y) = 0$  with coefficients in a finite field  $\mathbb{F}_q$ . The main classes of curves that are interesting from a cryptographic perspective are *elliptic curves* of equation  $\mathcal{C} = Y^2 - (X^3 + aX + b)$  and *hyperelliptic curves* of equation  $\mathcal{C} = Y^2 - (X^{2g+1} + \dots)$  with  $g \geq 2$ .

The cryptosystem is implemented in an associated finite abelian group, the *Jacobian*  $\text{Jac}_{\mathcal{C}}$ . Using the language of function fields exhibits a close analogy to the number fields discussed in the previous section. Let  $\mathbb{F}_q(X)$  (the analogue of  $\mathbb{Q}$ ) be the *rational function field* with subring  $\mathbb{F}_q[X]$  (which is principal just as  $\mathbb{Z}$ ). The *function field* of  $\mathcal{C}$  is  $K_{\mathcal{C}} = \mathbb{F}_q(X)[Y]/(\mathcal{C})$ ; it contains the *coordinate ring*  $\mathcal{O}_{\mathcal{C}} = \mathbb{F}_q[X, Y]/(\mathcal{C})$ . Definitions and properties carry over from the number field case  $K/\mathbb{Q}$  to the function field extension  $K_{\mathcal{C}}/\mathbb{F}_q(X)$ . The Jacobian  $\text{Jac}_{\mathcal{C}}$  is the divisor class group of  $K_{\mathcal{C}}$ , which is an extension of (and for the curves used in cryptography usually equals) the ideal class group of  $\mathcal{O}_{\mathcal{C}}$ .

The size of the Jacobian group, the main security parameter of the cryptosystem, is given by an  $L$ -function. The GRH for function fields, which has been proved by Weil, yields the Hasse–Weil bound  $(\sqrt{q} - 1)^{2g} \leq |\text{Jac}_{\mathcal{C}}| \leq (\sqrt{q} + 1)^{2g}$ , or  $|\text{Jac}_{\mathcal{C}}| \approx q^g$ , where the *genus*  $g$  is an invariant of the curve that correlates with the degree of its equation. For instance, the genus of an elliptic curve is 1, that of a hyperelliptic one is  $\frac{\deg_X \mathcal{C} - 1}{2}$ . An important algorithmic question is to compute the exact cardinality of the Jacobian.

The security of the cryptosystem requires more precisely that the *discrete logarithm problem* (DLP) be difficult in the underlying group; that is, given elements  $D_1$  and  $D_2 = xD_1$  of  $\text{Jac}_{\mathcal{C}}$ , it must be difficult to determine  $x$ . Computing  $x$  corresponds in fact to computing  $\text{Jac}_{\mathcal{C}}$  explicitly with an isomorphism to an abstract product of finite cyclic groups; in this sense, the DLP amounts to computing the class group in the function field setting.

For any integer  $n$ , the *Weil pairing*  $e_n$  on  $\mathcal{C}$  is a function that takes as input two elements of order  $n$  of  $\text{Jac}_{\mathcal{C}}$  and maps them into the multiplicative group of a finite field extension  $\mathbb{F}_{q^k}$  with  $k = k(n)$  depending on  $n$ . It is bilinear in both its arguments, which allows to transport the DLP from a curve into a finite field, where it is potentially easier to solve. The *Tate–Lichtenbaum pairing*, that is more difficult to define, but more efficient to implement, has similar properties. From a constructive point of view, the last few years have seen a wealth of cryptosystems with attractive novel properties relying on pairings.

For a random curve, the parameter  $k$  usually becomes so big that the result of a pairing cannot even be output any more. One of the major algorithmic problems related to pairings is thus the construction of curves with a given, smallish  $k$ .

### 3.3. Complex multiplication

**Participants:** Karim Belabas, Henri Cohen, Jean-Marc Couveignes, Andreas Enge, Fredrik Johansson, Chloë Martindale, Enea Milio, Damien Robert.

Complex multiplication provides a link between number fields and algebraic curves; for a concise introduction in the elliptic curve case, see [30], for more background material, [29]. In fact, for most curves  $\mathcal{C}$  over a finite field, the endomorphism ring of  $\text{Jac}_{\mathcal{C}}$ , which determines its  $L$ -function and thus its cardinality, is an order in a special kind of number field  $K$ , called *CM field*. The CM field of an elliptic curve is an imaginary-quadratic field  $\mathbb{Q}(\sqrt{D})$  with  $D < 0$ , that of a hyperelliptic curve of genus  $g$  is an imaginary-quadratic extension of a totally real number field of degree  $g$ . Deuring’s lifting theorem ensures that  $\mathcal{C}$  is the reduction modulo some prime of a curve with the same endomorphism ring, but defined over the *Hilbert class field*  $H_K$  of  $K$ .

Algebraically,  $H_K$  is defined as the maximal unramified abelian extension of  $K$ ; the Galois group of  $H_K/K$  is then precisely the class group  $\text{Cl}_K$ . A number field extension  $H/K$  is called *Galois* if  $H \simeq K[X]/(f)$  and  $H$  contains all complex roots of  $f$ . For instance,  $\mathbb{Q}(\sqrt{2})$  is Galois since it contains not only  $\sqrt{2}$ , but also the second root  $-\sqrt{2}$  of  $X^2 - 2$ , whereas  $\mathbb{Q}(\sqrt[3]{2})$  is not Galois, since it does not contain the root  $e^{2\pi i/3}\sqrt[3]{2}$  of  $X^3 - 2$ . The *Galois group*  $\text{Gal}_{H/K}$  is the group of automorphisms of  $H$  that fix  $K$ ; it permutes the roots of  $f$ . Finally, an *abelian* extension is a Galois extension with abelian Galois group.

Analytically, in the elliptic case  $H_K$  may be obtained by adjoining to  $K$  the *singular value*  $j(\tau)$  for a complex valued, so-called *modular function*  $j$  in some  $\tau \in \mathcal{O}_K$ ; the correspondence between  $\text{Gal}_{H/K}$  and  $\text{Cl}_K$  allows to obtain the different roots of the minimal polynomial  $f$  of  $j(\tau)$  and finally  $f$  itself. A similar, more involved construction can be used for hyperelliptic curves. This direct application of complex multiplication yields algebraic curves whose  $L$ -functions are known beforehand; in particular, it is the only possible way of obtaining ordinary curves for pairing-based cryptosystems.

The same theory can be used to develop algorithms that, given an arbitrary curve over a finite field, compute its  $L$ -function.

A generalisation is provided by *ray class fields*; these are still abelian, but allow for some well-controlled ramification. The tools for explicitly constructing such class fields are similar to those used for Hilbert class fields.



## POLSYS Project-Team

### 3. Research Program

#### 3.1. Introduction

Polynomial system solving is a fundamental problem in Computer Algebra with many applications in cryptography, robotics, biology, error correcting codes, signal theory, .... Among all available methods for solving polynomial systems, computation of Gröbner bases remains one of the most powerful and versatile method since it can be applied in the continuous case (rational coefficients) as well as in the discrete case (finite fields). Gröbner bases are also a building blocks for higher level algorithms who compute real sample points in the solution set of polynomial systems, decide connectivity queries and quantifier elimination over the reals. The major challenge facing the designer or the user of such algorithms is the intrinsic exponential behaviour of the complexity for computing Gröbner bases. The current proposal is an attempt to tackle these issues in a number of different ways: improve the efficiency of the fundamental algorithms (even when the complexity is exponential), develop high performance implementation exploiting parallel computers, and investigate new classes of structured algebraic problems where the complexity drops to polynomial time.

#### 3.2. Fundamental Algorithms and Structured Systems

**Participants:** Jean-Charles Faugère, Mohab Safey El Din, Elias Tsigaridas, Guénaél Renault, Dongming Wang, Jérémy Berthomieu, Thibaut Verron.

Efficient algorithms  $F_4/F_5^0$  for computing the Gröbner basis of a polynomial system rely heavily on a connection with linear algebra. Indeed, these algorithms reduce the Gröbner basis computation to a sequence of Gaussian eliminations on several submatrices of the so-called Macaulay matrix in some degree. Thus, we expect to improve the existing algorithms by

- (i) developing dedicated linear algebra routines performing the Gaussian elimination steps: this is precisely the objective 2 described below;
- (ii) generating smaller or simpler matrices to which we will apply Gaussian elimination.

We describe here our goals for the latter problem. First, we focus on algorithms for computing a Gröbner basis of *general polynomial systems*. Next, we present our goals on the development of dedicated algorithms for computing Gröbner bases of *structured polynomial systems* which arise in various applications.

**Algorithms for general systems.** Several degrees of freedom are available to the designer of a Gröbner basis algorithm to generate the matrices occurring during the computation. For instance, it would be desirable to obtain matrices which would be almost triangular or very sparse. Such a goal can be achieved by considering various interpretations of the  $F_5$  algorithm with respect to different monomial orderings. To address this problem, the tight complexity results obtained for  $F_5$  will be used to help in the design of such a general algorithm. To illustrate this point, consider the important problem of solving boolean polynomial systems; it might be interesting to preserve the sparsity of the original equations and, at the same time, using the fact that overdetermined systems are much easier to solve.

**Algorithms dedicated to structured polynomial systems.** A complementary approach is to exploit the structure of the input polynomials to design specific algorithms. Very often, problems coming from applications are not random but are highly structured. The specific nature of these systems may vary a lot: some polynomial systems can be sparse (when the number of terms in each equation is low), overdetermined (the number of the equations is larger than the number of variables), invariants by the action of some finite groups, multi-linear (each equation is linear w.r.t. to one block of variables) or more generally multihomogeneous. In each case, the ultimate goal is to identify large classes of problems whose theoretical/practical complexity drops and to propose in each case dedicated algorithms.

---

<sup>0</sup>J.-C. Faugère. *A new efficient algorithm for computing Gröbner bases without reduction to zero (F5)*. In Proceedings of ISSAC '02, pages 75-83, New York, NY, USA, 2002. ACM.

### 3.3. Solving Systems over the Reals and Applications.

**Participants:** Mohab Safey El Din, Daniel Lazard, Elias Tsigaridas, Ivan Bannwarth.

We shall develop algorithms for solving polynomial systems over complex/real numbers. Again, the goal is to extend significantly the range of reachable applications using algebraic techniques based on Gröbner bases and dedicated linear algebra routines. Targeted application domains are global optimization problems, stability of dynamical systems (e.g. arising in biology or in control theory) and theorem proving in computational geometry.

The following functionalities shall be requested by the end-users:

- (i) deciding the emptiness of the real solution set of systems of polynomial equations and inequalities,
- (ii) quantifier elimination over the reals or complex numbers,
- (iii) answering connectivity queries for such real solution sets.

We will focus on these functionalities.

We will develop algorithms based on the so-called critical point method to tackle systems of equations and inequalities (problem (i)). These techniques are based on solving 0-dimensional polynomial systems encoding "critical points" which are defined by the vanishing of minors of jacobian matrices (with polynomial entries). Since these systems are highly structured, the expected results of Objective 1 and 2 may allow us to obtain dramatic improvements in the computation of Gröbner bases of such polynomial systems. This will be the foundation of practically fast implementations (based on singly exponential algorithms) outperforming the current ones based on the historical Cylindrical Algebraic Decomposition (CAD) algorithm (whose complexity is doubly exponential in the number of variables). We will also develop algorithms and implementations that allow us to analyze, at least locally, the topology of solution sets in some specific situations. A long-term goal is obviously to obtain an analysis of the global topology.

### 3.4. Low level implementation and Dedicated Algebraic Computation and Linear Algebra.

**Participants:** Jean-Charles Faugère, Christian Eder, Elias Tsigaridas.

Here, the primary objective is to focus on *dedicated* algorithms and software for the linear algebra steps in Gröbner bases computations and for problems arising in Number Theory. As explained above, linear algebra is a key step in the process of computing efficiently Gröbner bases. It is then natural to develop specific linear algebra algorithms and implementations to further strengthen the existing software. Conversely, Gröbner bases computation is often a key ingredient in higher level algorithms from Algebraic Number Theory. In these cases, the algebraic problems are very particular and specific. Hence dedicated Gröbner bases algorithms and implementations would provide a better efficiency.

**Dedicated linear algebra tools.** FGB is an efficient library for Gröbner bases computations which can be used, for instance, via MAPLE. However, the library is sequential. A goal of the project is to extend its efficiency to new trend parallel architectures such as clusters of multi-processor systems in order to tackle a broader class of problems for several applications. Consequently, our first aim is to provide a durable, long term software solution, which will be the successor of the existing FGB library. To achieve this goal, we will first develop a high performance linear algebra package (under the LGPL license). This could be organized in the form of a collaborative project between the members of the team. The objective is not to develop a general library similar to the LINBOX project but to propose a dedicated linear algebra package taking into account the specific properties of the matrices generated by the Gröbner bases algorithms. Indeed these matrices are sparse (the actual sparsity depends strongly on the application), almost block triangular and not necessarily of full rank. Moreover, most of the pivots are known at the beginning of the computation. In practice, such matrices are huge (more than  $10^6$  columns) but taking into account their shape may allow us to speed up the computations by one or several orders of magnitude. A variant of a Gaussian elimination algorithm together with a corresponding C implementation has been presented. The main peculiarity is the order in which the operations are performed. This will be the kernel of the new linear algebra library that will be developed.

Fast linear algebra packages would also benefit to the transformation of a Gröbner basis of a zero-dimensional ideal with respect to a given monomial ordering into a Gröbner basis with respect to another ordering. In the generic case at least, the change of ordering is equivalent to the computation of the minimal polynomial of a so-called multiplication matrix. By taking into account the sparsity of this matrix, the computation of the Gröbner basis can be done more efficiently using a variant of the Wiedemann algorithm. Hence, our goal is also to obtain a dedicated high performance library for transforming (i.e. change ordering) Gröbner bases.

**Dedicated algebraic tools for Algebraic Number Theory.** Recent results in Algebraic Number Theory tend to show that the computation of Gröbner basis is a key step toward the resolution of difficult problems in this domain<sup>0</sup>. Using existing resolution methods is simply not enough to solve relevant problems. The main algorithmic bottleneck to overcome is to adapt the Gröbner basis computation step to the specific problems. Typically, problems coming from Algebraic Number Theory usually have a lot of symmetries or the input systems are very structured. This is the case in particular for problems coming from the algorithmic theory of Abelian varieties over finite fields<sup>0</sup> where the objects are represented by polynomial system and are endowed with intrinsic group actions. The main goal here is to provide dedicated algebraic resolution algorithms and implementations for solving such problems. We do not restrict our focus on problems in positive characteristic. For instance, tower of algebraic fields can be viewed as triangular sets; more generally, related problems (e.g. effective Galois theory) which can be represented by polynomial systems will receive our attention. This is motivated by the fact that, for example, computing small integer solutions of Diophantine polynomial systems in connection with Coppersmith's method would also gain in efficiency by using a dedicated Gröbner bases computations step.

### 3.5. Solving Systems in Finite Fields, Applications in Cryptology and Algebraic Number Theory.

**Participants:** Jean-Charles Faugère, Ludovic Perret, Guénaél Renault, Jérémy Berthomieu.

Here, we focus on solving polynomial systems over finite fields (i.e. the discrete case) and the corresponding applications (Cryptology, Error Correcting Codes, ...). Obviously this objective can be seen as an application of the results of the two previous objectives. However, we would like to emphasize that it is also the source of new theoretical problems and practical challenges. We propose to develop a systematic use of *structured systems* in *algebraic cryptanalysis*.

(i) So far, breaking a cryptosystem using algebraic techniques could be summarized as modeling the problem by algebraic equations and then computing a, usually, time consuming Gröbner basis. A new trend in this field is to require a theoretical complexity analysis. This is needed to explain the behavior of the attack but also to help the designers of new cryptosystems to propose actual secure parameters.

(ii) To assess the security of several cryptosystems in symmetric cryptography (block ciphers, hash functions, ...), a major difficulty is the size of the systems involved for this type of attack. More specifically, the bottleneck is the size of the linear algebra problems generated during a Gröbner basis computation.

We propose to develop a systematic use of *structured systems* in *algebraic cryptanalysis*.

The first objective is to build on the recent breakthrough in attacking McEliece's cryptosystem: it is the first structural weakness observed on one of the oldest public key cryptosystem. We plan to develop a well founded framework for assessing the security of public key cryptosystems based on coding theory from the algebraic cryptanalysis point of view. The answer to this issue is strongly related to the complexity of solving bihomogeneous systems (of bidegree  $(1, d)$ ). We also plan to use the recently gained understanding on the complexity of structured systems in other areas of cryptography. For instance, the MinRank problem – which can be modeled as an overdetermined system of bilinear equations – is at the heart of the structural attack proposed by Kipnis and Shamir against HFE (one of the most well known multivariate public cryptosystem). The same family of structured systems arises in the algebraic cryptanalysis of the Discrete Logarithmic

<sup>0</sup> P. Gaudry, *Index calculus for abelian varieties of small dimension and the elliptic curve discrete logarithm problem*, Journal of Symbolic Computation 44,12 (2009) pp. 1690-1702

<sup>0</sup> e.g. point counting, discrete logarithm, isogeny.

Problem (DLP) over curves (defined over some finite fields). More precisely, some bilinear systems appear in the polynomial modeling the points decomposition problem. Moreover, in this context, a natural group action can also be used during the resolution of the considered polynomial system.

Dedicated tools for linear algebra problems generated during the Gröbner basis computation will be used in algebraic cryptanalysis. The promise of considerable algebraic computing power beyond the capability of any standard computer algebra system will enable us to attack various cryptosystems or at least to propose accurate secure parameters for several important cryptosystems. Dedicated linear tools are thus needed to tackle these problems. From a theoretical perspective, we plan to further improve the theoretical complexity of the hybrid method and to investigate the problem of solving polynomial systems with noise, i.e. some equations of the system are incorrect. The hybrid method is a specific method for solving polynomial systems over finite fields. The idea is to mix exhaustive search and Gröbner basis computation to take advantage of the over-determinacy of the resulting systems.

Polynomial system with noise is currently emerging as a problem of major interest in cryptography. This problem is a key to further develop new applications of algebraic techniques; typically in side-channel and statistical attacks. We also emphasize that recently a connection has been established between several classical lattice problems (such as the Shortest Vector Problem), polynomial system solving and polynomial systems with noise. The main issue is that there is no sound algorithmic and theoretical framework for solving polynomial systems with noise. The development of such framework is a long-term objective.

## **SECRET Project-Team**

### **3. Research Program**

#### **3.1. Scientific foundations**

Our approach relies on a competence whose impact is much wider than cryptology. Our tools come from information theory, discrete mathematics, probabilities, algorithmics, quantum physics... Most of our work mixes fundamental aspects (study of mathematical objects) and practical aspects (cryptanalysis, design of algorithms, implementations). Our research is mainly driven by the belief that discrete mathematics and algorithmics of finite structures form the scientific core of (algorithmic) data protection.

#### **3.2. Symmetric cryptology**

Symmetric techniques are widely used because they are the only ones that can achieve some major features such as high-speed or low-cost encryption, fast authentication, and efficient hashing. It is a very active research area which is stimulated by a pressing industrial demand. The process which has led to the new block cipher standard AES in 2001 was the outcome of a decade of research in symmetric cryptography, where new attacks have been proposed, analyzed and then thwarted by some appropriate designs. However, even if its security has not been challenged so far, it clearly appears that the AES cannot serve as a Swiss knife in all environments. In particular an important challenge raised by several new applications is the design of symmetric encryption schemes with some additional properties compared to the AES, either in terms of implementation performance (low-cost hardware implementation, low latency, resistance against side-channel attacks...) or in terms of functionalities (like authenticated encryption). The past decade has then been characterized by a multiplicity of new proposals. This proliferation of symmetric primitives has been amplified by several public competitions (eSTREAM, SHA-3, CAESAR...) which have encouraged innovative constructions and promising but unconventional designs. We are then facing up to a very new situation where implementers need to make informed choices among more than 40 lightweight block ciphers<sup>0</sup> or 57 new authenticated-encryption schemes<sup>0</sup>. Evaluating the security of all these proposals has then become a primordial task which requires the attention of the community.

In this context we believe that the cryptanalysis effort cannot scale up without an in-depth study of the involved algorithms. Indeed most attacks are described as ad-hoc techniques dedicated to a particular cipher. To determine whether they apply to some other primitives, it is then crucial to formalize them in a general setting. Our approach relies on the idea that a unified description of generic attacks (in the sense that they apply to a large class of primitives) is the only methodology for a precise evaluation of the resistance of all these new proposals, and of their security margins. In particular, such a work prevents misleading analyses based on wrong estimations of the complexity or on non-optimized algorithms. It also provides security criteria which enable designers to guarantee that their primitive resists some families of attacks. The main challenge is to provide a generic description which captures most possible optimizations of the attack.

#### **3.3. Code-based cryptography**

Public-key cryptography is one of the key tools for providing network security (SSL, e-commerce, e-banking...). The security of nearly all public-key schemes used today relies on the presumed difficulty of two problems, namely factorization of large integers or computing the discrete logarithm over various groups. The hardness of those problems was questioned in 1994<sup>0</sup> when Shor showed that a quantum computer could solve them efficiently. Though large enough quantum computers that would be able to threaten the

---

<sup>0</sup>35 are described on [https://www.cryptolux.org/index.php/Lightweight\\_Block\\_Ciphers](https://www.cryptolux.org/index.php/Lightweight_Block_Ciphers).

<sup>0</sup>see <http://competitions.cr.yt.to/caesar-submissions.html>

<sup>0</sup>P. Shor, *Algorithms for quantum computation: Discrete logarithms and factoring*, FOCS 1994.

existing cryptosystems do not exist yet, the cryptographic research community has to get ready and has to prepare alternatives. This line of work is usually referred to as *post-quantum cryptography*. This has become a prominent research field. Most notably, an international call for post-quantum primitives<sup>0</sup> has been launched by the NIST very recently, with a submission deadline in November 2017.

The research of the project-team in this field is focused on the design and cryptanalysis of cryptosystems making use of coding theory. Code-based cryptography is one the main techniques for post-quantum cryptography (together with lattice-based, multivariate, or hash-based cryptography).

### 3.4. Quantum information

The field of quantum information and computation aims at exploiting the laws of quantum physics to manipulate information in radically novel ways. There are two main applications:

- (i) quantum computing, that offers the promise of solving some problems that seem to be intractable for classical computers such as for instance factorization or solving the discrete logarithm problem;
- (ii) quantum cryptography, which provides new ways to exchange data in a provably secure fashion. For instance it allows key distribution by using an authenticated channel and quantum communication over an unreliable channel with unconditional security, in the sense that its security can be proven rigorously by using only the laws of quantum physics, even with all-powerful adversaries.

Our team deals with quantum coding theoretic issues related to building a large quantum computer and with quantum cryptography. The first part builds upon our expertise in classical coding theory whereas the second axis focuses on obtaining security proofs for quantum protocols or on devising quantum cryptographic protocols (and more generally quantum protocols related to cryptography). A close relationship with partners working in the whole area of quantum information processing in the Parisian region has also been developed through our participation to the Fédération de Recherche “PCQC” (Paris Centre for Quantum Computing).

---

<sup>0</sup><http://csrc.nist.gov/groups/ST/post-quantum-crypto/>

## SPECFUN Project-Team

### 3. Research Program

#### 3.1. Studying special functions by computer algebra

Computer algebra manipulates symbolic representations of exact mathematical objects in a computer, in order to perform computations and operations like simplifying expressions and solving equations for “closed-form expressions”. The manipulations are often fundamentally of algebraic nature, even when the ultimate goal is analytic. The issue of efficiency is a particular one in computer algebra, owing to the extreme swell of the intermediate values during calculations.

Our view on the domain is that research on the algorithmic manipulation of special functions is anchored between two paradigms:

- adopting linear differential equations as the right data structure for special functions,
- designing efficient algorithms in a complexity-driven way.

It aims at four kinds of algorithmic goals:

- algorithms combining functions,
- functional equations solving,
- multi-precision numerical evaluations,
- guessing heuristics.

This interacts with three domains of research:

- computer algebra, meant as the search for quasi-optimal algorithms for exact algebraic objects,
- symbolic analysis/algebraic analysis;
- experimental mathematics (combinatorics, mathematical physics, ...).

This view is made explicit in the present section.

##### 3.1.1. Equations as a data structure

Numerous special functions satisfy linear differential and/or recurrence equations. Under a mild technical condition, the existence of such equations induces a finiteness property that makes the main properties of the functions decidable. We thus speak of *D-finite functions*. For example, 60 % of the chapters in the handbook [21] describe D-finite functions. In addition, the class is closed under a rich set of algebraic operations. This makes linear functional equations just the right data structure to encode and manipulate special functions. The power of this representation was observed in the early 1990s [74], leading to the design of many algorithms in computer algebra. Both on the theoretical and algorithmic sides, the study of D-finite functions shares much with neighbouring mathematical domains: differential algebra, D-module theory, differential Galois theory, as well as their counterparts for recurrence equations.

##### 3.1.2. Algorithms combining functions

Differential/recurrence equations that define special functions can be recombined [74] to define: additions and products of special functions; compositions of special functions; integrals and sums involving special functions. Zeilberger’s fast algorithm for obtaining recurrences satisfied by parametrised binomial sums was developed in the early 1990s already [75]. It is the basis of all modern definite summation and integration algorithms. The theory was made fully rigorous and algorithmic in later works, mostly by a group in RISC (Linz, Austria) and by members of the team [63], [71], [39], [37], [38], [57]. The past ÉPI Algorithms contributed several implementations (*gfun* [66], *Mgfun* [39]).

### 3.1.3. Solving functional equations

Encoding special functions as defining linear functional equations postpones some of the difficulty of the problems to a delayed solving of equations. But at the same time, solving (for special classes of functions) is a sub-task of many algorithms on special functions, especially so when solving in terms of polynomial or rational functions. A lot of work has been done in this direction in the 1990s; more intensively since the 2000s, solving differential and recurrence equations in terms of special functions has also been investigated.

### 3.1.4. Multi-precision numerical evaluation

A major conceptual and algorithmic difference exists for numerical calculations between data structures that fit on a machine word and data structures of arbitrary length, that is, *multi-precision* arithmetic. When multi-precision floating-point numbers became available, early works on the evaluation of special functions were just promising that “most” digits in the output were correct, and performed by heuristically increasing precision during intermediate calculations, without intended rigour. The original theory has evolved in a twofold way since the 1990s: by making computable all constants hidden in asymptotic approximations, it became possible to guarantee a *prescribed* absolute precision; by employing state-of-the-art algorithms on polynomials, matrices, etc, it became possible to have evaluation algorithms in a time complexity that is linear in the output size, with a constant that is not more than a few units. On the implementation side, several original works exist, one of which (*NumGfun* [62]) is used in our DDMF.

### 3.1.5. Guessing heuristics

“Differential approximation”, or “Guessing”, is an operation to get an ODE likely to be satisfied by a given approximate series expansion of an unknown function. This has been used at least since the 1970s and is a key stone in spectacular applications in experimental mathematics [36]. All this is based on subtle algorithms for Hermite–Padé approximants [25]. Moreover, guessing can at times be complemented by proven quantitative results that turn the heuristics into an algorithm [33]. This is a promising algorithmic approach that deserves more attention than it has received so far.

### 3.1.6. Complexity-driven design of algorithms

The main concern of computer algebra has long been to prove the feasibility of a given problem, that is, to show the existence of an algorithmic solution for it. However, with the advent of faster and faster computers, complexity results have ceased to be of theoretical interest only. Nowadays, a large track of works in computer algebra is interested in developing fast algorithms, with time complexity as close as possible to linear in their output size. After most of the more pervasive objects like integers, polynomials, and matrices have been endowed with fast algorithms for the main operations on them [44], the community, including ourselves, started to turn its attention to differential and recurrence objects in the 2000s. The subject is still not as developed as in the commutative case, and a major challenge remains to understand the combinatorics behind summation and integration. On the methodological side, several paradigms occur repeatedly in fast algorithms: “divide and conquer” to balance calculations, “evaluation and interpolation” to avoid intermediate swell of data, etc. [30].

## 3.2. Trusted computer-algebra calculations

### 3.2.1. Encyclopedias

Handbooks collecting mathematical properties aim at serving as reference, therefore trusted, documents. The decision of several authors or maintainers of such knowledge bases to move from paper books [21], [23], [67] to websites and wikis<sup>0</sup> allows for a more collaborative effort in proof reading. Another step toward further confidence is to manage to generate the content of an encyclopedia by computer-algebra programs, as is the case with the Wolfram Functions Site<sup>0</sup> or DDMF<sup>0</sup>. Yet, due to the lingering doubts about computer-algebra systems, some encyclopedias propose both cross-checking by different systems and handwritten companion paper proofs of their content<sup>0</sup>. As of today, there is no encyclopedia certified with formal proofs.

<sup>0</sup>for instance <http://dlmf.nist.gov/> for special functions or <http://oeis.org/> for integer sequences

<sup>0</sup><http://functions.wolfram.com/>

<sup>0</sup><http://ddmf.msr-inria.inria.fr/1.9.1/ddmf>



### ***3.2.2. Computer algebra and symbolic logic***

Several attempts have been made in order to extend existing computer-algebra systems with symbolic manipulations of logical formulas. Yet, these works are more about extending the expressivity of computer-algebra systems than about improving the standards of correctness and semantics of the systems. Conversely, several projects have addressed the communication of a proof system with a computer-algebra system, resulting in an increased automation available in the proof system, to the price of the uncertainty of the computations performed by this oracle.

### ***3.2.3. Certifying systems for computer algebra***

More ambitious projects have tried to design a new computer-algebra system providing an environment where the user could both program efficiently and elaborate formal and machine-checked proofs of correctness, by calling a general-purpose proof assistant like the Coq system. This approach requires a huge manpower and a daunting effort in order to re-implement a complete computer-algebra system, as well as the libraries of formal mathematics required by such formal proofs.

### ***3.2.4. Semantics for computer algebra***

The move to machine-checked proofs of the mathematical correctness of the output of computer-algebra implementations demands a prior clarification about the often implicit assumptions on which the presumably correctly implemented algorithms rely. Interestingly, this preliminary work, which could be considered as independent from a formal certification project, is seldom precise or even available in the literature.

### ***3.2.5. Formal proofs for symbolic components of computer-algebra systems***

A number of authors have investigated ways to organize the communication of a chosen computer-algebra system with a chosen proof assistant in order to certify specific components of the computer-algebra systems, experimenting various combinations of systems and various formats for mathematical exchanges. Another line of research consists in the implementation and certification of computer-algebra algorithms inside the logic [70], [49], [59] or as a proof-automation strategy. Normalization algorithms are of special interest when they allow to check results possibly obtained by an external computer-algebra oracle [42]. A discussion about the systematic separation of the search for a solution and the checking of the solution is already clearly outlined in [55].

### ***3.2.6. Formal proofs for numerical components of computer-algebra systems***

Significant progress has been made in the certification of numerical applications by formal proofs. Libraries formalizing and implementing floating-point arithmetic as well as large numbers and arbitrary-precision arithmetic are available. These libraries are used to certify floating-point programs, implementations of mathematical functions and for applications like hybrid systems.

## **3.3. Machine-checked proofs of formalized mathematics**

To be checked by a machine, a proof needs to be expressed in a constrained, relatively simple formal language. Proof assistants provide facilities to write proofs in such languages. But, as merely writing, even in a formal language, does not constitute a formal proof just per se, proof assistants also provide a proof checker: a small and well-understood piece of software in charge of verifying the correctness of arbitrarily large proofs. The gap between the low-level formal language a machine can check and the sophistication of an average page of mathematics is conspicuous and unavoidable. Proof assistants try to bridge this gap by offering facilities, like notations or automation, to support convenient formalization methodologies. Indeed, many aspects, from the logical foundation to the user interface, play an important role in the feasibility of formalized mathematics inside a proof assistant.

---

<sup>0</sup><http://129.81.170.14/~vhm/Table.html>

### 3.3.1. Logical foundations and proof assistants

While many logical foundations for mathematics have been proposed, studied, and implemented, type theory is the one that has been more successfully employed to formalize mathematics, to the notable exception of the Mizar system [60], which is based on set theory. In particular, the calculus of construction (CoC) [40] and its extension with inductive types (CIC) [41], have been studied for more than 20 years and been implemented by several independent tools (like Lego, Matita, and Agda). Its reference implementation, Coq [68], has been used for several large-scale formalizations projects (formal certification of a compiler back-end; four-color theorem). Improving the type theory underlying the Coq system remains an active area of research. Other systems based on different type theories do exist and, whilst being more oriented toward software verification, have been also used to verify results of mainstream mathematics (prime-number theorem; Kepler conjecture).

### 3.3.2. Computations in formal proofs

The most distinguishing feature of CoC is that computation is promoted to the status of rigorous logical argument. Moreover, in its extension CIC, we can recognize the key ingredients of a functional programming language like inductive types, pattern matching, and recursive functions. Indeed, one can program effectively inside tools based on CIC like Coq. This possibility has paved the way to many effective formalization techniques that were essential to the most impressive formalizations made in CIC.

Another milestone in the promotion of the computations-as-proofs feature of Coq has been the integration of compilation techniques in the system to speed up evaluation. Coq can now run realistic programs in the logic, and hence easily incorporates calculations into proofs that demand heavy computational steps.

Because of their different choice for the underlying logic, other proof assistants have to simulate computations outside the formal system, and indeed fewer attempts to formalize mathematical proofs involving heavy calculations have been made in these tools. The only notable exception, which was finished in 2014, the Kepler conjecture, required a significant work to optimize the rewriting engine that simulates evaluation in Isabelle/HOL.

### 3.3.3. Large-scale computations for proofs inside the Coq system

Programs run and proved correct inside the logic are especially useful for the conception of automated decision procedures. To this end, inductive types are used as an internal language for the description of mathematical objects by their syntax, thus enabling programs to reason and compute by case analysis and recursion on symbolic expressions.

The output of complex and optimized programs external to the proof assistant can also be stamped with a formal proof of correctness when their result is easier to *check* than to *find*. In that case one can benefit from their efficiency without compromising the level of confidence on their output at the price of writing and certify a checker inside the logic. This approach, which has been successfully used in various contexts, is very relevant to the present research project.

### 3.3.4. Relevant contributions from the Mathematical Component libraries

Representing abstract algebra in a proof assistant has been studied for long. The libraries developed by the MathComp project for the proof of the Odd Order Theorem provide a rather comprehensive hierarchy of structures; however, they originally feature a large number of instances of structures that they need to organize. On the methodological side, this hierarchy is an incarnation of an original work [43] based on various mechanisms, primarily type inference, typically employed in the area of programming languages. A large amount of information that is implicit in handwritten proofs, and that must become explicit at formalization time, can be systematically recovered following this methodology.

Small-scale reflection [46] is another methodology promoted by the MathComp project. Its ultimate goal is to ease formal proofs by systematically dealing with as many bureaucratic steps as possible, by automated computation. For instance, as opposed to the style advocated by Coq's standard library, decidable predicates are systematically represented using computable boolean functions: comparison on integers is expressed as

program, and to state that  $a \leq b$  one compares the output of this program run on  $a$  and  $b$  with *true*. In many cases, for example when  $a$  and  $b$  are values, one can prove or disprove the inequality by pure computation.

The MathComp library was consistently designed after uniform principles of software engineering. These principles range from simple ones, like naming conventions, to more advanced ones, like generic programming, resulting in a robust and reusable collection of formal mathematical components. This large body of formalized mathematics covers a broad panel of algebraic theories, including of course advanced topics of finite group theory, but also linear algebra, commutative algebra, Galois theory, and representation theory. We refer the interested reader to the online documentation of these libraries [69], which represent about 150,000 lines of code and include roughly 4,000 definitions and 13,000 theorems.

Topics not addressed by these libraries and that might be relevant to the present project include real analysis and differential equations. The most advanced work of formalization on these domains is available in the HOL-Light system [51], [52], [53], although some existing developments of interest [28], [61] are also available for Coq. Another aspect of the MathComp libraries that needs improvement, owing to the size of the data we manipulate, is the connection with efficient data structures and implementations, which only starts to be explored.

### 3.3.5. *User interaction with the proof assistant*

The user of a proof assistant describes the proof he wants to formalize in the system using a textual language. Depending on the peculiarities of the formal system and the applicative domain, different proof languages have been developed. Some proof assistants promote the use of a declarative language, when the Coq and Matita systems are more oriented toward a procedural style.

The development of the large, consistent body of MathComp libraries has prompted the need to design an alternative and coherent language extension for the Coq proof assistant [48], [47], enforcing the robustness of proof scripts to the numerous changes induced by code refactoring and enhancing the support for the methodology of small-scale reflection.

The development of large libraries is quite a novelty for the Coq system. In particular any long-term development process requires the iteration of many refactoring steps and very little support is provided by most proof assistants, with the notable exception of Mizar [65]. For the Coq system, this is an active area of research.

**VEGAS Project-Team (section vide)**

## CAIRN Project-Team

### 3. Research Program

#### 3.1. Panorama

The development of complex applications is traditionally split in three stages: a theoretical study of the algorithms, an analysis of the target architecture and the implementation. When facing new emerging applications such as high-performance, low-power and low-cost mobile communication systems or smart sensor-based systems, it is mandatory to strengthen the design flow by a joint study of both algorithmic and architectural issues.

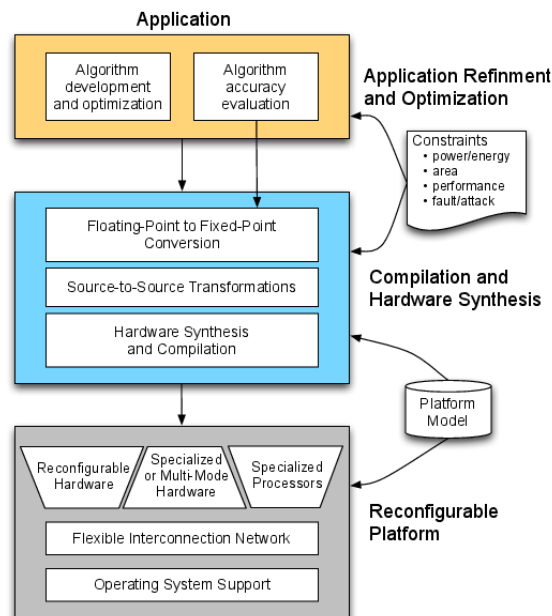


Figure 1. CAIRN's general design flow and related research themes

Figure 1 shows the global design flow we propose to develop. This flow is organized in levels which refer to our three research themes: application optimization (new algorithms, fixed-point arithmetic, advanced representations of numbers), architecture optimization (reconfigurable and specialized hardware, application-specific processors, arithmetic operators and functions), and stepwise refinement and code generation (code transformations, hardware synthesis, compilation).

In the rest of this part, we briefly describe the challenges concerning **new reconfigurable platforms** in Section 3.2 and the issues on **compiler and synthesis tools** related to these platforms in Section 3.3 .

## 3.2. Reconfigurable Architecture Design

Nowadays, FPGAs are not only suited for application specific algorithms, but also considered as fully-featured computing platforms, thanks to their ability to accelerate massively parallelizable algorithms much faster than their processor counterparts [84]. They also support to be dynamically reconfigured. At runtime, partially reconfigurable regions of the logic fabric can be reconfigured to implement a different task, which allows for a better resource usage and adaptation to the environment. Dynamically reconfigurable hardware can also cope with hardware errors by relocating some of its functionalities to another, sane, part of the logic fabric. It could also provide support for a multi-tasked computation flow where hardware tasks are loaded on-demand at runtime. Nevertheless, current design flows of FPGA vendors are still limited by the use of one partial bitstream for each reconfigurable region and for each design. These regions are defined at design time and it is not possible to use only one bitstream for multiple reconfigurable regions nor multiple chips. The multiplicity of such bitstreams leads to a significant increase in memory. Recent research has been conducted in the domain of task relocation on a reconfigurable fabric. All of the related work was conducted on architectures from commercial vendors (e.g., Xilinx, Altera) which share the same limitations: the inner details of the bitstream are not publicly known, which limits applicability of the techniques. To circumvent this issue, most dynamic reconfiguration techniques are either generating multiple bitstreams for each location [66] or implementing an online filter to relocate the tasks [78]. Both of these techniques still suffer from memory footprint and from the online complexity of task relocation.

Increasing the level and grain of reconfiguration is a solution to counterbalance the FPGA penalties. Coarse-grained reconfigurable architectures (CGRA) provide operator-level configurable functional blocks and word-level datapaths [85], [72], [83]. Compared to FPGA, they benefit from a massive reduction in configuration memory and configuration delay, as well as for routing and placement complexity. This in turns results in an improvement in the computation volume over energy cost ratio, although with a loss of flexibility compared to bit-level operations. Such constraints have been taken into account in the design of DART[10], Adres [81] or polymorphous computing fabrics[12]. These works have led to commercial products such as the PACT/XPP [65] or Montium from Recore systems, without however a real commercial success yet. Emerging platforms like Xilinx/Zynq or Intel/Altera are about to change the game.

In the context of emerging heterogenous multicore architecture, CAIRN advocates for associating general-purpose processors (GPP), flexible network-on-chip and coarse-grain or fine-grain dynamically reconfigurable accelerators. We leverage our skills on microarchitecture, reconfigurable computing, arithmetic, and low-power design, to discover and design such architectures with a focus on: -reduced energy per operation, - improved application performance through acceleration, - hardware flexibility and self-adaptive behavior, - tolerance to faults, computing errors, and process variation, - protections against side channel attacks, - limited silicon area overhead.

## 3.3. Compilation and Synthesis for Reconfigurable Platforms

In spite of their advantages, reconfigurable architectures, and more generally hardware accelerators, lack efficient and standardized compilation and design tools. As of today, this still makes the technology impractical for large-scale industrial use. Generating and optimizing the mapping from high-level specifications to reconfigurable hardware platforms are therefore key research issues, which have received considerable interest over the last years [70], [86], [82], [80], [79]. In the meantime, the complexity (and heterogeneity) of these platforms has also been increasing quite significantly, with complex heterogeneous multi-cores architectures becoming a *de facto* standard. As a consequence, the focus of designers is now geared toward optimizing overall system-level performance and efficiency [77]. Here again, existing tools are not well suited, as they fail at providing an unified programming view of the programmable and/or reconfigurable components implemented on the platform.

In this context, we have been pursuing our efforts to propose tools whose design principles are based on a tight coupling between the compiler and the target hardware architectures. We build on the expertise of the team members in High Level Synthesis (HLS) [6], ASIP optimizing compilers [13] and automatic parallelization for massively parallel specialized circuits [2]. We first study how to increase the efficiency of standard programmable processors by extending their instruction set to speed-up compute intensive kernels. Our focus is on efficient and exact algorithms for the identification, selection and scheduling of such instructions [7]. We address compilation challenges by borrowing techniques from high-level synthesis, optimizing compilers and automatic parallelization, especially when dealing with nested loop kernels. In addition, and independently of the scientific challenges mentioned above, proposing such flows also poses significant software engineering issues. As a consequence, we also study how leading edge software engineering techniques (Model Driven Engineering) can help the Computer Aided Design (CAD) and optimizing compiler communities prototyping new research ideas [14], [5], [3].

Efficient implementation of multimedia and signal processing applications (in software for DSP cores or as special-purpose hardware) often requires, for reasons related to cost, power consumption or silicon area constraints, the use of fixed-point arithmetic, whereas the algorithms are usually specified in floating-point arithmetic. Unfortunately, fixed-point conversion is very challenging and time-consuming, typically demanding up to 50% of the total design or implementation time. Thus, tools are required to automate this conversion. For hardware or software implementation, the aim is to optimize the fixed-point specification. The implementation cost is minimized under a numerical accuracy or an application performance constraint. For DSP-software implementation, methodologies have been proposed [8] to achieve fixed-point conversion. For hardware implementation, the best results are obtained when the word-length optimization process is coupled with the high-level synthesis [73]. Evaluating the effects of finite precision is one of the major and often the most time consuming step while performing fixed-point refinement. Indeed, in the word-length optimization process, the numerical accuracy is evaluated as soon as a new word-length is tested, thus, several times per iteration of the optimization process. Classical approaches are based on fixed-point simulations [74]. Leading to long evaluation times, they can hardly be used to explore the design space. Therefore, our aim is to propose closed-form expressions of errors due to fixed-point approximations that are used by a fast analytical framework for accuracy evaluation [11].

## CAMUS Team

### 3. Research Program

#### 3.1. Research Directions

The various objectives we are expecting to reach are directly related to the search of adequacy between the software and the new multicore processors evolution. They also correspond to the main research directions suggested by Hall, Padua and Pingali in [24]. Performance, correction and productivity must be the users' perceived effects. They will be the consequences of research works dealing with the following issues:

- Issue 1: Static Parallelization and Optimization
- Issue 2: Profiling and Execution Behavior Modeling
- Issue 3: Dynamic Program Parallelization and Optimization, Virtual Machine
- Issue 4: Proof of Program Transformations for Multicores

Efficient and correct applications development for multicore processors needs stepping in every application development phase, from the initial conception to the final run.

Upstream, all potential parallelism of the application has to be exhibited. Here static analysis and transformation approaches (issue 1) must be processed, resulting in a *multi-parallel* intermediate code advising the running virtual machine about all the parallelism that can be taken advantage of. However the compiler does not have much knowledge about the execution environment. It obviously knows the instruction set, it can be aware of the number of available cores, but it does not know the effective available resources at any time during the execution (memory, number of free cores, etc.).

That is the reason why a “virtual machine” mechanism will have to adapt the application to the resources (issue 3). Moreover the compiler will be able to take advantage only of a part of the parallelism induced by the application. Indeed some program information (variables values, accessed memory addresses, etc.) being available only at runtime, another part of the available parallelism will have to be generated on-the-fly during the execution, here also, thanks to a dynamic mechanism.

This on-the-fly parallelism extraction will be performed using speculative behavior models (issue 2), such models allowing to generate speculative parallel code (issue 3). Between our behavior modeling objectives, we can add the behavior monitoring, or profiling, of a program version. Indeed current and future architectures complexity avoids assuming an optimal behavior regarding a given program version. A monitoring process will allow to select on-the-fly the best parallelization.

These different parallelizing steps are schematized on figure 1 .

Our project lies on the conception of a production chain for efficient execution of an application on a multicore architecture. Each link of this chain has to be formally verified in order to ensure correction as well as efficiency. More precisely, it has to be ensured that the compiler produces a correct intermediate code, and that the virtual machine actually performs the parallel execution semantically equivalent to the source code: every transformation applied to the application, either statically by the compiler or dynamically by the virtual machine, must preserve the initial semantics. They must be proved formally (issue 4).

In the following, those different issues are detailed while forming our global and long term vision of what has to be done.

#### 3.2. Static Parallelization and Optimization

**Participants:** Vincent Loechner, Philippe Clauss, Éric Violard, Cédric Bastoul, Arthur Charguéraud.



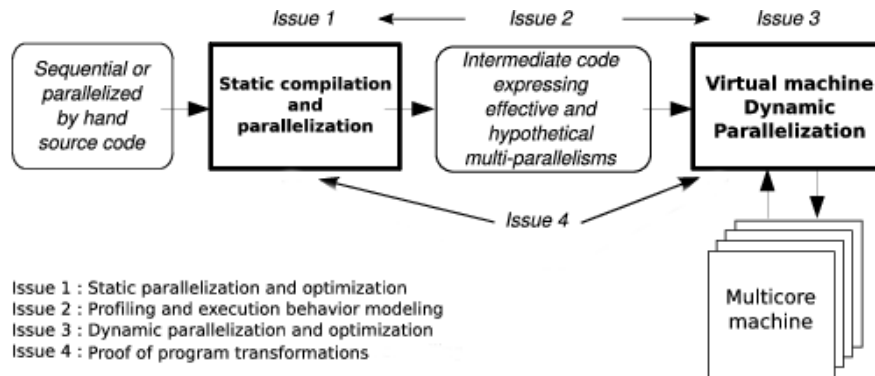


Figure 1. Automatic parallelizing steps for multicore architectures

Static optimizations, from source code at compile time, benefit from two decades of research in automatic parallelization: many works address the parallelization of loop nests accessing multi-dimensional arrays, and these works are now mature enough to generate efficient parallel code [23]. Low-level optimizations, in the assembly code generated by the compiler, have also been extensively dealt for single-core and require few adaptations to support multicore architectures. Concerning multicore specific parallelization, we propose to explore two research directions to take full advantage of these architectures: adapting parallelization to multicore architecture and expressing many potential parallelisms.

### 3.3. Profiling and Execution Behavior Modeling

**Participants:** Alain Ketterlin, Philippe Clauss, Manuel Selva.

The increasing complexity of programs and hardware architectures makes it ever harder to characterize beforehand a given program's run time behavior. The sophistication of current compilers and the variety of transformations they are able to apply cannot hide their intrinsic limitations. As new abstractions like transactional memories appear, the dynamic behavior of a program strongly conditions its observed performance. All these reasons explain why empirical studies of sequential and parallel program executions have been considered increasingly relevant. Such studies aim at characterizing various facets of one or several program runs, *e.g.*, memory behavior, execution phases, etc. In some cases, such studies characterize more the compiler than the program itself. These works are of tremendous importance to highlight all aspects that escape static analysis, even though their results may have a narrow scope, due to the possible incompleteness of their input data sets.

### 3.4. Dynamic Parallelization and Optimization, Virtual Machine

**Participants:** Manuel Selva, Juan Manuel Martinez Caamaño, Luis Esteban Campostrini, Artiom Baloian, Mariem Saied, Daniel Salas, Philippe Clauss, Jens Gustedt, Vincent Loechner, Alain Ketterlin.

This link in the programming chain has become essential with the advent of the new multicore architectures. Still being considered as secondary with mono-core architectures, dynamic analysis and optimization are now one of the keys for controlling those new mechanisms complexity. From now on, performed instructions are not only dedicated to the application functionalities, but also to its control and its transformation, and so in its own interest. Behaving like a computer virus, such a process should rather be qualified as a "vitamin". It perfectly knows the current characteristics of the execution environment and owns some qualitative information thanks to a behavior modeling process (issue 2). It appends a significant part of optimizing ability compared to a static compiler, while observing live resources availability evolution.

### **3.5. Proof of Program Transformations for Multicores**

**Participants:** Éric Violard, Alain Ketterlin, Julien Narboux, Nicolas Magaud, Arthur Charguéraud.

Our main objective consists in certifying the critical modules of our optimization tools (the compiler and the virtual machine). First we will prove the main loop transformation algorithms which constitute the core of our system.

The optimization process can be separated into two stages: the transformations consisting in optimizing the sequential code and in exhibiting parallelism, and those consisting in optimizing the parallel code itself. The first category of optimizations can be proved within a sequential semantics. For the other optimizations, we need to work within a concurrent semantics. We expect the first stage of optimizations to produce data-race free code. For the second stage of optimizations, we will first assume that the input code is data-race free. We will prove those transformations using Appel's concurrent separation logic [25]. Proving transformations involving program which are not data-race free will constitute a longer term research goal.

## COMPSYS Team

### 3. Research Program

#### 3.1. Architecture and Compilation Trends

The embedded system design community is facing two challenges:

- The complexity of embedded applications is increasing at a rapid rate.
- The needed increase in processing power is no longer obtained by increases in the clock frequency, but by increased parallelism.

While, in the past, each type of embedded application was implemented in a separate appliance, the present tendency is toward a universal hand-held object, which must serve as a cell-phone, as a personal digital assistant, as a game console, as a camera, as a Web access point, and much more. One may say that embedded applications are of the same level of complexity as those running on a PC, but they must use a more constrained platform in terms of processing power, memory size, and energy consumption. Furthermore, most of them depend on international standards (e.g., in the field of radio digital communication), which are evolving rapidly. Lastly, since ease of use is at a premium for portable devices, these applications must be integrated seamlessly to a degree that is unheard of in standard computers.

All of this dictates that modern embedded systems retain some form of programmability. For increased designer productivity and reduced time-to-market, programming must be done in some high-level language, with appropriate tools for compilation, run-time support, and debugging. This does not mean however that all embedded systems (or all of an embedded system) must be processor based. Another solution is the use of field programmable gate arrays (FPGA), which may be programmed at a much finer grain than a processor, although the process of FPGA “programming” is less well understood than software generation. Processors are better than application-specific circuits at handling complicated control and unexpected events. On the other hand, FPGAs may be tailored to just meet the needs of their application, resulting in better energy and silicon area usage. It is expected that most embedded systems will use a combination of general-purpose processors, specific processors like DSPs, and FPGA accelerators (or even low-power GPUs). Such a combination DSP+FPGA is already present in recent versions of the Atom Intel processor.

As a consequence, parallel programming, which has long been confined to the high-performance community, must become the common place rather than the exception. In the same way that sequential programming moved from assembly code to high-level languages at the price of a slight loss in performance, parallel programming must move from low-level tools, like OpenMP or even MPI, to higher-level programming environments. While fully-automatic parallelization is a Holy Grail that will probably never be reached in our lifetimes, it will remain as a component in a comprehensive environment, including general-purpose parallel programming languages, domain-specific parallelizers, parallel libraries and run-time systems, back-end compilation, dynamic parallelization. The landscape of embedded systems is indeed very diverse and many design flows and code optimization techniques must be considered. For example, embedded processors (micro-controllers, DSP, VLIW) require powerful back-end optimizations that can take into account hardware specificities, such as special instructions and particular organizations of registers and memories. FPGA and hardware accelerators, to be used as small components in a larger embedded platform, require “hardware compilation”, i.e., design flows and code generation mechanisms to generate non-programmable circuits. For the design of a complete system-on-chip platform, architecture models, simulators, debuggers are required. The same is true for multicores of any kind, GPGPU (“general-purpose” graphical processing units), CGRA (coarse-grain reconfigurable architectures), which require specific methodologies and optimizations, although all these techniques converge or have connections. In other words, embedded systems need all usual aspects of the process that transforms some specification down to an executable, software or hardware. In this wide range of topics, Compsys concentrated on the code optimizations aspects (and the associated analysis) in this transformation chain, restricting to compilation (transforming a program to a program) for embedded

processors and programmable accelerators, and to high-level synthesis (transforming a program into a circuit description) for FPGAs.

Actually, it is not a surprise to see compilation and high-level synthesis getting closer (in the last 10 years now). Now that high-level synthesis has grown up sufficiently to be able to rely on place-and-route tools, or even to synthesize C-like languages, standard techniques for back-end code generation (register allocation, instruction selection, instruction scheduling, software pipelining) are used in HLS tools. At the higher level, programming languages for programmable parallel platforms share many aspects with high-level specification languages for HLS, for example the description and manipulations of nested loops, or the model of computation/communication (e.g., Kahn process networks and its many “streaming” variants). In all aspects, the frontier between software and hardware is vanishing. For example, in terms of architecture, customized processors (with processor extensions as first proposed by Tensilica) share features with both general-purpose processors and hardware accelerators. FPGAs are both hardware and software as they are fed with “programs” representing their hardware configurations.

In other words, this convergence in code optimizations explains why Compsys studied both program compilation and high-level synthesis, and at both front-end and back-end levels, the first one acting more at the granularity of memories, transfers, and multiple cores, the second one more at the granularity of registers, system calls, and single core. Both levels must be considered as they interact with each other. Front-end optimizations must be aware of what back-end optimizations will do, as single core performance remain the basis for good parallel performances. Some front-end optimizations even act directly on back-end features, for example register tiling considered as a source-level transformation. Also, from a conceptual point of view, the polyhedral techniques developed by Compsys are actually the symbolic front-end counterpart, for structured loops, of back-end analysis and optimizations of unstructured programs (through control-flow graphs), such as dependence analysis, scheduling, lifetime analysis, register allocation, etc. A strength of Compsys was to juggle with both aspects, the first one based on graph theory with SSA-type optimizations, the other on polyhedra representing loops, and to exploit the correspondence between both. This has still to be exploited, for applying polyhedral techniques to more irregular programs. Besides, Compsys had a tradition of building free software tools for linear programming and optimization in general, as needed for our research.

### **3.1.1. *Compilation and Languages Issues in the Context of Embedded Processors, “Embedded Systems”, and Programmable Accelerators***

Compilation is an old activity, in particular back-end code optimizations. The development of embedded systems was one of the reasons for the revival of compilation activities as a research topic. Applications for embedded computing systems generate complex programs and need more and more processing power. This evolution is driven, among others, by the increasing impact of digital television, the first instances of UMTS networks, and the increasing size of digital supports, like recordable DVD, and even Internet applications. Furthermore, standards are evolving very rapidly (see for instance the successive versions of MPEG). As a consequence, the industry has focused on programmable structures, whose flexibility more than compensates for their larger size and power consumption. The appliance provider has a choice between hard-wired structures (Asic), special-purpose processors (Asip), (quasi) general-purpose processors (DSP for multimedia applications), and now hardware accelerators (dedicated platforms – such as those developed by Thales or the CEA –, or more general-purpose accelerators such as GPUs or even multicores, even if these are closer to small HPC platforms than truly embedded systems). Our cooperation with STMicroelectronics, until 2012, focused on investigating the compilation for specialized processors, such as the ST100 (DSP processor) and the ST200 (VLIW DSP processor) family. Even for this restricted class of processors, the diversity is large, and the potential for instruction level parallelism (SIMD, MMX), the limited number of registers and the small size of the memory, the use of direct-mapped instruction caches, of predication, generated many open problems. Our goal was to contribute to their understanding and their solutions.

An important concept to cope with the diversity of platforms is the concept of *virtualization*, which is a key for more portability, more simplicity, more reliability, and of course more security. This concept – implemented at low level through binary translation and just-in-time (JIT) compilation<sup>0</sup> – consists in hiding the architecture-

---

dependent features as long as possible during the compilation process. It has been used for a while for servers such as HotSpot, a bit more recently for workstations, and now for embedded computing. The same needs drive the development of intermediate languages such as OpenCL to, not necessarily hide, but at least make more uniform, the different facets of the underlying architectures. The challenge is then to design and compile high-productivity and high-performance languages<sup>0</sup> (coping with parallelism and heterogeneity) that can be ported to such intermediate languages, or to architecture-dependent runtime systems. The offloading of computation kernels, through source-to-source compilation, targeting back-end C dialects, has the same goals: to automate application porting to the variety of accelerators.

For JIT compilation, the compactness of the information representation, and thus its pertinence, is an important criterion for such late compilation phases. Indeed, the intermediate representation (IR) is evolving not only from a target-independent description to a target-dependent one, but also from a situation where the compilation time is almost unlimited (cross-compilation) to one where any type of resource is limited. This is one of the reasons why static single assignment (SSA), a sparse compact representation of liveness information, became popular in embedded compilation. If time constraints are common to all JIT compilers (not only for embedded computing), the benefit of using SSA is also in terms of its good ratio pertinence/storage of information. It also enables to simplify algorithms, which is also important for increasing the reliability of the compiler. In this context, our aim has been, in particular, to develop exact or heuristic solutions to *combinatorial* problems that arise in compilation for VLIW and DSP processors, and to integrate these methods into industrial compilers for DSP processors (mainly ST100, ST200, Strong ARM). Such combinatorial problems can be found in register allocation, opcode selection, code placement, when removing the SSA multiplexer functions (known as  $\phi$  functions). These optimizations are usually done in the last phases of the compiler, using an assembly-level intermediate representation. As mentioned in Sections 2.3 and 2.4, we made a lot of progress in this area in our past collaborations with STMicroelectronics (see also previous activity reports). Through the Sceptre and Mediacom projects, we first revisited, in the light of SSA, some code optimizations in an aggressive context, to develop better strategies, without eliminating too quickly solutions that may have been considered as too expensive in the past. Then we exploited the new concepts introduced in the aggressive context to design better algorithms in a JIT context, focusing on the speed of algorithms and their memory footprint, without compromising too much on the quality of the generated code.

Our recent research directions were more focused on programmable accelerators, such as GPU and multicores, but still considering *static* compilation and without forgetting the link between high-level (in general at source-code level) and low-level (i.e., at assembly-code level) optimizations. They concerned program analysis (of both sequential and parallel specifications), program optimizations (for memory hierarchies, parallelism, streaming, etc.), and also the link with applications, and between compilers and users (programmers). Polyhedral techniques play an important role in these directions, even if control-flow-based techniques remain in the background and may come back at any time in the foreground. This is also the case for high-level synthesis, as exposed in the next section.

### 3.1.2. Context of High-Level Synthesis and FPGA Platforms

High-level synthesis has become a necessity, mainly because the exponential increase in the number of gates per chip far outstrips the productivity of human designers. Besides, applications that need hardware accelerators usually belong to domains, like telecommunications and game platforms, where fast turn-around and time-to-market minimization are paramount. When Compsys started, we were convinced that our expertise in compilation and automatic parallelization could contribute to the development of the needed tools.

---

<sup>0</sup>Aggressive compilation consists in allowing more time to implement more complete and costly solutions: compilation time is less relevant than execution time, size, and energy consumption of the produced code, which can have a critical impact on the cost and quality of the final product. The application is usually cross-compiled, i.e., compiled on a powerful platform distinct from the target processor. *Just-in-time compilation*, on the other hand, corresponds to compiling applets on demand on the target processor. The code can be uploaded or sold separately on a flash memory. Compilation is performed at load time and even dynamically during execution. The optimization heuristics, constrained by time and limited resources, are far from being aggressive. They must be fast but smart enough.

<sup>0</sup>For examples of such languages, see the keynotes event we organized in 2013: <http://labexcompilation.ens-lyon.fr/hpc-languages>.

Today, synthesis tools for FPGAs or ASICs come in many shapes. At the lowest level, there are proprietary Boolean, layout, and place-and-route tools, whose input is a VHDL or Verilog specification at the structural or register-transfer level (RTL). Direct use of these tools is difficult, for several reasons:

- A structural description is completely different from an usual algorithmic language description, as it is written in term of interconnected basic operators. One may say that it has a spatial orientation, in place of the familiar temporal orientation of algorithmic languages.
- The basic operators are extracted from a library, which poses problems of selection, similar to the instruction selection problem in ordinary compilation.
- Since there is no accepted standard for VHDL synthesis, each tool has its own idiosyncrasies and reports its results in a different format. This makes it difficult to build portable HLS tools.
- HLS tools have trouble handling loops. This is particularly true for logic synthesis systems, where loops are systematically unrolled (or considered as sequential) before synthesis. An efficient treatment of loops needs the polyhedral model. This is where past results from the automatic parallelization community are useful.
- More generally, a VHDL specification is too low level to allow the designer to perform, easily, higher-level code optimizations, especially on multi-dimensional loops and arrays, which are of paramount importance to exploit parallelism, pipelining, and perform communication and memory optimizations.

Some intermediate tools were proposed that generate VHDL from a specification in restricted C, both in academia (such as SPARK, Gaut, UGH, CloogVHDL), and in industry (such as C2H, CatapultC, Pico-Express, Vivado HLS). All these tools use only the most elementary form of parallelization, equivalent to instruction-level parallelism in ordinary compilers, with some limited form of block pipelining, and communication through FIFOs. Targeting one of these tools for low-level code generation, while we concentrate on exploiting loop parallelism, might be a more fruitful approach than directly generating VHDL. However, it may be that the restrictions they impose preclude efficient use of the underlying hardware. Our first experiments with these HLS tools reveal two important issues. First, they are, of course, limited to certain types of input programs so as to make their design flows successful, even if, over the years, they become more and more mature. But it remains a painful and tricky task for the user to transform the program so that it fits these constraints and to tune it to get good results. Automatic or semi-automatic program transformations can help the user achieve this task. Second, users, even expert users, have only a very limited understanding of what back-end compilers do and why they do not lead to the expected results. An effort must be done to analyze the different design flows of HLS tools, to explain what to expect from them, and how to use them to get a good quality of results. Our first goal was thus to develop high-level techniques that, used in front of existing HLS tools, improve their utilization. This should also give us directions on how to modify them or to design new tools from scratch.

More generally, HLS has to be considered as a more global parallelization process. So far, no HLS tool is capable of generating designs with communicating *parallel* accelerators, even if, in theory, at least for the scheduling part, a tool such as Pico-Express could have such capabilities. The reason is that it is, for example, very hard to automatically design parallel memories and to decide the distribution of array elements in memory banks to get the desired performances with parallel accesses. Also, how to express communicating processes at the language level? How to express constraints, pipeline behavior, communication media, etc.? To better exploit parallelism, a first solution is to extend the source language with parallel constructs, as in all derivations of the Kahn process networks model, including communicating regular processes (CRP). The other solution is a form of automatic parallelization. However, classical methods, which are mostly based on scheduling, need to be revisited, to pay more attention to locality, process streaming, and low-level pipelining, which are of paramount importance in hardware. Besides, classical methods mostly rely on the runtime system to tailor the parallelism degree to the available resources. Obviously, there is no runtime system in hardware. The real challenge is thus to invent new scheduling algorithms that take resource, locality, and pipelining into account, and then to infer the necessary hardware from the schedule. This is probably possible only for programs that fit into the polyhedral model, or in an incrementally-extended model.

Our research activities on polyhedral code analysis and optimizations directly targeted these HLS challenges. But they are not limited to the automatic generation of hardware as can be seen from our different contributions on X10, OpenStream, parametric tiling, etc. The same underlying concepts also arise when optimizing codes for GPUs and multicores. In this context of polyhedral analysis and optimizations, we focused on three aspects:

- developing high-level transformations, especially for loops and memory/communication optimizations, that can be used in front of HLS tools so as to improve their use, as well as for hardware accelerators;
- developing concepts and techniques in a more global view of high-level synthesis and high-level parallel programming, starting from specification languages down to hardware implementation;
- developing more general code analysis so as to extract more information from codes as well as to extend the programs that can be handled.

## 3.2. Code Analysis, Code Transformations, Code Optimizations

Embedded systems, as we recalled earlier, generated new problems in code analysis and optimization both for optimizing embedded software (compilation) and hardware (HLS). We now give a bit more details on some general challenges for program analysis, optimizations, and transformations, induced by this context, and on our methodology, in particular our development and use of polyhedral optimizations and its extensions.

### 3.2.1. Processes, Scheduling, Mapping, Communications, etc.

Before mapping an application to an architecture, one has to decide which execution model is targeted and where to intervene in the design flow. Then one has to solve scheduling, placement, and memory management problems. These three aspects should be handled as a whole, but present state of the art dictates that they be treated separately. One of our aims was to develop more comprehensive solutions. The last task is code generation, both for the processing elements and the interfaces processors/accelerators.

There are basically two execution models for embedded systems: one is the classical accelerator model, in which data is deposited in the memory of the accelerator, which then does its job, and returns the results. In the streaming model, computations are done on the fly, as data items flow from an input channel to the output. Here, the data are never stored in (addressable) memory. Other models are special cases, or sometimes compositions of the basic models. For instance, a systolic array follows the streaming model, and sometimes extends it to higher dimensions. Software radio modems follow the streaming model in the large, and the accelerator model in detail. The use of first-in first-out queues (FIFO) in hardware design is an application of the streaming model. Experience shows that designs based on the streaming model are more efficient than those based on memory, for such applications. One of the points to be investigated is whether it is general enough to handle arbitrary (regular) programs. The answer is probably negative. One possible implementation of the streaming model is as a network of communicating processes either as Kahn process networks (FIFO based) or as our more recent model of communicating regular processes (memory based, such as CRP mentioned hereafter). It is an interesting fact that several researchers have investigated the translation from process networks [12] and to process networks [20], [21]. Streaming languages such as StreamIt and OpenStream are also interesting solutions to explore.

Kahn process networks (KPN) were introduced 30 years ago as a notation for representing parallel programs. Such a network is built from processes that communicate via perfect FIFO channels. Because the channel histories are deterministic, one can define a semantics and talk meaningfully about the equivalence of two implementations. As a bonus, the dataflow diagrams used by signal processing specialists can be translated on-the-fly into process networks. The problem with KPNs is that they rely on an asynchronous execution model, while VLIW processors and FPGAs are synchronous or partially synchronous. Thus, there is a need for a tool for synchronizing KPNs. This can be done by computing a schedule that has to satisfy data dependences within each process, a causality condition for each channel (a message cannot be received before it is sent), and real-time constraints. However, there is a difficulty in writing the channel constraints because one has to count messages in order to establish the send/receive correspondence and, in multi-dimensional loop nests, the counting functions may not be affine. The same situation arises for the OpenStream language (see Section 7.2).

Recent developments on the theory of polynomials (see Section 7.1) may offer a solution to this problem. One can also define another model, *communicating regular processes* (CRP), in which channels are represented as write-once/read-many arrays. One can then dispense with counting functions and prove that the determinacy property still holds. As an added benefit, a communication system in which the receive operation is not destructive is closer to the expectations of system designers.

The main difficulty with this approach is that ordinary programs are usually not constructed as process networks. One needs automatic or semi-automatic tools for converting sequential programs into process networks. One possibility is to start from array dataflow analysis [15] or variants. Another approach attempts to construct threads, i.e., pieces of sequential code with the smallest possible interactions. In favorable cases, one may even find outermost parallelism, i.e., threads with no interactions whatsoever. Tiling mechanisms can also be used to define atomic processes that can be pipelined as we proposed initially for FPGA [9].

Whatever the chosen solution (FIFO or addressable memory) for communicating between two accelerators or between the host processor and an accelerator, the problems of optimizing communication between processes and of optimizing buffers have to be addressed. Many local memory optimization problems have already been solved theoretically. Some examples are loop fusion and loop alignment for array contraction, techniques for data allocation in scratch-pad memory, or techniques for folding multi-dimensional arrays [11]. Nevertheless, the problem is still largely open. Some questions are: how to schedule a loop sequence (or even a process network) for minimal scratch-pad memory size? How is the problem modified when one introduces unlimited and/or bounded parallelism (same questions for analyzing explicitly-parallel programs)? How does one take into account latency or throughput constraints, bandwidth constraints for input and output channels, memory hierarchies? All loop transformations are useful in this context, in particular loop tiling, and may be applied either as source-to-source transformations (when used in front of HLS or C-level compilers) or to generate directly VHDL or lower-level C-dialects such as OpenCL. One should keep in mind that theory will not be sufficient to solve these problems. Experiments are required to check the relevance of the various models (computation model, memory model, power consumption model) and to select the most important factors according to the architecture. Besides, optimizations do interact: for instance, reducing memory size and increasing parallelism are often antagonistic. Experiments will be needed to find a global compromise between local optimizations. In particular, the design of cost models remain a fundamental challenge.

Finally, there remains the problem of code generation for accelerators. It is a well-known fact that methods for program optimization and parallelization do not generate a new program, but just deliver blueprints for program generation, in the form, e.g., of schedules, placement functions, or new array subscripting functions. A separate code generation phase must be crafted with care, as a too naive implementation may destroy the benefits of high-level optimization. There are two possibilities here as suggested before; one may target another high-level synthesis or compilation tool, or one may target directly VHDL or low-level code. Each approach has its advantages and drawbacks. However, both situations require that the input program respects some strong constraints on the code shape, array accesses, memory accesses, communication protocols, etc. Furthermore, to get the compilers do what the user wants requires a lot of program tuning, i.e., of program rewriting or of program annotations. What can be automated in this rewriting process? Semi-automated?

In other words, we still need to address scheduling, memory, communication, and code generation issues, in the light of the developments of new languages and architectures, pushing the limits of such an automation of program analysis, program optimizations, and code generation.

### 3.2.2. Beyond Static Control Programs

With the advent of parallelism in supercomputers, the bulk of research in code transformation resulted in (semi-)automatic parallelization, with many techniques (analysis, scheduling, code generation, etc.) based on the description and manipulation of nested loops with polyhedra. Compsys has always taken an active part in the development of these so-called “polyhedral techniques”. Historically, these analysis were (wrongly) understood to be limited to static control programs.

Actually, the polyhedral model is neither a programming language nor an execution model, rather an intermediate representation. As such, it can be generated from imperative sequential languages like C or



Fortran, streaming languages like CRP, or equational languages like Alpha. While the structure of the model is the same in all three cases, it may enjoy different properties, e.g., a schedule always exists in the first case, not in the two others. The import of the polyhedral model is that many questions relative to the analysis of a program and the applicability of transformations can be answered precisely and efficiently by applying well-known mathematical results to the model.

For irregular programs, the basic idea is to construct a polyhedral over-approximation, i.e., a program which has more operations, a larger memory footprint, and more dependences than the original. One can then parallelize the approximated program using polyhedral tools, and then return to the original, either by introducing guards, or by insuring that approximations are harmless. This technique is the standard way of dealing with approximated dependences. We already started to study the impact of approximations in our kernel offloading technique, for optimizing remote communications [10]. It is clear however that this extension method based on over-approximation will apply only to mildly non-polyhedral programs. The restriction to arrays as the only data structure is still present. Its advantage is that it will be able to subsume in a coherent framework many disparate tricks: the extraction of SCoPs, induction variable detection, the omission of non-affine subscripts, or the conversion of control dependences into data dependences. The link with the techniques developed in the PIPS compiler (based on array region analysis) is strong and will have to be explored.

Such over-approximations can be found by mean of abstract interpretation, a general framework to develop static analysis on real-life programs. However, they were designed mainly for verification purposes, thus precision was the main issue before scalability. Although many efforts were made in designing specialized analyses (pointers, data structures, arrays), these approaches still suffer from a lack of experimental evidence concerning their applicability for code optimization. Following our experience and work on termination analysis (that connects the work on back-end CFG-like and front-end polyhedral-like optimizations), and our work on range analysis of numerical variables and on the memory footprint on real-world C programs [18], one of our objectives for the future was to bridge the gap between abstract interpretation and compilation, by designing cheaper analyses that scale well, mainly based on compact representations derived from variants of static single assignment (SSA), with a special focus on complex control, and complex data structures (pointers, lists) that still suffer from complexity issues in the area of optimization.

Another possibility is to rely on application specific knowledge to guide compiler decisions, as it is impossible for a compiler alone to fully exploit such pieces of information. A possible approach to better utilize such knowledge is to put the programmers “in the loop”. Expert parallel programmers often have a good idea about coarse-grain parallelism and locality that they want to use for an application. On the other hand, fine-grain parallelism (e.g., ILP, SIMD) is tedious and specific to each underlying architecture, and is best left to the compiler. Furthermore, approximations will have opportunities to be refined using programmer knowledge. The key challenge is to create a programming environment where compiler techniques and programmer knowledge can be combined effectively. One of the difficulties is to design a common language between the compiler and the programmer. The first step towards this objective is to establish inter-disciplinary collaborations with users, and take the time to analyze and optimize their applications together.M

### 3.3. Mathematical Tools

All compilers have to deal with *sets* and relations. In classical compilers, these sets are finite: the set of statements of a program, the set of its variables, its abstract syntax tree (AST), its control-flow graph (CFG), and many others. It is only in the first phase of compilation, parsing, that one has to deal with infinite objects, regular and context-free languages, and those are represented by finite grammars, and are processed by a symbolic algorithm, yacc or one of its clones.

When tackling parallel programs and parallel compilation, it was soon realized that this position was no longer tenable. Since it makes no sense to ask whether a statement can be executed in parallel with itself, one has to consider sets of operations, which may be so large as to forbid an extensive representation, or even be infinite. The same is true for dependence sets, for memory cells, for communication sets, and for many other objects a parallel compiler has to consider. The representation is to be *symbolic*, and all necessary algorithms have to be promoted to symbolic versions.

Such symbolic representations have to be efficient – the formula representing a set has to be much smaller than the set itself – and effective – the operations one needs, union, intersection, emptiness tests and many others – have to be feasible and fast. As an aside, note that progress in algorithm design has blurred the distinction between polynomially-solvable and NP-complete problems, and between decidable and undecidable questions. For instance SAT, SMT, and ILP software tools solve efficiently many NP-complete problems, and the Z3 tool is able to “solve” many instances of the undecidable Hilbert’s 10th problem.

Since the times of Pip and of the Polylib, Compsys has been active in the implementation of basic mathematical tools for program analysis and synthesis. Pip is still developed by Paul Feautrier and Cédric Bastoul, while the Polylib is now taken care of by the Inria Camus project, which introduced Ehrhart polynomials. These tools are still in use world-wide and they also have been reimplemented many times with (sometimes slight) improvements, e.g., as part of the Parma Polylib, of Sven Verdoolaege’s Isl and Barvinok libraries, or of the Jollylib of Reservoir Labs. Other groups also made a lot of efforts towards the democratization of the use of polyhedral techniques, in particular the Alchemy Inria project, with Cloog and the development of Graphite in GCC, and Sadayappan’s group in the USA, with the development of U. Bondhugula’s Pluto prototype compiler. The same effort is made through the PPCG prototype compiler (for GPU) and Pencil (directives-based language on top of PPCG).

After 2009, Compsys continued to focus on the introduction of concepts and techniques to extend the polytope model, with a shift toward tools that may prepare the future. For instance, PoCo and C2fsm are able to parse general programs, not just SCoPs (static control programs), while the efficient handling of Boolean affine formulas [13] is a prerequisite for the construction of non-convex approximations. Euclidean lattices provide an efficient abstraction for the representation of spatial phenomena, and the construction of *critical lattices* as embedded in the tool Cl@k is a first step towards memory optimization in stream languages and may be useful in other situations. Our work on Chuba introduced a new element-wise array reuse analysis and the possibility of handling approximations. Our work on the analysis of while loops is both an extension of the polytope model itself (i.e., beyond SCoPs) and of its applications, here links with program termination and worst-case execution time (WCET) tools.

A recent example of this extension idea is the proposal by Paul Feautrier to use polynomials for program analysis and optimization [14]. The associated tools are based on Handelman and Schweighofer theorems, the polynomial analogue of Farkas lemma. While this is definitely work in progress, with many unsolved questions, it has the potential of greatly enlarging the set of tractable programs.

As a last remark, observe that a common motif of these developments is the transformation of finite algorithms into symbolic algorithms, able to solve very large or even infinite instances. For instance, PIP is a symbolic extension of the Simplex; our work on memory allocation is a symbolic extension of the familiar register allocation problem; loop scheduling extends DAG scheduling. Many other algorithms await their symbolic transformation: a case in point is resource-constrained scheduling.

## **CORSE Project-Team**

### **3. Research Program**

#### **3.1. Scientific Foundations**

One of the characteristics of CORSE is to base our researches on diverse advanced mathematical tools. Compiler optimization requires the usage of the several tools around discrete mathematics: combinatorial optimization, algorithmic, and graph theory. The aim of CORSE is to tackle optimization not only for regular but also for irregular applications. We believe that new challenges in compiler technology design and in particular for split compilation should also take advantage of graph labeling techniques. In addition to runtime and compiler techniques for program instrumentation, hybrid analysis and compilation advances will be mainly based on polynomial and linear algebra.

The other specificity of CORSE is to address technical challenges related to compiler technology, runtime systems, and hardware characteristics. This implies mastering the details of each. This is especially important as any optimization is based on a reasonably accurate model. Compiler expertise will be used in modeling applications (e.g. through automatic analysis of memory and computational complexity); Runtime expertise will be used in modeling the concurrent activities and overhead due to contention (including memory management); Hardware expertise will be extensively used in modeling physical resources and hardware mechanisms (including synchronization, pipelines, etc.).

The core foundation of the team is related to the combination of static and dynamic techniques, of compilation, and runtime systems. We believe this to be essential in addressing high-performance and low energy challenges in the context of new important changes shown by current application, software, and architecture trends.

Our project is structured along two main directions. The first direction belongs to the area of runtime systems with the objective of developing strong relations with compilers. The second direction belongs to the area of compiler analysis and optimization with the objective of combining dynamic analysis and optimization with static techniques. The aim of CORSE is to ground those two research activities on the development of the end-to-end optimization of some specific domain applications.

## DREAMPAL Project-Team

### 3. Research Program

#### 3.1. New Models for New Technologies

Over the past 25 years there have been several hardware-architecture generations dedicated to massively parallel computing. We have contributed to them in the past, and shall continue doing so in the Dreampal project. The three generations, chronologically ordered, are:

- Supercomputers from the 80s and 90s, based on massively parallel architectures that are more or less distributed (from the Cray T3D or Connection Machine CM2 to GRID 5000). Computer scientists have proposed methods and tools for mapping sequential algorithms to those parallel architectures in order to extract maximum power from them. We have contributed in this area in the past.
- Parallelism pervades the chips! A new challenge appears: hardware/software co-design, in order to obtain performance gains by designing algorithms together with the parallel architectures of chips adapted to the algorithms. During the previous decade many studies, including ours in the Inria DaRT team, were dedicated to this type of co-design. DaRT has contributed to the development of the OMG MARTE standard (<http://www.omgmarTE.org>) and to its implementation on several parallel platforms. Gaspard2, our implementation of this concept, was identified as one of the key software tools developed at Inria: <http://www.inria.fr/en/centre/lille/research/platforms-and-flagship-software/flagship-software>.
- The new challenge of the 2010s is, in our opinion, the integration of dynamic reconfiguration and massive parallelism. New circuits with high-density integration and supporting dynamic hardware reconfiguration have been proposed. In such architectures one can dynamically change the architecture while an algorithm is running on it. The Dynamic Partial Reconfiguration (DPR) feature offered by recent FPGA boards even allows, in theory, to generate optimized hardware at runtime, by adding, removing, and replacing components on a by-need basis. This integration of dynamic reconfiguration and massive parallelism induces a new degree of complexity, which we, as computer scientists, need to understand and deal with in order to make possible the design of applications running on such architectures. This is the main challenge that we address in the Dreampal project. We note that we address these problems as computer scientists; we do, however, collaborate with electronics specialists in order to benefit from their expertise in 3-D FPGAs.

Excerpt from the HiPEAC vision 2011/12

*“The advent of 3D stacking enables higher levels of integration and reduced costs for off-chip communications. The overall complexity is managed due to the separation in different dies, independently designed.”*

FPGAs (Field Programmable Gate Arrays) are configurable circuits that have emerged as a privileged target platform for intensive signal processing applications. FPGAs take advantage of the latest technological developments in circuits. For example, the Virtex7 from Xilinx offers a 28-nanometer integration, which is only one or two generations behind the latest general-purpose processors. 3D-Stacked Integrated Circuits (3D SICs) consist of two or more conventional 2D circuits stacked on the top of each other and built into the same IC. Recently, 3D SICs have been released by Xilinx for the Virtex 7 FPGA family. 3D integration will vastly increase the integration capabilities of FPGA circuits. The convergence of massive parallelism and dynamic reconfiguration is inevitable: we believe it is one of the main challenges in computing for the current decade.

By incorporating the configuration and/or data/program memory on the top of the FPGA fabric, with fast and numerous connections between memory and elementary logic blocks (~10000 connections between dies), it will be possible to obtain dynamically reconfigurable computing platforms with a very high reconfiguration rate. Such a rate was not possible before, due to the serial nature of the interface between the configuration memory and the FPGA fabric itself. The FPGA technology also enables massively parallel architectures due to the large number of programmable logic fabrics available on the chip. For instance, Xilinx demonstrated 3600 8-bit picoBlaze softcore processors running simultaneously on the Virtex-7 2000T FPGA. For specific applications, picoBlaze can be replaced by specialized hardware accelerators or other IPs (Intellectual Property) components. This opens the possibility of creating massively parallel IP-based machines.

## 3.2. Multi-softcore on 3D FPGA

From the 2010 Xilinx white paper on FPGAs:

*“Unlike a processor, in which architecture of the ALU is fixed and designed in a general-purpose manner to execute various operations, the CLBs (configurable logic blocks ) can be programmed with just the operations needed by the application... The FPGA architecture provides the flexibility to create a massive array of application-specific ALUs..The new solution enables high-bandwidth connectivity between multiple die by providing a much greater number of connections... enabling the integration of massive quantities of interconnect logic resources within a single package”*

Softcore processors are processors implemented using hardware synthesis. Proprietary solutions include PicoBlaze, MicroBlaze, Nios, and Nios II; open-source solutions include Leon, OpenRisk, and FC16. The choice is wide and many new solutions emerge, including multi-softcore implementations on FPGAs. An alternative to softcores are hardware accelerators on FPGAs, which are dedicated circuits that are an order of magnitude faster than softcores. Between these two approaches, there are other various approaches that connect IPs to softcores, in which, the processor’s machine-code language is extended, and IP invocations become new instructions. We envisage a new class of softcores (we call them reflective softcores <sup>0</sup>), where almost everything is implemented in IPs; only the control flow is assigned to the softcore itself. The partial dynamic reconfiguration of next-generation FPGAs makes such dynamic IP management possible in practice. We believe that efficient reflective softcores on the new 3D-FPGAs should be as small as possible: low-performance generic hardware components (ALU, registers, memory, I/O...) should be replaced by dedicated high-performance IPs.

We are developing a softcore processor called HoMade (<http://www.lifl.fr/~dekeyser/Homade>) following these ideas.

In the multi-reflective softcores that we develop, some softcores will be slaves and others will be masters. Massively parallel dynamically reconfigurable architectures of softcores can thus be envisaged. This requires, additionally, a parallel management of the partial dynamic reconfiguration system. This can be done, for example, on a given subset of softcores: a massively parallel reconfiguration will replace the current replication of a given IP with the replication of a new IP. Thanks to the new 3D-FPGAs this task can be performed efficiently and in parallel using the large number of 3D communication links (Through-Silicon-Vias). Our roadmap for HoMade is to evolve towards this multi-reflective softcore model.

## 3.3. When Hardware Meets Software

HIPEAC vision 2011/12: *“The number of cores and instruction set extensions increases with every new generation, requiring changes in the software to effectively exploit the new features.”*

<sup>0</sup>Hereafter, by reflective system, we mean a system that is able to modify its own structure and behaviour while it is running. A reflective softcore thus dynamically adds, removes, and replaces IPs in the application running on it, and is able to dynamically modify its own program memory, thereby dynamically altering the program it is executing.

When the new massively parallel dynamically reconfigurable architectures become reality users will need languages for programming software applications on them. The languages will be themselves dynamic and parallel, in order to reflect and to fully exploit the dynamicity and parallelism of the architectures. Thus, developers will be able to invoke reconfiguration and call parallel instructions in their programs. This expressiveness comes with a cost, however, because new classes of bugs can be induced by the interaction between dynamic reconfiguration and parallelism; for example, deadlocks due to waiting for output from an IP that does not exist any more due to a reconfiguration. The detection and elimination of such bugs before deployment is paramount for cost-effectiveness and safety reasons.

Thus, we shall build an environment for developing software on parallel, dynamically reconfigurable architectures that will include languages and adequate formal analyses and verification tools for them, in addition to more traditional tools (emulators, compilers, etc). To this end we shall be using formal-semantics frameworks associated with easy-to-use formal verification tools in order to formally define our languages of interest and allow users to formally verify their programs. The K semantic framework (<http://k-framework.org>), developed jointly by Univ. Urbana Champaign, USA, and Iasi, Romania) is one such framework, which is mature enough (it has allowed defining a formal semantics of the largest subset of the C language to date, as well as many other languages from essentially all programming paradigms) and is familiar to us from previous work. In K, one can rapidly prototype a language definition and try several versions of the syntax and semantics of instructions. This is important in our project, where the proposed programming languages (in particular, the HoMade assembly language) will go through several versions before being stabilized. Moreover, once a language is defined in K one gets an interpreter of the language and one gains access to formal verification tools for free. We are also developing new analysis verification tools for K (in collaboration with the K team), which will be adapted and used in the Dreampal project.

## PACAP Project-Team

### 3. Research Program

#### 3.1. Motivation

Our research program is naturally driven by the evolution of our ecosystem. Relevant recent changes can be classified in the following categories: technological constraints, evolving community, and domain constraints. We hereby summarize these evolutions.

##### 3.1.1. Technological constraints

Until recently, binary compatibility guaranteed portability of programs, while increased clock frequency and improved micro-architecture provided increased performance. However, in the last decade, advances in technology and micro-architecture started translating into more parallelism instead. Technology roadmaps even predict the feasibility of thousands of cores on a chip by 2020. Hundreds are already commercially available. Since the vast majority of applications are still sequential, or contain significant sequential sections, such a trend put an end to the automatic performance improvement enjoyed by developers and users. Many research groups consequently focused on parallel architectures and compiling for parallelism.

The focus of ALF – and the DAL ERC – was paradoxically on Amdahl’s law: the performance of applications will ultimately be driven by the performance of the sequential part. Despite a number of advances (some of them contributed by ALF), sequential tasks are still a major performance bottleneck. Addressing it is still on the agenda of the proposed PACAP project-team.

In addition, due to power constraints, only part of the billions of transistors of a microprocessor can be operated at any given time (the *dark silicon* paradigm). A sensible approach consists in specializing parts of the silicon area to provide dedicated accelerators (not run simultaneously). This results in diverse and heterogeneous processor cores. Application and compiler designers are now confronted with a moving target, challenging portability and jeopardizing performance.

Finally, we live in a world where billions of sensors, actuators, and computers play a crucial role in our life: flight control, nuclear plant management, defense systems, banking, or health care. These systems must be reliable, despite the fact that they are subject to faults (for example due to aging, charged particle hit, or random noise). Faults will soon become the new *de facto* standard. The evolutions of the semiconductor industry predict an exponential growth of the number of permanent faults [58]. Reliability considerations usually degrade performance. We will propose solutions to mitigate this impact (for example by limiting overheads to critical sections).

*Note on technology.*

Technology also progresses at a fast pace. We do not propose to pursue any research on technology *per se*. Recently proposed paradigms (quantum computing, non-Si, brain-inspired) have received lots of attention from the research community. We do *not* intend to invest in those paradigms, but we will continue to investigate compilation and architecture for more conventional programming paradigms. Still, several technological shifts may have consequences for us, and we will closely monitor their developments, they include for example non-volatile memory (impacts security, makes writes longer than loads), 3D-stacking (impacts bandwidth), and photonics (impacts latencies and connection network).

##### 3.1.2. Evolving community

The PACAP project-team will tackle performance-related issues, for conventional programming paradigms. In fact, programming complex environments is no longer reserved to experts in compilation and architecture. A large community now develops applications for a wide range of targets, including mobile “apps”, cloud, multicore or heterogeneous processors.

This also includes domain scientists (in biology, medicine, but also social sciences) who started relying heavily on computational resources, gathering huge amounts of data, and requiring considerable amount of processing to analyze them. Our research is motivated by the growing discrepancy between on the one hand the complexity of the workloads and the computing systems, and on the other hand the expanding community of developers at large, with limited expertise to optimize and to map efficiently computations to compute nodes.

### 3.1.3. Domain constraints

Mobile, embedded systems have become ubiquitous. Many of them have real-time constraints. For this class of systems, correctness implies not only producing the correct result, but also doing so within specified deadlines. In the presence of heterogeneous, complex and highly dynamic systems, producing *tight* (i.e. useful) upper bound to the worst-case execution time has become extremely challenging. Our research will aim at improving the tightness as well as enlarging the set of features that can be safely analyzed.

The ever growing dependence of our economy on computing systems also implies that security has become of utmost importance. Many systems are under constant attacks from intruders. Protection has a cost also in terms of performance. We plan to leverage our background to contribute solutions that minimize this impact.

*Note on Applications Domains.*

As was already the case for ALF, PACAP will work on fundamental technologies for computer science: processor architecture, performance-oriented compilation and guaranteed response time for real-time. The research results may have impacts on any application domain that requires high performance execution (telecommunication, multimedia, biology, health, engineering, environment...), but also on many embedded applications that exhibit other constraints such as power consumption, code size and guaranteed response time.

We strive to extract from active domains the fundamental characteristics that are relevant to our research. For example, *big data* is of interest to PACAP because it relates to the study of hardware/software mechanisms to efficiently transfer huge amounts of data to the computing nodes. Similarly, the *Internet of Things* is of interest because it has implications in terms of ultra low power consumption.

## 3.2. Research Objectives

Processor micro-architecture and compilation have been at the core of the research carried by the CAPS and ALF project teams for two decades, with undeniable contributions. They will continue to be the foundation of PACAP.

Heterogeneity and diversity of processor architectures now require new techniques to guarantee that the hardware is satisfactorily exploited by the software. We will devise new static compilation techniques (cf. Section 3.2.1), but also build upon iterative [1] and split [2] compilation to continuously adapt software to its environment (Section 3.2.2). Dynamic binary optimization will also play a key role in delivering adapting software and delivering performance.

The end of Moore's law and Dennard's scaling<sup>0</sup> offer an exciting window of opportunity, where performance improvements will no longer derive from additional transistor budget or increased clock frequency, but rather come from breakthroughs in microarchitecture (Section 3.2.3). We will also consider how to reconcile CPU and GPU designs (Section 3.2.4).

Heterogeneity and multicores are also major obstacles to determining tight worst-case execution times of real-time systems (Section 3.2.5), which we plan to tackle.

Finally, we also describe how we plan to address transversal aspects such reliability (Section 3.2.6), power efficiency (Section 3.2.7), and security (Section 3.2.8).

---

<sup>0</sup>According to Dennard scaling, as transistors get smaller the power density remains constant, and the consumed power remains proportional to the area.



### 3.2.1. *Static Compilation*

Static compilation techniques will continue to be relevant to address the characteristics of emerging hardware technologies, such as non-volatile memories, 3D-stacking, or novel communication technologies. These techniques expose new characteristics to the software layers. As an example, non-volatile memories typically have asymmetric read-write latencies (writes are much longer than reads) and different power consumption profiles. PACAP will study the new optimization opportunities and develop tailored compilation techniques for the upcoming compute nodes. New technologies may also be coupled with traditional solutions to offer new trade-offs. We will study how programs can adequately exploit the specific features of the proposed heterogeneous compute nodes.

We propose to build upon iterative compilation [1] to explore how applications perform on different configurations. When possible, Pareto points will be related to application characteristics. The best configuration, however, may actually depend on runtime information, such as input data, dynamic events, or properties that are available only at runtime. Unfortunately a runtime system has little time and means to determine the best configuration. For these reasons, we will also leverage split-compilation [2]: the idea consists in pre-computing alternatives, and embedding in the program enough information to assist and drive a runtime system towards to the best solution.

### 3.2.2. *Software Adaptation*

More than ever, software will need to adapt to their environment. In most cases, this environment will remain unknown until runtime. This is already the case when one deploys an application to a cloud, or an “app” to mobile devices. The dilemma is the following: for maximum portability, developers should target the most general device; but for performance they would like to exploit the most recent and advanced hardware features. JIT compilers can handle the situation to some extent, but binary deployment requires dynamic binary rewriting. Our work has shown how SIMD instructions can be upgraded from SSE to AVX [3]. Many more opportunities will appear with diverse and heterogeneous processors, featuring various kinds of accelerators.

On shared hardware, the environment is also defined by other applications competing for the same computational resources. It will become increasingly important to adapt to changing runtime conditions, such as the contention of the cache memories, available bandwidth, or hardware faults. Fortunately, optimizing at runtime is also an opportunity, because this is the first time the program is visible as a whole: executable and libraries (including library versions). Optimizers may also rely on dynamic information, such as actual input data, parameter values, etc. We have already developed a software platform [14] to analyze and optimize programs at runtime, and we started working on automatic dynamic parallelization of sequential code, and dynamic specialization.

We started addressing some of these challenges in ongoing projects such as Nano2017 PSAIC Collaborative research program with STMicroelectronics, as well as within the Inria Project Lab MULTICORE. The starting H2020 FET HPC project ANTAREX will also address these challenges from the energy perspective. We will further leverage our platform and initial results to address other adaptation opportunities. Efficient software adaptation will require expertise from all domains tackled by PACAP, and strong interaction between all team members is expected.

### 3.2.3. *Research directions in uniprocessor microarchitecture*

Achieving high single-thread performance remains a major challenge even in the multicore era (Amdahl’s law). The members of the PACAP project-team have been conducting research in uniprocessor micro-architecture research for about 20 years covering major topics including caches, instruction front-end, branch prediction, out-of-order core pipeline, branch prediction and value prediction. In particular, in the recent years they have been recognized world leaders in branch prediction [19][9] and in cache prefetching [7] and they have revived the forgotten concept of value prediction [12], [11]. This research was supported by the ERC Advanced grant DAL (2011-2016) and also by Intel. We intend to pursue research on achieving ultimate uncore performance. Below are several non-orthogonal directions that we have identified for mid-term research:

1. management of the memory hierarchy (particularly the hardware prefetching);
2. practical design of very wide issue execution core;
3. speculative execution.

#### *Memory design issues:*

Performance of many applications is highly impacted by the memory hierarchy behavior. The interactions between the different components in the memory hierarchy and the out-of-order execution engine have high impact on performance.

The last *Data Prefetching Contest* held with ISCA 2015 has illustrated that achieving high prefetching efficiency is still a challenge for wide-issue superscalar processors, particularly those featuring a very large instruction window. The large instruction window enables an implicit data prefetcher. The interaction between this implicit hardware prefetcher and the explicit hardware prefetcher is still relatively mysterious as illustrated by Pierre Michaud's BO prefetcher (winner of DPC2) [7]. The first objective of the research is to better understand how the implicit prefetching enabled by the large instruction window interacts with the L2 prefetcher and then to understand how explicit prefetching on the L1 also interacts with the L2 prefetcher.

The second objective of the research is related to the interaction of prefetching and virtual/physical memory. On real hardware, prefetching is stopped by page frontiers. The interaction between TLB prefetching (and on which level) and cache prefetching must be analyzed.

The prefetcher is not the only actor in the hierarchy that must be carefully controlled. Significant benefit can also be achieved through careful management of memory access bandwidth, particularly the management of spatial locality on memory accesses, both for reads and writes. The exploitation of this locality is traditionally handled in the memory controller. However, it could be better handled if larger temporal granularity was available. Finally, we also intend to continue to explore the promising avenue of compressed caches. In particular we recently proposed the skewed compressed cache [15]. It offers new possibility for efficient compression schemes.

#### *Ultra wide-issue superscalar:*

To effectively leverage memory level parallelism, one requires huge out-of-order execution structures as well as very wide issue superscalar processor. For the two past decades, implementing always wider issue superscalar processor has been challenging. The objective of our research on the execution core is to explore (and revisit) directions to allow the design of a very wide-issue (8-to-16 way) out-of-order execution core while mastering its complexity (silicon area, hardware logic complexity, power/energy consumption).

The first direction that we intend to explore is the use of clustered architecture as in our recent work [8]. Symmetric clustered organization allows to benefit from simpler bypass network, but induce large complexity on the issue queue. One remarkable finding of our study [8] is that, when considering two large clusters (e.g. 8-wide) steering large groups of consecutive instructions (e.g. 64  $\mu$ ops) to the same cluster is quite efficient. This opens opportunities to limit the complexity of the issue queues (monitoring less buses) and register files (reducing number of ports, and number of physical registers) in the clusters, since not all results have to be forwarded to the other cluster.

The second direction that we intend to explore is associated with the approach we developed with Sembrant et al. [16]. It reduces the number of instructions waiting in the instruction queues for the applications benefiting from very large instruction windows. Instructions are dynamically classified as ready (independent from any long latency instruction) or non-ready, and as urgent (part of a dependency chain leading to a long latency instruction) or non-urgent. Non-ready non-urgent instructions can be delayed until the long latency instruction has been executed; this allows to reduce the pressure on the issue queue. This proposition opens the opportunity to consider an asymmetric microarchitecture with a cluster dedicated to the execution of urgent instructions and a second cluster executing the non-urgent instructions. The microarchitecture of this second cluster could be optimized to reduce complexity and power consumption (smaller instruction queue, less aggressive scheduling...)

#### *Speculative execution.*

Out-of-order (OoO) execution relies on speculative execution that requires predictions of all sorts: branch, memory dependency, value...

The PACAP members have been major actors of the branch prediction research for the last 20 years; and their proposals have influenced the design of most of the hardware branch predictors in current microprocessors. We will continue to steadily explore new branch predictor designs as for instance [18].

In speculative execution, we have recently revisited value prediction (VP) which was a hot research topic between 1996 and 2002. However it was considered up to recently that value prediction would lead to a huge increase in complexity and power consumption in every stage of the pipeline. Fortunately, we have recently shown that complexity usually introduced by value prediction in the OoO engine can be overcome [12], [11], [19], [9]. First, very high accuracy can be enforced at reasonable cost in coverage and minimal complexity [12]. Thus, both prediction validation and recovery by squashing can be done outside the out-of-order engine, at commit time. Furthermore, we propose a new pipeline organization, EOLE ({Early | Out-of-order | Late} Execution), that leverages VP with validation at commit to execute many instructions outside the OoO core, in-order [11]. With EOLE, the issue-width in OoO core can be reduced without sacrificing performance, thus benefiting the performance of VP without a significant cost in silicon area and/or energy. In the near future, we will explore new avenues related to value prediction. These directions include register equality prediction and compatibility of value prediction with weak memory models in multiprocessors.

### 3.2.4. Towards heterogeneous single-ISA CPU-GPU architectures

Heterogeneous single-ISA architectures have been proposed in the literature during the 2000's [56] and are now widely used in the industry (ARM big.LITTLE, NVIDIA 4+1...) as a way to improve power-efficiency in mobile processors. These architectures include multiple cores whose respective microarchitectures offer different trade-offs between performance and energy efficiency, or between latency and throughput, while offering the same interface to software. Dynamic task migration policies leverage the heterogeneity of the platform by using the most suitable core for each application, or even each phase of processing. However, these works only tune cores by changing their complexity. Energy-optimized cores are either identical cores implemented in a low-power process technology, or simplified in-order superscalar cores, which are far from state-of-the-art throughput-oriented architectures such as GPUs.

We propose to investigate the convergence of CPU and GPU at both architecture and compilation levels.

#### *Architecture.*

The architecture convergence between Single Instruction Multiple Threads (SIMT) GPUs and multicore processors that we have been pursuing [36] opens the way for heterogeneous architectures including latency-optimized superscalar cores and throughput-optimized GPU-style cores, which all share the same instruction set. Using SIMT cores in place of superscalar cores will enable the highest energy efficiency on regular sections of applications. As with existing single-ISA heterogeneous architectures, task migration will not necessitate any software rewrite and will accelerate existing applications.

#### *Compilers for emerging heterogeneous architectures.*

Single-ISA CPU+GPU architectures will provide the necessary substrate to enable efficient heterogeneous processing. However, it will also introduce substantial challenges at the software and firmware level. Task placement and migration will require advanced policies that leverage both static information at compile time and dynamic information at run-time. We are tackling the heterogeneous task scheduling problem at the compiler level. As a first step, we are prototyping scheduling algorithms on existing multiple-ISA CPU+GPU architectures like NVIDIA Tegra X1.

### 3.2.5. Real-time systems

Safety-critical systems (e.g. avionics, medical devices, automotive...) have so far used simple uncore hardware systems as a way to control their predictability, in order to meet timing constraints. Still, many critical embedded systems have increasing demand in computing power, and simple uncore processors are not sufficient anymore. General-purpose multicore processors are not suitable for safety-critical real-time systems, because they include complex micro-architectural elements (cache hierarchies, branch, stride and value

predictors) meant to improve average-case performance, and for which worst-case performance is difficult to predict. The prerequisite for calculating tight WCET is a deterministic hardware system that avoids dynamic, time-unpredictable calculations at run-time.

Even for multi and manycore systems designed with time-predictability in mind (Kalray MPPA manycore architecture <sup>0</sup>, or the Recore manycore hardware <sup>0</sup>) calculating WCETs is still challenging. The following two challenges will be addressed in the mid-term:

1. definition of methods to estimate WCETs tightly on manycores, that smartly analyzes and/or controls shared resources such as buses, NoCs or caches;
2. methods to improve the programmability of real-time applications through automatic parallelization and optimizations from model-based designs.

### 3.2.6. Fault Tolerance

Technology trends suggest that, in tomorrow's computing world, failures will become commonplace due to many factors, and the expected probability of failure will increase with scaling. While well-known approaches, such as error correcting codes, exist to recover from failures and provide fault-free chips, the exponential growth of the number of faults will make them unaffordable in the future. Consequently, other approaches such as fine-grained disabling and reconfiguration of hardware elements (e.g. individual functional units or cache blocks) will become economically necessary. We are going to enter a new era: functionally correct chips with variable performance among chips and throughout their lifetime [58].

Transient and permanent faults may be detected by similar techniques, but correcting them generally involves different approaches. We are primarily interested in permanent faults, even though we do not necessarily disregard transient faults (e.g. the TMR approach in the next paragraph addresses both kind of faults).

#### *CPU.*

Permanent faults can occur anywhere in the processor. The performance implications of faulty cells vary depending on how the array is used in a processor. Most of micro-architectural work aiming at assessing the performance implications of permanently faulty cells relies on simulations with random fault-maps. These studies are, therefore, limited by the fault-maps they use that may not be representative for the average and distributed performance. They also do not consider aging effect.

Considering the memory hierarchy, we have already studied [5] the impact of permanent faults on the average and worst-case performance based on analytical models. We will extend these models to cover other components and other designs, and to analyze the interaction between faulty components.

For identified critical hardware structures, such as the memory hierarchy, we will propose protection mechanisms by for instance using larger cells, or even by selecting a different array organization to mitigate the impact of faults.

Another approach to deal with faults is to introduce redundancy at the code level. We propose to consider static compilation techniques focusing on existing hardware. As an example, we plan to leverage SIMD extensions of current instruction sets to introduce redundancy in scalar code at minimum cost. With these instructions, it will be possible to protect the execution from both soft errors by using TMR (triple modular redundancy) with voters in the code itself, and permanent faults without the need of extra hardware support to deconfigure faulty functional units.

#### *Reconfigurable Computing.*

---

<sup>0</sup><http://www.kalrayinc.com>

<sup>0</sup><http://www.recoresystems.com/>

In collaboration with the CAIRN project-team, we propose to construct Coarse Grain Reconfigurable Architectures (CGRA) from a sea of basic arithmetic and memory elements organized into clusters and connected through a hierarchical interconnection network. These clusters of basic arithmetic operators (e.g. 8-bit arithmetic and logic units) would be able to be seamlessly configured to various accuracy and data types to adapt the consumed energy to application requirements taking advantage of approximate computations. We propose to add new kinds of error detection (and sometimes correction) directly at the operator level by taking advantage of the massive redundancy of the array. As an example, errors can be tracked and detected in a complex sequence of double floating-point operations by using a reduced-precision version of the same processing.

Such reconfigurable blocks will be driven by compilation techniques, in charge of computing checkpoints, detecting faults, and replaying computations when needed.

Dynamic compilation techniques will help better exploit faulty hardware, by allocating data and computations on correct resources. In case of permanent faults, we will provide a mechanism to reconfigure the hardware, for example by reducing the issue width of VLIW processors implemented in CGRA. Dynamic code generation (JIT compiler) will re-generate code for the new configuration, guaranteeing portability and optimal exploitation of the hardware.

### 3.2.7. Power efficiency

PACAP will address power-efficiency at several levels. First, we will design static and split compilation techniques to contribute to the race for Exascale computing (the general goal is to reach  $10^{18}$  FLOP/s at less than 20 MW). Second, we will focus on high-performance low-power embedded compute nodes. We will research new static and dynamic compilation techniques that fully exploit emerging memory and NoC technologies. Finally, in collaboration with the CAIRN project-team, we will investigate the synergy of reconfigurable computing and dynamic code generation.

#### *Green and heterogeneous high-performance computing.*

Concerning HPC systems, our approach consists in mapping, runtime managing and autotuning applications for green and heterogeneous High-Performance Computing systems up to the Exascale level. One key innovation of the proposed approach consists of introducing a separation of concerns (where self-adaptivity and energy efficient strategies are specified aside to application functionalities) promoted by the definition of a Domain Specific Language (DSL) inspired by aspect-oriented programming concepts for heterogeneous systems. The new DSL will be introduced for expressing adaptivity/energy/performance strategies and to enforce at runtime application autotuning and resource and power management. The goal is to support the parallelism, scalability and adaptability of a dynamic workload by exploiting the full system capabilities (including energy management) for emerging large-scale and extreme-scale systems, while reducing the Total Cost of Ownership (TCO) for companies and public organizations.

#### *High-performance low-power embedded compute nodes.*

We will address the design of next generation energy-efficient high-performance embedded compute nodes. It focuses at the same time on software, architecture and emerging memory and communication technologies in order to synergistically exploit their corresponding features. The approach of the project is organized around three complementary topics: 1) compilation techniques; 2) multicore architectures; 3) emerging memory and communication technologies. PACAP will focus on the compilation aspects, taking as input the software-visible characteristics of the proposed emerging technology, and making the best possible use of the new features (non-volatility, density, endurance, low-power).

#### *Hardware Accelerated JIT Compilation.*

Reconfigurable hardware offers the opportunity to limit power consumption by dynamically adjusting the number of available resources to the requirements of the running software. In particular, VLIW processors can adjust the number of available issue lanes. Unfortunately, changing the processor width often requires recompiling the application, and VLIW processors are highly dependent of the quality of the compilation, mainly because of the instruction scheduling phase performed by the compiler. Another challenge lies in the high constraints of the embedded system: the energy and execution time overhead due to the JIT compilation must be carefully kept under control.

We started exploring ways to reduce the cost of JIT compilation targeting VLIW-based heterogeneous many-core systems. Our approach lies on a hardware/software JIT compiler framework. While basic optimizations and JIT management are performed in software, the compilation back-end is implemented by means of specialized hardware. This back-end involves both instruction scheduling and register allocation, which are known to be the most time-consuming stages of such a compiler.

### 3.2.8. Security

Security is a mandatory concern of any modern computing system. Various threat models have led to a multitude of protection solutions. ALF already has contributions, thanks to the HAVEGE [62] random number generator, and code obfuscating techniques (the obfuscating just-in-time compiler [55], or thread-based control flow mangling [60]).

We plan to partner with security experts who can provide intuition, know-how and expertise, in particular in defining threat models, and assessing the quality of the solutions. Our background in compilation and architecture will help design more efficient and less expensive protection mechanisms.

We already have ongoing research directions related to security. We also plan to partner with the Inria/CentraleSupélec CIDRE project-team to design a tainting technique based on a just-in-time compiler.

#### *Compiler-based data protection.*

We will specify and design error correction codes suitable for an efficient protection of sensitive information in the context of Internet of Things (IoT) and connected objects. We will partner with experts in security and codes to prototype a platform that demonstrates resilient software. PACAP's expertise will be key to select and tune the protection mechanisms developed within the project, and to propose safe, yet cost-effective solutions from an implementation point of view.

#### *JIT-based tainting.*

Dynamic information flow control (DIFC, also known as *tainting*) is used to detect intrusions and to identify vulnerabilities. It consists in attaching metadata (called *taints* or *labels*) to information containers, and to propagate the taints when particular operations are applied to the containers: reads, writes, etc. The goal is then to guarantee that confidential information is never used to generate data sent to an untrusted container; conversely, data produced by untrusted entities cannot be used to update sensitive data.

The containers can be of various granularities: fine-grain approaches can deal with single variables, coarser-grain approaches consider a file as a whole. The CIDRE project-team has developed several DIFC monitors. kBlare is coarse-grain monitor in the Linux kernel. JBlare is a fine-grain monitor for Java applications. Fine-grain monitors provide a better precision at the cost of a significant overhead in execution time.

We propose to combine the expertise of CIDRE in DIFC with our expertise in JIT compilation to design hybrid approaches. An initial static analysis of the program prior to installation or execution will feed information to a dynamic analyzer that propagates taints during just-in-time compilation.

## TASC Project-Team

### 3. Research Program

#### 3.1. Overview

Basic research is guided by the challenges raised before: to classify and enrich the models, to automate reformulation and resolution, to dissociate declarative and procedural knowledge, to come up with theories and tools that can handle problems involving both continuous and discrete variables, to develop modelling tools and to come up with solving tools that scale well. On the one hand, **classification aspects** of this research are integrated within a knowledge base about combinatorial problem solving: the global constraint catalog (see <http://sofdem.github.io/gccat/>). On the other hand, **solving aspects** are capitalized within the constraint solving system **CHOCO**. Lastly, within the framework of its activities of valorisation, teaching and of partnership research, the team uses constraint programming for solving various concrete problems. The challenge is, on one side to increase the visibility of the constraints in the others disciplines of computer science, and on the other side to contribute to a broader diffusion of the constraint programming in the industry.

#### 3.2. Fundamental Research Topics

This part presents the research topics investigated by the project:

- Global Constraints Classification, Reformulation and Filtering,
- Convergence between Discrete and Continuous,
- Dynamic, Interactive and over Constrained Problems,
- Solvers.

These research topics are in fact not independent. The work of the team thus frequently relates transverse aspects such as explained global constraints, Benders decomposition and explanations, flexible and dynamic constraints, linear models and relaxations of constraints.

##### 3.2.1. *Constraints Classification, Reformulation and Filtering*

In this context our research is focused (a) first on identifying recurring combinatorial structures that can be used for modelling a large variety of optimization problems, and (b) exploit these combinatorial structures in order to come up with efficient algorithms in the different fields of optimization technology. The key idea for achieving point (b) is that many filtering algorithms both in the context of Constraint Programming, Mathematical Programming and Local Search can be interpreted as the maintenance of invariants on specific domains (e.g., graph, geometry). The systematic classification of **global constraints** and of their relaxation brings a synthetic view of the field. It establishes links between the properties of the concepts used to describe constraints and the properties of the constraints themselves. Together with **SICS**, the team develops and maintains *a catalog of global constraints*, which describes the semantics of more than 431 constraints, and proposes a unified mathematical model for expressing them. This model is based on graphs, automata and logic formulae and allows to derive filtering methods and automatic reformulation for each constraint in a unified way (see <http://www.emn.fr/x-info/sdemasse/gccat/index.html>). We consider hybrid methods (i.e., methods that involve more than one optimization technology such as constraint programming, mathematical programming or local search), to draw benefit from the respective advantages of the combined approaches. More fundamentally, the study of hybrid methods makes it possible to compare and connect strategies of resolution specific to each approach for then conceiving new strategies. Beside the works on classical, complete resolution techniques, we also investigate local search techniques from a mathematical point of view. These partly random algorithms have been proven very efficient in practice, although we have little theoretical knowledge on their behaviour, which often makes them problem-specific. Our research in that area is focused on a probabilistic model of local search techniques, from which we want to derive quantified information on their behaviour, in order to

use this information directly when designing the algorithms and exploit their performances better. We also consider algorithms that maintain local and global consistencies, for more specific models. Having in mind the trade off between genericity and effectiveness, the effort is put on the efficiency of the algorithms with guarantee on the produced levels of filtering. This effort results in adapting existing techniques of resolution such as graph algorithms. For this purpose we identify necessary conditions of feasibility that can be evaluated by efficient incremental algorithms. Genericity is not neglected in these approaches: on the one hand the constraints we focus on are applicable in many contexts (for example, graph partitioning constraints can be used both in logistics and in phylogeny); on the other hand, this work led to study the portability of such constraints and their independence with specific solvers. This research orientation gathers various work such as strong local consistencies, graph partitioning constraints, geometrical constraints, and optimization and soft constraints. Within the perspective to deal with complex industrial problems, we currently develop meta constraints (e.g. *geost*) handling all together the issues of large-scale problems, dynamic constraints, combination of spatial and temporal dimensions, expression of business rules described with first order logic.

### 3.2.2. *Convergence between Discrete and Continuous*

Many industrial problems mix continuous and discrete aspects that respectively correspond to physical (e.g., the position, the speed of an object) and logical (e.g., the identifier, the nature of an object) elements. Typical examples of problems are for instance:

- *Geometrical placement problems* where one has to place in space a set of objects subject to various geometrical constraints (i.e., non-overlapping, distance). In this context, even if the positions of the objects are continuous, the structure of optimal configurations has a discrete nature.
- *Trajectory and mission planning problems* where one has to plan and synchronize the moves of several teams in order to achieve some common goal (i.e., fire fighting, coordination of search in the context of rescue missions, surveillance missions of restricted or large areas).
- *Localization problems in mobile robotic* where a robot has to plan alone (only with its own sensors) its trajectory. This kind of problematic occurs in situations where the GPS cannot be used (e.g., under water or Mars exploration) or when it is not precise enough (e.g., indoor surveillance, observation of contaminated sites).

Beside numerical constraints that mix continuous and integer variables we also have global constraints that involve both type of variables. They typically correspond to graph problems (i.e., graph colouring, domination in a graph) where a graph is dynamically constructed with respect to geometrical and-or temporal constraints. In this context, the key challenge is avoiding decomposing the problem in a discrete and continuous parts as it is traditionally the case. As an illustrative example consider *the wireless network deployment problem*. On the one hand, the continuous part consists of finding out where to place a set of antenna subject to various geometrical constraints. On the other hand, by building an interference graph from the positions of the antenna, the discrete part consists of allocating frequencies to antenna in order to avoid interference. In the context of convergence between discrete and continuous variables, our goals are:

- First to identify and compare typical class of techniques that are used in the context of continuous and discrete solvers.
- To see how one can unify and/or generalize these techniques in order to handle in an integrated way continuous and discrete constraints within the same framework.

### 3.2.3. *Dynamic, Interactive and over Constrained Problems*

Some industrial applications are defined by a set of constraints which may change over time, for instance due to an interaction with the user. Many other industrial applications are over-constrained, that is, they are defined by set of constraints which are more or less important and cannot be all satisfied at the same time. Generic, dedicated and explanation-based techniques can be used to deal efficiently with such applications. Especially, these applications rely on the notion of *soft constraints* that are allowed to be (partially) violated. The generic concept that captures a wide variety of soft constraints is the violation measure, which is coupled with specific resolution techniques. Lastly, soft constraints allow to combine the expressive power of global constraints with local search frameworks.



### 3.2.4. Solvers

- *Discrete solver* Our theoretical work is systematically validated by concrete experimentations. We have in particular for that purpose the **CHOCO** constraint platform. The team develops and maintains **CHOCO** initially with the assistance of the laboratory e-lab of Bouygues (G. Rochart), the company Amadeus (F. Laburthe), and others researchers such as **N. Jussien** and **H. Cambazard** (4C, INP Grenoble). Since 2008 the main developments are done by **Charles Prud'homme** and **Xavier Lorca**. The functionalities of **CHOCO** are gradually extended with the outcomes of our works: design of constraints, analysis and visualization of explanations, etc. The open source **CHOCO** library is downloaded on average 450 times each month since 2006. **CHOCO** is developed in line with the research direction of the team, in an open-minded scientific spirit. Contrarily to other solvers where the efficiency often relies on problem-specific algorithms, **CHOCO** aims at providing the users both with reusable techniques (based on an up-to-date implementation of the [global constraint catalogue](#)) and with a variety of tools to ease the use of these techniques (clear separation between model and resolution, event-based solver, management of the over-constrained problems, explanations, etc.).
- *Discrete continuous* We use discrete convexity to describe filtering for families of constraints: We introduce a propagator for pairs of Sum constraints, where the expressions in the sums respect a form of convexity. This propagator is parametric and can be instantiated for various concrete pairs, including Deviation, Spread, and the conjunction of  $\text{Linear} \leq$  and Among. We show that despite its generality, our propagator is competitive in theory and practice with state-of-the-art propagators.
- *Constraint programming and verification* Constraint Programming has already had several applications to verification problems. It also has many common ideas with Abstract Interpretation, a theory of approximation of the semantics of programs. In both cases, we are interested in a particular set (solutions in CP, program traces in semantics), which is hard or impossible to compute, and this set is replaced by an over-approximation (consistent domains / abstract domains). Previous works (internship of Julie Laniau, PhD of **Marie Pelleau**, collaboration with the Abstract Interpretation team at the ENS and **Antoine Miné** in particular) have exhibited some of these links, and identified some situations where the two fields, Abstract Interpretation and Constraint Programming, can complement each other. It is the case in real-time stream processing languages, where Abstract Interpretation techniques may not be precise enough when analyzing loops. With the PhD of **Anicet Bart**, we are currently working on using CP techniques to find loop invariants for the **Faust language**, a functional sound processing language.

This work around the design and the development of solvers thus forms the fourth direction of basic research of the project.

## AOSTE Project-Team

### 3. Research Program

#### 3.1. Models of Computation and Communication (MoCCs)

**Participants:** Julien Deantoni, Robert de Simone, Frédéric Mallet, Dumitru Potop Butucaru.

Esterel, SyncCharts, synchronous formalisms, Process Networks, Marked Graphs, Kahn networks, compilation, synthesis, formal verification, optimization, allocation, refinement, scheduling

Formal Models of Computation form the basis of our approach to Embedded System Design. Because of the growing importance of communication handling, it is now associated with the name, MoCC in short. The appeal of MoCCs comes from the fact that they combine features of mathematical models (formal analysis, transformation, and verification) with these of executable specifications (close to code level, simulation, and implementation). Examples of MoCCs in our case are mainly synchronous reactive formalisms and dataflow process networks. Various extensions or specific restrictions enforce respectively greater expressivity or more focused decidable analysis results.

DataFlow Process Networks and Synchronous Reactive Languages such as ESTEREL/SYNCHARTS and SIGNAL/POLYCHRONY [54], [55], [49], [15], [4], [13] share one main characteristics: they are specified in a self-timed or loosely timed fashion, in the asynchronous data-flow style. But formal criteria in their semantics ensure that, under good correctness conditions, a sound synchronous interpretation can be provided, in which all treatments (computations, signaling communications) are precisely temporally mapped. This is referred to as clock calculus in synchronous reactive systems, and leads to a large body of theoretical studies and deep results in the case of DataFlow Process Networks [50], [48] (consider SDF balance equations for instance [56]).

As a result, explicit schedules become an important ingredient of design, which ultimately can be considered and handled by the designer him/herself. In practice such schedules are sought to optimize other parts of the design, mainly buffering queues: production and consumption of data can be regulated in their relative speeds. This was specially taken into account in the recent theories of Latency-Insensitive Design [51], or N-synchronous processes [52], with some of our contributions [6].

Explicit schedule patterns should be pictured in the framework of low-power distributed mapping of embedded applications onto manycore architectures, where they could play an important role as theoretical formal models on which to compute and optimize allocations and performances. We describe below two lines of research in this direction. Striking in these techniques is the fact that they include time and timing as integral parts of early functional design. But this original time is logical, multiform, and only partially ordering the various functional computations and communications. This approach was radically generalized in our team to a methodology for logical time based design, described next (see 3.2).

##### 3.1.1. K-periodic static scheduling and routing in Process Networks

In the recent years we focused on the algorithm treatments of ultimately k-periodic schedule regimes, which are the class of schedules obtained by many of the theories described above. An important breakthrough occurred when realizing that the type of ultimately periodic binary words that were used for reporting *static scheduling* results could also be employed to record a completely distinct notion of ultimately k-periodic route switching patterns, and furthermore that commonalities of representation could ease combine them together. A new model, by the name of K-periodical Routed marked Graphs (KRG) was introduced, and extensively studied for algebraic and algorithmic properties [5].

The computations of optimized static schedules and other optimal buffering configurations in the context of latency-insensitive design led to the K-Passa software tool development (now terminated)

### 3.1.2. Endochrony and GALS implementation of conflict-free polychronous programs

The possibility of exploring various schedulings for a given application comes from the fact that some behaviors are truly concurrent, and mutually *conflict-free* (so they can be executed independently, with any choice of ordering). Discovering potential asynchronous inside synchronous reactive specifications then becomes something highly desirable. It can benefit to potential distributed implementation, where signal communications are restricted to a minimum, as they usually incur loss in performance and higher power consumption. This general line of research has come to be known as Endochrony, with some of our contributions [11].

## 3.2. Logical Time in Model-Driven Embedded System Design

**Participants:** Julien Deantoni, Frédéric Mallet, Marie Agnes Peraldi Frati, Robert de Simone.

Starting from specific needs and opportunities for formal design of embedded systems as learned from our work on MoCCs (see 3.1 ), we developed a Logical Time Model as part of the official **OMG UML profile MARTE** for Modeling and Analysis of Real-Time Embedded systems. With this model is associated a Clock Constraint Specification Language (CCSL), which allows to provide loose or strict logical time constraints between design ingredients, be them computations, communications, or any kind of events whose repetitions can be conceived as generating a logical conceptual clock (or activation condition). The definition of CCSL is provided in [1].

Our vision is that many (if not all) of the timing constraints generally expressed as physical prescriptions in real-time embedded design (such as periodicity, sporadicity) could be expressed in a logical setting, while actually many physical timing values are still unknown or unspecified at this stage. On the other hand, our logical view may express much more, such as loosely stated timing relations based on partial orderings or partial constraints.

So far we have used CCSL to express important phenomena as present in several formalisms: **AADL** (used in avionics domain), **EAST-ADL2** (proposed for the **AutoSar** automotive electronic design approach), **IP-Xact** (for System-on-Chip (*SoC*) design). The difference here comes from the fact that these formalisms were formerly describing such issues in informal terms, while CCSL provides a dedicated formal mathematical notation. Close connections with synchronous and polychronous languages, especially Signal, were also established; so was the ability of CCSL to model dataflow process network static scheduling.

In principle the MARTE profile and its Logical Time Model can be used with any UML editor supporting profiles. It has also evolved to become a Domain-Specific Language, independent of UML. It is connected to the **CAPELLA** environment, and the **PAPYRUS** open-source editor. We developed under Eclipse the **TIMESQUARE** solver and emulator for CCSL constraints (see 5.6 ), with its own graphical interface, as a stand-alone software module, again now coupled with MARTE and Papyrus, but also as part of the **GeMoC studio** environment developed in the GeMoC ANR project.

The MARTE profile and its Logical Time Model can be used with any UML editor supporting profiles but evolved to become a DSL independent of UML. We developed as a set of eclipse plugins the TIMESQUARE tool to edit and simulate CCSL specifications. TimeSquare has been coupled with various tools like Papyrus or Capella and is now part of the concurrent solver integrated in the GEMOC studio.

While CCSL constraints may be introduced as part of the intended functionality, some may also be extracted from requirements imposed either from real-time user demands, or from the resource limitations and features from the intended execution platform. Sophisticated detailed descriptions of platform architectures are allowed using MARTE, as well as formal allocations of application operations (computations and communications) onto platform resources (processors and interconnects). This is of course of great value at a time where embedded architectures are becoming more and more heterogeneous and parallel or distributed, so that application mapping in terms of spatial allocation and temporal scheduling becomes harder and harder. This approach is extensively supported by the MARTE profile and its various models. As such it originates from the Application-Architecture-Adequation (AAA) methodology, first proposed by Yves Sorel, member of Aoste. AAA aims at specific distributed real-time algorithmic methods, described next in 3.3 .

Of course, while logical time in design is promoted here, and our works show how many current notions used in real-time and embedded systems synthesis can naturally be phrased in this model, there will be in the end a phase of validation of the logical time assumptions (as is the case in synchronous circuits and SoC design with timing closure issues). This validation is usually conducted from Worst-Case Execution Time (WCET) analysis on individual components, which are then used in further analysis techniques to establish the validity of logical time assumptions (as partial constraints) asserted during the design.

### 3.3. The AAA (Algorithm-Architecture Adequation) methodology and Real-Time Scheduling

**Participants:** Liliana Cucu, Laurent George, Dumitru Potop Butucaru, Yves Sorel.

Note: The AAA methodology and the SynDEX environment are fully described at <http://www.syndex.org/>, together with [relevant publications](#).

#### 3.3.1. Algorithm-Architecture Adequation

The AAA methodology relies on distributed real-time scheduling and relevant optimization to connect an Algorithm/Application model to an Architectural one. We now describe its premises and benefits.

The Algorithm model is an extension of the well known data-flow model from Dennis [53]. It is a directed acyclic hyper-graph (DAG) that we call “conditioned factorized data dependence graph”, whose vertices are “operations” and hyper-edges are directed “data or control dependences” between operations. The data dependences define a partial order on the operations execution. The basic data-flow model was extended in three directions: first infinite (resp. finite) repetition of a sub-graph pattern in order to specify the reactive aspect of real-time systems (resp. in order to specify the finite repetition of a sub-graph consuming different data similar to a loop in imperative languages), second “state” when data dependences are necessary between different infinite repetitions of the sub-graph pattern introducing cycles which must be avoided by introducing specific vertices called “delays” (similar to  $z^{-n}$  in automatic control), third “conditioning” of an operation by a control dependence similar to conditional control structure in imperative languages, allowing the execution of alternative subgraphs. Delays combined with conditioning allow the programmer to specify automata necessary for describing “mode changes”.

The Architecture model is a directed graph, whose vertices are of two types: “processor” (one sequencer of operations and possibly several sequencers of communications) and “medium” (support of communications), and whose edges are directed connections.

The resulting implementation model [9] is obtained by an external compositional law, for which the architecture graph operates on the algorithm graph. Thus, the result of such compositional law is an algorithm graph, “architecture-aware”, corresponding to refinements of the initial algorithm graph, by computing spatial (distribution) and timing (scheduling) allocations of the operations onto the architecture graph resources. In that context “Adequation” refers to some search amongst the solution space of resulting algorithm graphs, labelled by timing characteristics, for one algorithm graph which verifies timing constraints and optimizes some criteria, usually the total execution time and the number of computing resources (but other criteria may exist). The next section describes distributed real-time schedulability analysis and optimization techniques for that purpose.

#### 3.3.2. Distributed Real-Time Scheduling and Optimization

We address two main issues: uniprocessor and multiprocessor real-time scheduling where constraints must mandatorily be met, otherwise dramatic consequences may occur (hard real-time) and where resources must be minimized because of embedded features.

In the case of uniprocessor real-time scheduling, besides the classical deadline constraint, often equal to a period, we take into consideration dependences between tasks and several, latencies. The latter are complex related “end-to-end” constraints. Dealing with multiple real-time constraints raises the complexity of the scheduling problems. Moreover, because the preemption leads, at least, to a waste of resources due to its approximation in the WCET (Worst Execution Time) of every task, as proposed by Liu and Leyland [57], we first studied non-preemptive real-time scheduling with dependences, periodicities, and latencies constraints. Although a bad approximation of the preemption cost, may have dramatic consequences on real-time scheduling, there are only few researches on this topic. We have been investigating preemptive real-time scheduling since few years, and we focus on the exact cost of the preemption. We have integrated this cost in the schedulability conditions that we propose, and in the corresponding scheduling algorithms. More generally, we are interested in integrating in the schedulability analyses the cost of the RTOS (Real-Time Operating System), for which the cost of preemption is the most difficult part because it varies according to the instance (job) of each task.

In the case of multiprocessor real-time scheduling, we chose at the beginning the partitioned approach, rather than the global approach, since the latter allows task migrations whose cost is prohibitive for current commercial processors. The partitioned approach enables us to reuse the results obtained in the uniprocessor case in order to derive solutions for the multiprocessor case. We consider also the semi-partitioned approach which allows only some migrations in order to minimize the overhead they involve. In addition to satisfy the multiple real-time constraints mentioned in the uniprocessor case, we have to minimize the total execution time (makespan) since we deal with automatic control applications involving feedback loops. Furthermore, the domain of embedded systems leads to solving minimization resources problems. Since these optimization problems are NP-hard we develop exact algorithms (B & B, B & C) which are optimal for simple problems, and heuristics which are sub-optimal for realistic problems corresponding to industrial needs. Long time ago we proposed a very fast “greedy” heuristics [8] whose results were regularly improved, and extended with local neighborhood heuristics, or used as initial solutions for metaheuristics.

In addition to the spatial dimension (distributed) of the real-time scheduling problem, other important dimensions are the type of communication mechanisms (shared memory vs. message passing), or the source of control and synchronization (event-driven vs. time-triggered). We explore real-time scheduling on architectures corresponding to all combinations of the above dimensions. This is of particular impact in application domains such as automotive and avionics (see 4.3).

The arrival of complex hardware responding to the increasing demand for computing power in next generation systems exacerbates the limitations of the current worst-case real-time reasoning. Our solution to overcome these limitations is based on the fact that worst-case situations may have a extremely low probability of appearance within one hour of functioning ( $10^{-45}$ ), compared to the certification requirements for instance ( $10^{-9}$  for the highest level of certification in avionics). Thus we model and analyze the real-time systems using probabilistic models and we propose results that are fundamental for the probabilistic worst-case reasoning over a given time window.

## CONVECS Project-Team

### 3. Research Program

#### 3.1. New Formal Languages and their Concurrent Implementations

We aim at proposing and implementing new formal languages for the specification, implementation, and verification of concurrent systems. In order to provide a complete, coherent methodological framework, two research directions must be addressed:

- *Model-based specifications*: these are operational (i.e., constructive) descriptions of systems, usually expressed in terms of processes that execute concurrently, synchronize together and communicate. Process calculi are typical examples of model-based specification languages. The approach we promote is based on LOTOS NT (LNT for short), a formal specification language that incorporates most constructs stemming from classical programming languages, which eases its acceptance by students and industry engineers. LNT [29] is derived from the ISO standard E-LOTOS (2001), of which it represents the first successful implementation, based on a source-level translation from LNT to the former ISO standard LOTOS (1989). We are working both on the semantic foundations of LNT (enhancing the language with module interfaces and timed/probabilistic/stochastic features, compiling the  $m$  among  $n$  synchronization, etc.) and on the generation of efficient parallel and distributed code. Once equipped with these features, LNT will enable formally verified asynchronous concurrent designs to be implemented automatically.
- *Property-based specifications*: these are declarative (i.e., non-constructive) descriptions of systems, which express *what* a system should do rather than *how* the system should do it. Temporal logics and  $\mu$ -calculi are typical examples of property-based specification languages. The natural models underlying value-passing specification languages, such as LNT, are Labeled Transition Systems (LTSs or simply *graphs*) in which the transitions between states are labeled by actions containing data values exchanged during handshake communications. In order to reason accurately about these LTSs, temporal logics involving data values are necessary. The approach we promote is based on MCL (*Model Checking Language*) [51], which extends the modal  $\mu$ -calculus with data-handling primitives, fairness operators encoding generalized Büchi automata, and a functional-like language for describing complex transition sequences. We are working both on the semantic foundations of MCL (extending the language with new temporal and hybrid operators, translating these operators into lower-level formalisms, enhancing the type system, etc.) and also on improving the MCL on-the-fly model checking technology (devising new algorithms, enhancing ergonomomy by detecting and reporting vacuity, etc.).

We address these two directions simultaneously, yet in a coherent manner, with a particular focus on applicable concurrent code generation and computer-aided verification.

#### 3.2. Parallel and Distributed Verification

Exploiting large-scale high-performance computers is a promising way to augment the capabilities of formal verification. The underlying problems are far from trivial, making the correct design, implementation, fine-tuning, and benchmarking of parallel and distributed verification algorithms long-term and difficult activities. Sequential verification algorithms cannot be reused as such for this task: they are inherently complex, and their existing implementations reflect several years of optimizations and enhancements. To obtain good speedup and scalability, it is necessary to invent new parallel and distributed algorithms rather than to attempt a parallelization of existing sequential ones. We seek to achieve this objective by working along two directions:

- *Rigorous design:* Because of their high complexity, concurrent verification algorithms should themselves be subject to formal modeling and verification, as confirmed by recent trends in the certification of safety-critical applications. To facilitate the development of new parallel and distributed verification algorithms, we promote a rigorous approach based on formal methods and verification. Such algorithms will be first specified formally in LNT, then validated using existing model checking algorithms of the CADP toolbox. Second, parallel or distributed implementations of these algorithms will be generated automatically from the LNT specifications, enabling them to be experimented on large computing infrastructures, such as clusters and grids. As a side-effect, this “bootstrapping” approach would produce new verification tools that can later be used to self-verify their own design.
- *Performance optimization:* In devising parallel and distributed verification algorithms, particular care must be taken to optimize performance. These algorithms will face concurrency issues at several levels: grids of heterogeneous clusters (architecture-independence of data, dynamic load balancing), clusters of homogeneous machines connected by a network (message-passing communication, detection of stable states), and multi-core machines (shared-memory communication, thread synchronization). We will seek to exploit the results achieved in the parallel and distributed computing field to improve performance when using thousands of machines by reducing the number of connections and the messages exchanged between the cooperating processes carrying out the verification task. Another important issue is the generalization of existing LTS representations (explicit, implicit, distributed) in order to make them fully interoperable, such that compilers and verification tools can handle these models transparently.

### 3.3. Timed, Probabilistic, and Stochastic Extensions

Concurrent systems can be analyzed from a *qualitative* point of view, to check whether certain properties of interest (e.g., safety, liveness, fairness, etc.) are satisfied. This is the role of functional verification, which produces Boolean (yes/no) verdicts. However, it is often useful to analyze such systems from a *quantitative* point of view, to answer non-functional questions regarding performance over the long run, response time, throughput, latency, failure probability, etc. Such questions, which call for numerical (rather than binary) answers, are essential when studying the performance and dependability (e.g., availability, reliability, etc.) of complex systems.

Traditionally, qualitative and quantitative analyzes are performed separately, using different modeling languages and different software tools, often by distinct persons. Unifying these separate processes to form a seamless design flow with common modeling languages and analysis tools is therefore desirable, for both scientific and economic reasons. Technically, the existing modeling languages for concurrent systems need to be enriched with new features for describing quantitative aspects, such as probabilities, weights, and time. Such extensions have been well-studied and, for each of these directions, there exist various kinds of automata, e.g., discrete-time Markov chains for probabilities, weighted automata for weights, timed automata for hard real-time, continuous-time Markov chains for soft real-time with exponential distributions, etc. Nowadays, the next scientific challenge is to combine these individual extensions altogether to provide even more expressive models suitable for advanced applications.

Many such combinations have been proposed in the literature, and there is a large amount of models adding probabilities, weights, and/or time. However, an unfortunate consequence of this diversity is the confuse landscape of software tools supporting such models. Dozens of tools have been developed to implement theoretical ideas about probabilities, weights, and time in concurrent systems. Unfortunately, these tools do not interoperate smoothly, due both to incompatibilities in the underlying semantic models and to the lack of common exchange formats.

To address these issues, CONVECS follows two research directions:

- *Unifying the semantic models.* Firstly, we will perform a systematic survey of the existing semantic models in order to distinguish between their essential and non-essential characteristics, the goal being to propose a unified semantic model that is compatible with process calculi techniques for specifying and verifying concurrent systems. There are already proposals for unification either

theoretical (e.g., Markov automata) or practical (e.g., PRISM and MODEST modeling languages), but these languages focus on quantitative aspects and do not provide high-level control structures and data handling features (as LNT does, for instance). Work is therefore needed to unify process calculi and quantitative models, still retaining the benefits of both worlds.

- *Increasing the interoperability of analysis tools.* Secondly, we will seek to enhance the interoperability of existing tools for timed, probabilistic, and stochastic systems. Based on scientific exchanges with developers of advanced tools for quantitative analysis, we plan to evolve the CADP toolbox as follows: extending its perimeter of functional verification with quantitative aspects; enabling deeper connections with external analysis components for probabilistic, stochastic, and timed models; and introducing architectural principles for the design and integration of future tools, our long-term goal being the construction of a European collaborative platform encompassing both functional and non-functional analyzes.

### 3.4. Component-Based Architectures for On-the-Fly Verification

On-the-fly verification fights against state explosion by enabling an incremental, demand-driven exploration of LTSs, thus avoiding their entire construction prior to verification. In this approach, LTS models are handled implicitly by means of their *post* function, which computes the transitions going out of given states and thus serves as a basis for any forward exploration algorithm. On-the-fly verification tools are complex software artifacts, which must be designed as modularly as possible to enhance their robustness, reduce their development effort, and facilitate their evolution. To achieve such a modular framework, we undertake research in several directions:

- *New interfaces for on-the-fly LTS manipulation.* The current application programming interface (API) for on-the-fly graph manipulation, named OPEN/CAESAR [35], provides an “opaque” representation of states and actions (transitions labels): states are represented as memory areas of fixed size and actions are character strings. Although appropriate to the pure process algebraic setting, this representation must be generalized to provide additional information supporting an efficient construction of advanced verification features, such as: handling of the types, functions, data values, and parallel structure of the source program under verification, independence of transitions in the LTS, quantitative (timed/probabilistic/stochastic) information, etc.
- *Compositional framework for on-the-fly LTS analysis.* On-the-fly model checkers and equivalence checkers usually perform several operations on graph models (LTSs, Boolean graphs, etc.), such as exploration, parallel composition, partial order reduction, encoding of model checking and equivalence checking in terms of Boolean equation systems, resolution and diagnostic generation for Boolean equation systems, etc. To facilitate the design, implementation, and usage of these functionalities, it is necessary to encapsulate them in software components that could be freely combined and replaced. Such components would act as graph transformers, that would execute (on a sequential machine) in a way similar to coroutines and to the composition of lazy functions in functional programming languages. Besides its obvious benefits in modularity, such a component-based architecture will also make it possible to take advantage of multi-core processors.
- *New generic components for on-the-fly verification.* The quest for new on-the-fly components for LTS analysis must be pursued, with the goal of obtaining a rich catalog of interoperable components serving as building blocks for new analysis features. A long-term goal of this approach is to provide an increasingly large catalog of interoperable components covering all verification and analysis functionalities that appear to be useful in practice. It is worth noticing that some components can be very complex pieces of software (e.g., the encapsulation of an on-the-fly model checker for a rich temporal logic). Ideally, it should be possible to build a novel verification or analysis tool by assembling on-the-fly graph manipulation components taken from the catalog. This would provide a flexible means of building new verification and analysis tools by reusing generic, interoperable model manipulation components.



### **3.5. Real-Life Applications and Case Studies**

We believe that theoretical studies and tool developments must be confronted with significant case studies to assess their applicability and to identify new research directions. Therefore, we seek to apply our languages, models, and tools for specifying and verifying formally real-life applications, often in the context of industrial collaborations.

## HYCOMES Project-Team

### 3. Research Program

#### 3.1. Hybrid Systems Modeling

Systems industries today make extensive use of mathematical modeling tools to design computer controlled physical systems. This class of tools addresses the modeling of physical systems with models that are simpler than usual scientific computing problems by using only Ordinary Differential Equations (ODE) and Difference Equations but not Partial Differential Equations (PDE). This family of tools first emerged in the 1980's with SystemBuild by MatrixX (now distributed by National Instruments) followed soon by Simulink by Mathworks, with an impressive subsequent development.

In the early 90's control scientists from the University of Lund (Sweden) realized that the above approach did not support component based modeling of physical systems with reuse<sup>0</sup>. For instance, it was not easy to draw an electrical or hydraulic circuit by assembling component models of the various devices. The development of the Omola language by Hilding Elmqvist was a first attempt to bridge this gap by supporting some form of Differential Algebraic Equations (DAE) in the models. Modelica quickly emerged from this first attempt and became in the 2000's a major international concerted effort with the Modelica Consortium<sup>0</sup>. A wider set of tools, both industrial and academic, now exists in this segment<sup>0</sup>. In the EDA sector, VHDL-AMS was developed as a standard [13].

Despite these tools are now widely used by a number of engineers, they raise a number of technical difficulties. The meaning of some programs, their mathematical semantics, can be tainted with uncertainty. A main source of difficulty lies in the failure to properly handle the discrete and the continuous parts of systems, and their interaction. How the propagation of mode changes and resets should be handled? How to avoid artifacts due to the use of a global ODE solver causing unwanted coupling between seemingly non interacting subsystems? Also, the mixed use of an equational style for the continuous dynamics with an imperative style for the mode changes and resets is a source of difficulty when handling parallel composition. It is therefore not uncommon that tools return complex warnings for programs with many different suggested hints for fixing them. Yet, these "pathological" programs can still be executed, if wanted so, giving surprising results — See for instance the Simulink examples in [21], [1] and [17].

Indeed this area suffers from the same difficulties that led to the development of the theory of synchronous languages as an effort to fix obscure compilation schemes for discrete time equation based languages in the 1980's. Our vision is that hybrid systems modeling tools deserve similar efforts in theory as synchronous languages did for the programming of embedded systems.

#### 3.2. Background on non-standard analysis

Non-Standard analysis plays a central role in our research on hybrid systems modeling [1], [21], [18], [17]. The following text provides a brief summary of this theory and gives some hints on its usefulness in the context of hybrid systems modeling. This presentation is based on our paper [1], a chapter of Simon Bliudze's PhD thesis [27], and a recent presentation of non-standard analysis, not axiomatic in style, due to the mathematician Lindström [50].

---

<sup>0</sup><http://www.lccc.lth.se/media/LCCC2012/WorkshopSeptember/slides/Astrom.pdf>

<sup>0</sup><https://www.modelica.org/>

<sup>0</sup>SimScape by Mathworks, Amesim by LMS International, now Siemens PLM, and more.

Non-standard numbers allowed us to reconsider the semantics of hybrid systems and propose a radical alternative to the *super-dense time semantics* developed by Edward Lee and his team as part of the Ptolemy II project, where cascades of successive instants can occur in zero time by using  $\mathbb{R}_+ \times \mathbb{N}$  as a time index. In the non-standard semantics, the time index is defined as a set  $\mathbb{T} = \{n\partial \mid n \in \mathbb{N}\}$ , where  $\partial$  is an *infinitesimal* and  $\mathbb{N}$  is the set of *non-standard integers*. Remark that 1/  $\mathbb{T}$  is dense in  $\mathbb{R}_+$ , making it “continuous”, and 2/ every  $t \in \mathbb{T}$  has a predecessor in  $\mathbb{T}$  and a successor in  $\mathbb{T}$ , making it “discrete”. Although it is not effective from a computability point of view, the *non-standard semantics* provides a framework that is familiar to the computer scientist and at the same time efficient as a symbolic abstraction. This makes it an excellent candidate for the development of provably correct compilation schemes and type systems for hybrid systems modeling languages.

Non-standard analysis was proposed by Abraham Robinson in the 1960s to allow the explicit manipulation of “infinitesimals” in analysis [56], [42], [12]. Robinson’s approach is axiomatic; he proposes adding three new axioms to the basic Zermelo-Fraenkel (ZFC) framework. There has been much debate in the mathematical community as to whether it is worth considering non-standard analysis instead of staying with the traditional one. We do not enter this debate. The important thing for us is that non-standard analysis allows the use of the non-standard discretization of continuous dynamics “as if” it was operational.

Not surprisingly, such an idea is quite ancient. Iwasaki et al. [46] first proposed using non-standard analysis to discuss the nature of time in hybrid systems. Bliudze and Krob [28], [27] have also used non-standard analysis as a mathematical support for defining a system theory for hybrid systems. They discuss in detail the notion of “system” and investigate computability issues. The formalization they propose closely follows that of Turing machines, with a memory tape and a control mechanism.

The introduction to non-standard analysis in [27] is very pleasant and we take the liberty to borrow it. This presentation was originally due to Lindstrøm, see [50]. Its interest is that it does not require any fancy axiomatic material but only makes use of the axiom of choice — actually a weaker form of it. The proposed construction bears some resemblance to the construction of  $\mathbb{R}$  as the set of equivalence classes of Cauchy sequences in  $\mathbb{Q}$  modulo the equivalence relation  $(u_n) \approx (v_n)$  iff  $\lim_{n \rightarrow \infty} (u_n - v_n) = 0$ .

### 3.3. Contract-Based Design, Interfaces Theories, and Requirements Engineering

System companies such as automotive and aeronautic companies are facing significant difficulties due to the exponentially raising complexity of their products coupled with increasingly tight demands on functionality, correctness, and time-to-market. The cost of being late to market or of imperfections in the products is staggering as witnessed by the recent recalls and delivery delays that many major car and airplane manufacturers had to bear in the recent years. The specific root causes of these design problems are complex and relate to a number of issues ranging from design processes and relationships with different departments of the same company and with suppliers, to incomplete requirement specification and testing.

We believe the most promising means to address the challenges in systems engineering is to employ structured and formal design methodologies that seamlessly and coherently combine the various viewpoints of the design space (behavior, space, time, energy, reliability, ...), that provide the appropriate abstractions to manage the inherent complexity, and that can provide correct-by-construction implementations. The following technology issues must be addressed when developing new approaches to the design of complex systems:

- The overall design flows for heterogeneous systems and the associated use of models across traditional boundaries are not well developed and understood. Relationships between different teams inside a same company, or between different stake-holders in the supplier chain, are not well supported by solid technical descriptions for the mutual obligations.
- System requirements capture and analysis is in large part a heuristic process, where the informal text and natural language-based techniques in use today are facing significant challenges. Formal requirements engineering is in its infancy: mathematical models, formal analysis techniques and links to system implementation must be developed.

- Dealing with variability, uncertainty, and life-cycle issues, such as extensibility of a product family, are not well-addressed using available systems engineering methodologies and tools.

The challenge is to address the entire process and not to consider only local solutions of methodology, tools, and models that ease part of the design.

*Contract-based design* has been proposed as a new approach to the system design problem that is rigorous and effective in dealing with the problems and challenges described before, and that, at the same time, does not require a radical change in the way industrial designers carry out their task as it cuts across design flows of different type. Indeed, contracts can be used almost everywhere and at nearly all stages of system design, from early requirements capture, to embedded computing infrastructure and detailed design involving circuits and other hardware. Contracts explicitly handle pairs of properties, respectively representing the assumptions on the environment and the guarantees of the system under these assumptions. Intuitively, a contract is a pair  $C = (A, G)$  of assumptions and guarantees characterizing in a formal way 1) under which context the design is assumed to operate, and 2) what its obligations are. Assume/Guarantee reasoning has been known for a long time, and has been used mostly as verification mean for the design of software [54]. However, contract based design with explicit assumptions is a philosophy that should be followed all along the design, with all kinds of models, whenever necessary. Here, specifications are not limited to profiles, types, or taxonomy of data, but also describe the functions, performances of various kinds (time and energy), and reliability. This amounts to enrich a component's interface with, on one hand, formal specifications of the behavior of the environment in which the component may be instantiated and, on the other hand, of the expected behavior of the component itself. The consideration of rich interfaces is still in its infancy. So far, academic researchers have addressed the mathematics and algorithmics of interfaces theories and contract-based reasoning. To make them a technique of choice for system engineers, we must develop:

- Mathematical foundations for interfaces and requirements engineering that enable the design of frameworks and tools;
- A system engineering framework and associated methodologies and tool sets that focus on system requirements modeling, contract specification, and verification at multiple abstraction layers.

A detailed bibliography on contract and interface theories for embedded system design can be found in [2]. In a nutshell, contract and interface theories fall into two main categories:

*Assume/guarantee contracts.* By explicitly relying on the notions of assumptions and guarantees, A/G-contracts are intuitive, which makes them appealing for the engineer. In A/G-contracts, assumptions and guarantees are just properties regarding the behavior of a component and of its environment. The typical case is when these properties are formal languages or sets of traces, which includes the class of safety properties [47], [36], [53], [15], [37]. Contract theories were initially developed as specification formalisms able to refuse some inputs from the environment [43]. A/G-contracts were advocated by the SPEEDS project [20]. They were further experimented in the framework of the CESAR project [38], with the additional consideration of *weak* and *strong* assumptions. This is still a very active research topic, with several recent contributions dealing with the timed [25] and probabilistic [32], [33] viewpoints in system design, and even mixed-analog circuit design [55].

*Automata theoretic interfaces.* Interfaces combine assumptions and guarantees in a single, automata theoretic specification. Most interface theories are based on Lynch Input/Output Automata [52], [51]. Interface Automata [59], [58], [60], [34] focus primarily on parallel composition and compatibility: Two interfaces can be composed and are compatible if there is at least one environment where they can work together. The idea is that the resulting composition exposes as an interface the needed information to ensure that incompatible pairs of states cannot be reached. This can be achieved by using the possibility, for an Interface Automaton, to refuse selected inputs from the environment in a given state, which amounts to the implicit assumption that the environment will never produce any of the refused inputs, when the interface is in this state. Modal Interfaces [3] inherit from both Interface Automata and the originally unrelated notion of Modal Transition System [49], [14], [29], [48]. Modal Interfaces are strictly more expressive than Interface Automata by decoupling the I/O orientation of an event and its deontic modalities (mandatory, allowed or forbidden). Informally, a

*must* transition is available in every component that realizes the modal interface, while a *may* transition needs not be. Research on interface theories is still very active. For instance, timed [61], [22], [24], [40], [39], [23], probabilistic [32], [41] and energy-aware [35] interface theories have been proposed recently.

Requirements Engineering is one of the major concerns in large systems industries today, particularly so in sectors where certification prevails [57]. DOORS projects collecting requirements are poorly structured and cannot be considered a formal modeling framework today. They are nothing more than an informal documentation enriched with hyperlinks. As examples, medium size sub-systems may have a few thousands requirements and the Rafale fighter aircraft has above 250,000 of them. For the Boeing 787, requirements were not stable while subcontractors performed the development of the fly-by-wire and of the landing gear subsystems.

We see Contract-Based Design and Interfaces Theories as innovative tools in support of Requirements Engineering. The Software Engineering community has extensively covered several aspects of Requirements Engineering, in particular:

- the development and use of large and rich *ontologies*; and
- the use of Model Driven Engineering technology for the structural aspects of requirements and resulting hyperlinks (to tests, documentation, PLM, architecture, and so on).

Behavioral models and properties, however, are not properly encompassed by the above approaches. This is the cause of a remaining gap between this phase of systems design and later phases where formal model based methods involving behavior have become prevalent—see the success of Matlab/Simulink/Scade technologies. We believe that our work on contract based design and interface theories is best suited to bridge this gap.

## MUTANT Project-Team

### 3. Research Program

#### 3.1. Machine Listening

**Participants:** Arshia Cont, Philippe Cuvillier, Florent Jacquemard, Maxime Sirbu, Adrien Ycart.

When human listeners are confronted with musical sounds, they rapidly and automatically find their way in the music. Even musically untrained listeners have an exceptional ability to make rapid judgments about music from short examples, such as determining music style, performer, beating, and specific events such as instruments or pitches. Making computer systems capable of similar capabilities requires advances in both music cognition, and analysis and retrieval systems employing signal processing and machine learning.

Machine listening in our context refers to the capacity of our computers to understand “non-speech sound” by analyzing the content of music and audio signals and combining advanced signal processing and machine learning. The major focus of MuTant has been on Real-time Machine listening algorithms spanning *Real-time Recognition Systems* (such as event detection) and also *Information Retrieval* (such as structure discovery and qualitative parameter estimation). Our major achievement lies in our unique Real-time Score Following (aka Audio-to-Score Alignment) system that are featured in the Antescofo system (cf. Section 5.1). We also contributed to the field of On-line Music Structure Discovery in Audio Processing, and lately to the problem of off-line rhythmic quantization on Symbolic Data.

##### 3.1.1. Real-time Audio-to-Score Alignment.

This is a continuation of prior work of team-founder [1] which proved the utility of strongly-timed probabilistic models in form of Semi-Markov Hidden States. Our most important theoretical contribution is reported in [37], [38] that introduced Time-coherency criteria for probabilistic models and led to general robustness of the Antescofo listening machine, and allowed its deployment for all music instruments and all setups around the world. We further studied the integration of other recognition algorithms in the algorithm in form of *Information Fusion* and for singing voice based on Lyric data in [49]. Collaboration with our Japanese counterparts led to extensions of our model to the symbolic domain reported in [56]. Collaboration with the SIERRA team created a joint research momentum for fostering such applications to weakly-supervised discriminative models reported in [54]. Our Real-time Audio-to-Score alignment is a major component of the Antescofo software described in Section 5.1.

##### 3.1.2. Online Methods for Audio Segmentation and Clustering.

To extend our listening approach to general sound, we envisioned dropping the prior information provided by music scores and replacing it by the inherent structure in general audio signals. Early attempts by the team leader employed [2] Methods of Information Geometry, an attempt to join Information Theory, Differential Geometry and Signal Processing. We were among the first teams in the world advocating the use of such approaches for audio signal processing and we participated in the growth of the community. A major breakthrough of this approach is reported in [39] and the PhD Thesis [40] that outline a general real-time change detection mechanism. Automatic structure discovery was further pursued in a MS thesis project in 2013 [55]. By that time we realized that Information Manifolds do not necessarily provide the invariance needed for automatic structure discovery of audio signals, especially for natural sounds. Following this report, we pursued an alternative approach in 2014 and in collaboration with the Inria SIERRA Team [30]. The result of this joint work was published in IEEE ICASSP 2015 and won the best student paper award [29]. We are currently studying massive applications of this approach to natural sounds and in robotics applications in the framework of Maxime Sirbu’s PhD project.

### 3.1.3. Symbolic Music Information Retrieval and Rhythm Transcription.

Rhythmic data are commonly represented by tree structures (rhythms trees) due to the theoretical proximity of such structures with the proportional representation of time values in traditional musical notation. We are studying the application to rhythm notation of techniques and tools for symbolic processing of tree structures, in particular tree automata and term rewriting.

Our main contribution in that context is the development of a new framework for rhythm transcription [23], [22], [65], [31] addressing the problem of converting a sequence of timestamped notes, *e.g.* a file in MIDI format, into a score in traditional music notation. This problem is crucial in the context assisted music composition environments and music score editors. It arises immediately as insoluble unequivocally: in order to fit the musical context, the system has to balance constraints of precision and readability of the generated scores. Our approach is based on algorithms for the exploration and lazy enumeration of large sets of weighted trees (tree series), representing possible solutions to a problem of transcription. A side problem concerns the equivalent notations of the same rhythm, for which we have developed a term rewrite approach, based on a new equational theory of rhythm notation [42], [51], [52].

## 3.2. Synchronous and realtime programming for computer music

**Participants:** Julia Blondeau, Arshia Cont, Jean-Louis Giavitto.

The research presented here aims at the development of a programming model dedicated to authoring of time and interaction for the next generation of interactive music systems. Study, formalization and implementation of such programming paradigm, strongly coupled to recognition systems discussed in the previous section, constitutes the second objective of the MuTant project.

The tangible result of this research is the development of the Antescofo system (cf. Section 5.1) for the design and implementation of musical scenarios in which the human and computer actions are in constant real-time interaction. Through such development, Antescofo has already made itself into the community; it serves as the backbone of temporal organization of more than 100 performances since 2012 and used both for preexisting pieces and new creations by music ensembles such as Berliner Philharmoniker, Los Angeles Philharmonic, Ensemble Intercontemporain or Orchestre de Paris to name a few.

Compared to programmable sequencers or interactive music systems (like Max or PureData) the Antescofo DSL offers a rich notion of time reference and provides explicit time frame for the environment with a comprehensive list of musical synchronization strategies and proposes and predictable mechanisms for controlling time at various timescales (temporal determinism) and across concurrent code modules (time-mediated concurrency).

### 3.2.1. Multiple Times.

Audio and music often involve the presence and cooperation of multiple notions of *time*: an ideal time authored by the composer in a score and also a performance time produced jointly by the performers and the real-time electronics; where instant and duration are expressed both in physical time (milliseconds), in relative time (relative to an unknown dynamic tempo) or through logical events and relations (“at the peak of intensity”, “at the end of the musical phrase”, “twice faster”).

Antescofo is the first languages that addresses this variety of temporal notions, relying on the synchronous approach for the handling of atomic and logical events and an anticipative notion of tempo for the handling of relative duration [35], [45]. A first partial model of time at work in Antescofo (single time, static activities) has been formalized relying on parametric timed automata [43] and constitutes the reference semantics for tests (cf. section 3.3). A denotational semantics of the complete language (multiple times and dynamic constructions including anticipative synchronization strategies) has been published in [44].

### 3.2.2. Human-Computer Synchronizations.

Antescofo introduces the notion of *temporal scope* to formalize relationships between temporal information specified in the score and their realization during a performance [36]. A temporal scope is attached to a sequence of actions, can be inherited or dynamically changed as a result of a computation. A synchronization strategy is part of a temporal scope definition. They use the performer's position information and its tempo estimation from the listening module, to drive the passing of time in a sequence of atomic and durative actions.

Synchronization strategies have been systematically studied to evaluate their musical relevance in collaboration with Orchestre de Paris and composer Marco Stroppa. Anticipative strategies enable handling of uncertainties inherent in musical event occurrence, exhibiting a smooth musical rendering whilst preserving articulation points and target events [63].

### 3.2.3. Temporal Organization.

Several constructions dedicated to the expression of the temporal organization of musical entities and their control have enriched the language from the start of the project. These constructions have been motivated by composer's research residences in our team: representation of open scores (J. Freeman); anticipative synchronization strategies (C. Trapani); adaptive sampling of continuous curve in relative time for the dynamic control of sound synthesis (J.-M. Fernandez); musical gesture (J. Blondeau); first class processes, actors and continuation combinators for the development of libraries of reusable parametric temporal behaviors (M. Stroppa, Y. Maresz); *etc.*

The reaction to a logical event is a unique feature in the computer music system community [57]. It extends the well known *when operator* in synchronous languages with process creation. Elaborating on this low-level mechanism, *temporal patterns* [48] enable expression of complex temporal constraints mixing instant and duration. The problem of online matching where the events are presented in real time and the matching is computed incrementally as well, has received a recent attention from the model-checking community, but with less constrained causal constraints.

### 3.2.4. Visualization and Monitoring of Event-driven and Time-driven Computations.

The authoring of complex temporal organization can be greatly improved through adapted visual interfaces, and has led to the development of *AscoGraph*, a dedicated user interface to Antescofo. *AscoGraph* is used both for edition and monitoring interface of the system during performances [34]. This project was held from end 2012 to end 2014 thanks to Inria ADT and ANR support.

An information visualisation perspective has been taken for the design of timeline-based representation of action items, looking for information coherence and clarity, facility of seeking and navigation, hierarchical distinction and explicit linking [33] while minimizing the information overload for the presentation of the nested structure of complex concurrent activities [32].

## 3.3. Semantics, Verification and Test of Mixed Scores

**Participants:** Jean-Louis Giavitto, Florent Jacquemard, Clément Poncelet.

We address the questions of *functional reliability* and *temporal predictability* in score-based interactive music systems such as Antescofo. On the one hand, checking these properties is difficult for these systems involving an amount of human interactions as well as timing constraints (for audio computations) beyond those of many other real-time applications such as embedded control. On the other hand, although they are expected to behave properly during public concerts, these systems are not safety critical, and therefore a complete formal certification is not strictly necessary in our case.

Our objective in this context is to provide techniques and tools to assist both programmers of scores (*i.e.* composers) and the developers of the system itself. [47], [46]. It should be outlined that the former are generally not experts in real-time programming, and we aim at giving them a clear view of what will be the outcome of the score that they are writing, and what are the limits of what is playable by the system. To help the development of Antescofo, we have built a framework for automated timed conformance testing. [14], [18], [58], [60], [59].



In both cases, it is important to be able to predict statically the behavior of the system in response to every possible musician input. This cannot be done manually and requires first a formal definition of the semantics of scores, and second using advanced symbolic state exploration techniques (model checking) [43].

## PARKAS Project-Team

### 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,<sup>0</sup> 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,<sup>0</sup> with MODELICA<sup>0</sup> 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.

<sup>0</sup><http://www.esterel-technologies.com/products/scade-suite>

<sup>0</sup><http://www.mathworks.com/products/simulink>

<sup>0</sup><https://www.modelica.org>

## 3.2. Efficient Compilation for Parallel and Distributed Computing

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 executed code and source programs. We develop techniques to formally prove the correctness of compilation passes for synchronous languages (Lustre), and to validate compilation optimization for C code in the presence of threads.

### 3.3.1. Lustre:

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.

### **3.3.2. C/C++:**

The recently approved C11 and C++11 standards define a concurrency model for the C and C++ languages, which were originally designed without concurrency support. Their intent is to permit most compiler and hardware optimizations, while providing escape mechanisms for writing portable, high-performance, low-level code. Mainstream compilers are being modified to support the new standards. A subtle class of compiler bugs is the so-called concurrency compiler bugs, where compilers generate correct sequential code but break the concurrency memory model of the programming language. Such bugs are observable only when the miscompiled functions interact with concurrent contexts, making them particularly hard to detect. All previous techniques to test compiler correctness miss concurrency compiler bugs.

## POSET Team

### 3. Research Program

#### 3.1. Research Program

Our research programs is structured into three complementary research axis : models, languages and systems, allowing us to develop our multi-disciplinary approach while validating each progress in the related specific fields of computer science ranging among computer music, multi-modal system design, reactive and real-time programing, typed functional programming, formal languages, graph representation theory, applied algebra, logic in computer science, etc.

##### 3.1.1. Models

Inverse semigroup theory has recently been shown [13], [7], [12] [20] to unify most string-based, tree-based or even graph-based modeling approaches. It thus provides a consistent and robust mathematical framework to model the sequential, parallel and reactive aspects of temporal media. Developing the mathematical foundations of our proposal amounts to:

- studying the combinatorial and algorithmic properties of the emerging algebra-based model of structured temporal media,
- developing formal techniques and tools for expressing and verifying properties of temporal media programs especially with a view towards capturing temporal media programing by constraint satisfaction approaches,
- deriving from the known generators of these models adequate sets of application-oriented modeling functions.

##### 3.1.2. Languages

Functional programming is the key link between well-defined mathematical structures and their computerized realizations. Based on functional programming frameworks such as Haskell<sup>0</sup>, we are prototyping a Domain Specific Language (DSL) [10] [15] dedicated to the programming of interactive temporal media programming. In this research axis, we aim more specifically at

- designing a robust and modular software architecture that allows to reuse existing pieces of software as well as simply combining them together with new ones,
- defining and implementing a DSL for programming interactive multimedia systems via a simple algebra-based high-level and multi-scale control and combination layer,
- finding the right balance between generic views of temporal media when seen as abstract temporal frames and their specializations when representing concrete gestures, sound, audio, videos, animations, etc.

##### 3.1.3. Systems

Multi-modal interactive systems gather various techniques to capture and analyze gestures, and to combine, transform and produce temporal media. Through regular experiments in collaboration with artists, we also aim at assessing, refining and extending the applicability of our proposal by:

- developing a robust and mathematically well-founded representation of systems and of their behaviors, both programmatic and visual,
- developing and evaluating the adequacy of the GUI induced by this representation when used by artists,
- relating the new models with more classical models of music formalisms and, beyond, other temporal media such as animations, videos, etc.

<sup>0</sup>See [31] for an historical presentation of the Haskell programming language.

## SPADES Project-Team

### 3. Research Program

#### 3.1. Introduction

The SPADES research program is organized around three main themes, *Components and contracts*, *Real-time multicore programming*, and *Language-based fault tolerance*, that seek to answer the three key questions identified in Section 2.1. We plan to do so by developing and/or building on programming languages and techniques based on formal methods and formal semantics (hence the use of “*sound programming*” in the project-team title). In particular, we seek to support design where correctness is obtained by construction, relying on proven tools and verified constructs, with programming languages and programming abstractions designed with verification in mind.

#### 3.2. Components and Contracts

Component-based construction has long been advocated as a key approach to the “correct-by-construction” design of complex embedded systems [65]. Witness component-based toolsets such as UC Berkeley’s PTOLEMY [53], Verimag’s BIP [36], or the modular architecture frameworks used, for instance, in the automotive industry (AUTOSAR) [28]. For building large, complex systems, a key feature of component-based construction is the ability to associate with components a set of *contracts*, which can be understood as rich behavioral types that can be composed and verified to guarantee a component assemblage will meet desired properties. The goal in this theme is to study the formal foundations of the component-based construction of embedded systems, to develop component and contract theories dealing with real-time, reliability and fault-tolerance aspects of components, and to develop proof-assistant-based tools for the computer-aided design and verification of component-based systems.

Formal models for component-based design are an active area of research (see *e.g.*, [29], [30]). However, we are still missing a comprehensive formal model and its associated behavioral theory able to deal *at the same time* with different forms of composition, dynamic component structures, and quantitative constraints (such as timing, fault-tolerance, or energy consumption). Notions of contracts and interface theories have been proposed to support modular and compositional design of correct-by-construction embedded systems (see *e.g.*, [40], [41] and the references therein), but having a comprehensive theory of contracts that deals with all the above aspects is still an open question [71]. In particular, it is not clear how to accommodate different forms of composition, reliability and fault-tolerance aspects, or to deal with evolving component structures in a theory of contracts.

Dealing in the same component theory with heterogeneous forms of composition, different quantitative aspects, and dynamic configurations, requires to consider together the three elements that comprise a component model: behavior, structure and types. *Behavior* refers to behavioral (interaction and execution) models that characterize the behavior of components and component assemblages (*e.g.*, transition systems and their multiple variants – timed, stochastic, etc.). *Structure* refers to the organization of component assemblages or configurations, and the composition operators they involve. *Types* refer to properties or contracts that can be attached to components and component interfaces to facilitate separate development and ensure the correctness of component configurations with respect to certain properties. Taking into account dynamicity requires to establish an explicit link between behavior and structure, as well as to consider higher-order systems, both of which have a direct impact on types.

We plan to develop our component theory by progressing on two fronts: component calculi, and semantical framework. The work on typed component calculi aims to elicit process calculi that capture the main insights of component-based design and programming and that can serve as a bridge towards actual architecture description and programming language developments. The work on the semantical framework should, in the longer term, provide abstract mathematical models for the more operational and linguistic analysis afforded by component calculi. Our work on component theory will find its application in the development of a COQ-based toolchain for the certified design and construction of dependable embedded systems, which constitutes our third main objective for this axis.

### 3.3. Real-Time Multicore Programming

Programming real-time systems (*i.e.*, systems whose correct behavior depends on meeting timing constraints) requires appropriate languages (as exemplified by the family of synchronous languages [39]), but also the support of efficient scheduling policies, execution time and schedulability analyses to guarantee real-time constraints (*e.g.*, deadlines) while making the most effective use of available (processing, memory, or networking) resources. Schedulability analysis involves analyzing the worst-case behavior of real-time tasks under a given scheduling algorithm and is crucial to guarantee that time constraints are met in any possible execution of the system. Reactive programming and real-time scheduling and schedulability for multiprocessor systems are old subjects, but they are nowhere as mature as their uniprocessor counterparts, and still feature a number of open research questions [35], [48], in particular in relation with mixed criticality systems. The main goal in this theme is to address several of these open questions.

We intend to focus on two issues: multicriteria scheduling on multiprocessors, and schedulability analysis for real-time multiprocessor systems. Beyond real-time aspects, multiprocessor environments, and multicore ones in particular, are subject to several constraints *in conjunction*, typically involving real-time, reliability and energy-efficiency constraints, making the scheduling problem more complex for both the offline and the online cases. Schedulability analysis for multiprocessor systems, in particular for systems with mixed criticality tasks, is still very much an open research area.

Distributed reactive programming is rightly singled out as a major open issue in the recent, but heavily biased (it essentially ignores recent research in synchronous and dataflow programming), survey by Bainomugisha et al. [35]. For our part, we intend to focus on two questions: devising synchronous programming languages for distributed systems and precision-timed architectures, and devising dataflow languages for multiprocessors supporting dynamicity and parametricity while enjoying effective analyses for meeting real-time, resource and energy constraints in conjunction.

### 3.4. Language-Based Fault Tolerance

Tolerating faults is a clear and present necessity in networked embedded systems. At the hardware level, modern multicore architectures are manufactured using inherently unreliable technologies [43], [58]. The evolution of embedded systems towards increasingly distributed architectures highlighted in the introductory section means that dealing with partial failures, as in Web-based distributed systems, becomes an important issue. While fault-tolerance is an old and much researched topic, several important questions remain open: automation of fault-tolerance provision, composable abstractions for fault-tolerance, fault diagnosis, and fault isolation.

The first question is related to the old question of “system structure for fault-tolerance” as originally discussed by Randell for software fault tolerance [77], and concerns in part our ability to clearly separate fault-tolerance aspects from the design and programming of purely “functional” aspects of an application. The classical arguments in favor of a clear separation of fault-tolerance concerns from application code revolve around reduced code and maintenance complexity [49]. The second question concerns the definition of appropriate abstractions for the modular construction of fault-tolerant embedded systems. The current set of techniques available for building such systems spans a wide range, including exception handling facilities, transaction management schemes, rollback/recovery schemes, and replication protocols. Unfortunately, these different

techniques do not necessarily compose well – for instance, combining exception handling and transactions is non trivial, witness the flurry of recent work on the topic, see *e.g.*, [64] and the references therein –, they have no common semantical basis, and they suffer from limited programming language support. The third question concerns the identification of causes for faulty behavior in component-based assemblages. It is directly related to the much researched area of fault diagnosis, fault detection and isolation [66].

We intend to address these questions by leveraging programming language techniques (programming constructs, formal semantics, static analyses, program transformations) with the goal to achieve provable fault-tolerance, *i.e.*, the construction of systems whose fault-tolerance can be formally ensured using verification tools and proof assistants. We aim in this axis to address some of the issues raised by the above open questions by using aspect-oriented programming techniques and program transformations to automate the inclusion of fault-tolerance in systems (software as well as hardware), by exploiting reversible programming models to investigate composable recovery abstractions, and by leveraging causality analyses to study fault-ascription in component-based systems. Compared to the huge literature on fault-tolerance in general, in particular in the systems area (see *e.g.*, [59] for an interesting but not so recent survey), we find by comparison much less work exploiting formal language techniques and tools to achieve or support fault-tolerance. The works reported in [42], [44], [47], [54], [67], [76], [81] provide a representative sample of recent such works.

A common theme in this axis is the use and exploitation of causality information. Causality, *i.e.*, the logical dependence of an effect on a cause, has long been studied in disciplines such as philosophy [72], natural sciences, law [73], and statistics [74], but it has only recently emerged as an important focus of research in computer science. The analysis of logical causality has applications in many areas of computer science. For instance, tracking and analyzing logical causality between events in the execution of a concurrent system is required to ensure reversibility [70], to allow the diagnosis of faults in a complex concurrent system [61], or to enforce accountability [69], that is, designing systems in such a way that it can be determined without ambiguity whether a required safety or security property has been violated, and why. More generally, the goal of fault-tolerance can be understood as being to prevent certain causal chains from occurring by designing systems such that each causal chain either has its premises outside of the fault model (*e.g.*, by introducing redundancy [59]), or is broken (*e.g.*, by limiting fault propagation [78]).



## TEA Project-Team

### 3. Research Program

#### 3.1. Previous Works

The challenges of team TEA support the claim that sound Cyber-Physical System design (including embedded, reactive, and concurrent systems altogether) should consider multi-form time models as a central aspect. In this aim, architectural specifications found in software engineering are a natural focal point to start from. Architecture descriptions organize a system model into manageable components, establish clear interfaces between them, collect domain-specific constraints and properties to help correct integration of components during system design. The definition of a formal design methodology to support heterogeneous or multi-form models of time in architecture descriptions demands the elaboration of sound mathematical foundations and the development of formal calculi and methods to instrument them. This constitutes the research program of team TEA.

System design based on the “synchronous paradigm” has focused the attention of many academic and industrial actors on abstracting non-functional implementation details from system design. This elegant design abstraction focuses on the logic of interaction in reactive programs rather than their timed behavior, allowing to secure functional correctness while remaining an intuitive programming model for embedded systems. Yet, it corresponds to embedded technologies of single cores and synchronous buses from the 90s, and may hardly cover the semantic diversity of distribution, parallelism, heterogeneity, of cyber-physical systems found in 21st century Internet-connected, true-time<sup>TM</sup>-synchronized clouds, of tomorrow’s grids.

By contrast with a synchronous hypothesis yet from the same era, the polychronous MoCC implemented in the data-flow specification language Signal, available in the Eclipse project POP<sup>0</sup> and in the CCSL standard,<sup>0</sup> are inherently capable of describing multi-clock abstractions of GALS systems. The POP and TimeSquare projects provide toolled infrastructures to refine high-level specifications into real-time streaming applications or locally synchronous and globally asynchronous systems, through a series of model analysis, verification, and synthesis services. These tool-supported refinement and transformation techniques can assist the system engineer from the earliest design stages of requirement specification to the latest stages of synthesis, scheduling and deployment. These characteristics make polychrony much closer to the required semantic for compositional, refinement-based, architecture-driven, system design.

While polychrony was a step ahead of the traditional synchronous hypothesis, CCSL is a leap forward from synchrony and polychrony. The essence of CCSL is “multi-form time” toward addressing all of the domain-specific physical, electronic and logical aspects of cyber-physical system design.

#### 3.2. Modeling Times

To make a sense and eventually formalize the semantics of time in system design, we should most certainly rely on algebraic representations of time found in previous works and introduce the paradigm of “time systems” (type systems to represent time) in a way reminiscent to CCSL. Just as a type system abstracts data carried along operations in a program, a time system abstracts the causal interaction of that program module or hardware element with its environment, its pre and post conditions, its assumptions and guarantees, either logical or numerical, discrete or continuous. Some fundamental concepts of the time systems we envision are present in the clock calculi found in data-flow synchronous languages like Signal or Lustre, yet bound to a particular model of concurrency, hence time.

<sup>0</sup> Polychrony on Polarsys, <https://www.polarsys.org/projects/polarsys.pop>

<sup>0</sup> Clock Constraints in UML/MARTE CCSL. C. André, F. Mallet. RR-6540. Inria, 2008. <http://hal.inria.fr/inria-00280941>

In particular, the principle of refinement type systems<sup>0</sup>, is to associate information (data-types) inferred from programs and models with properties pertaining, for instance, to the algebraic domain on their value, or any algebraic property related to its computation: effect, memory usage, pre-post condition, value-range, cost, speed, time, temporal logic<sup>0</sup>. Being grounded on type and domain theories, a time system should naturally be equipped with program analysis techniques based on type inference (for data-type inference) or abstract interpretation (for program properties inference) to help establish formal relations between heterogeneous component “types”. Just as a time calculus may formally abstract timed concurrent behaviors of system components, timed relations (abstraction and refinement) represent interaction among components.

Scalability and compositionality requires the use of assume-guarantee reasoning to represent them, and to facilitate composition by behavioral sub-typing, in the spirit of the (static) contract-based formalism proposed by Passerone et al.<sup>0</sup>. Verification problems encompassing heterogeneously timed specifications are common and of great variety: checking correctness between abstract and concrete time models relates to desynchronisation (from synchrony to asynchrony) and scheduling analysis (from synchrony to hardware). More generally, they can be perceived from heterogeneous timing viewpoints (e.g. mapping a synchronous-time software on a real-time middle-ware or hardware).

This perspective demands capabilities not only to inject time models one into the other (by abstract interpretation, using refinement calculi), to compare time abstractions one another (using simulation, refinement, bi-simulation, equivalence relations) but also to prove more specific properties (synchronization, determinism, endochrony). All this formalization effort will allow to effectively perform the tooled validation of common cross-domain properties (e.g. cost v.s. power v.s. performance v.s. software mapping) and tackle equally common yet though case studies such as these linking battery capacity, to on-board CPU performance, to static software schedulability, to logical software correctness and plant controllability: the choice of the right sampling period across the system components.

### 3.3. Modeling Architectures

To address the formalization of such cross-domain case studies, modeling the architecture formally plays an essential role. An architectural model represents components in a distributed system as boxes with well-defined interfaces, connections between ports on component interfaces, and specifies component properties that can be used in analytical reasoning about the model. Several architectural modeling languages for embedded systems have emerged in recent years, including the SAE AADL<sup>0</sup>, SysML<sup>0</sup>, UML MARTE<sup>0</sup>.

In system design, an architectural specification serves several important purposes. First, it breaks down a system model into manageable components to establish clear interfaces between components. In this way, complexity becomes manageable by hiding details that are not relevant at a given level of abstraction. Clear, formally defined, component interfaces allow us to avoid integration problems at the implementation phase. Connections between components, which specify how components affect each other, help propagate the effects of a change in one component to the linked components.

Most importantly, an architectural model is a repository to share knowledge about the system being designed. This knowledge can be represented as requirements, design artifacts, component implementations, held together by a structural backbone. Such a repository enables automatic generation of analytical models for different aspects of the system, such as timing, reliability, security, performance, energy, etc. Since all the models are generated from the same source, the consistency of assumptions w.r.t. guarantees, of abstractions w.r.t. refinements, used for different analyses becomes easier, and can be properly ensured in a design methodology based on formal verification and synthesis methods.

<sup>0</sup> *Abstract Refinement Types*. N. Vazou, P. Rondon, and R. Jhala. European Symposium on Programming. Springer, 2013.

<sup>0</sup> *LTL types FRP*. A. Jeffrey. Programming Languages meets Program Verification.

<sup>0</sup> *A contract-based formalism for the specification of heterogeneous systems*. L. Benvenistu, et al. FDL, 2008

<sup>0</sup> *Architecture Analysis and Design Language*, AS-5506. SAE, 2004. <http://standards.sae.org/as5506b>

<sup>0</sup> *System modeling Language*. OMG, 2007. <http://www.omg.org/spec/SysML>

<sup>0</sup> *UML Profile for MARTE*. OMG, 2009. <http://www.omg.org/spec/MARTE>

Related works in this aim, and closer in spirit to our approach (to focus on modeling time) are domain-specific languages such as Prelude<sup>0</sup> to model the real-time characteristics of embedded software architectures. Conversely, standard architecture description languages could be based on algebraic modeling tools, such as interface theories with the ECDAR tool<sup>0</sup>.

In project TEA, it takes form by the normalization of the AADL standard's formal semantics and the proposal of a time specification annex in the form of related standards, such as CCSL, to model concurrency time and physical properties, and PSL, to model timed traces.

### 3.4. Scheduling Theory

Based on sound formalization of time and CPS architectures, real-time scheduling theory provides tools for predicting the timing behavior of a CPS which consists of many interacting software and hardware components. Expressing parallelism among software components is a crucial aspect of the design process of a CPS. It allows for efficient partition and exploitation of available resources.

The literature about real-time scheduling<sup>0</sup> provides very mature schedulability tests regarding many scheduling strategies, preemptive or non-preemptive scheduling, uniprocessor or multiprocessor scheduling, etc. Scheduling of data-flow graphs has also been extensively studied in the past decades.

A milestone in this prospect is the development of abstract affine scheduling techniques<sup>0</sup>. It consists, first, of approximating task communication patterns (here Safety-Critical Java threads) using cyclo-static data-flow graphs and affine functions. Then, it uses state of the art ILP techniques to find optimal schedules and concretize them as real-time schedules for Safety Critical Java programs<sup>00</sup>.

Abstract scheduling, or the use of abstraction and refinement techniques in scheduling borrowed to the theory of abstract interpretation<sup>0</sup> is a promising development toward tooling methodologies to orchestrate thousands of heterogeneous hardware/software blocks on modern CPS architectures (just consider modern cars or aircrafts). It is an issue that simply defies the state of the art and known bounds of complexity theory in the field, and consequently requires a particular address.

To develop the underlying theory of this promising research topic, we first need to deepen the theoretical foundation to establish links between scheduling analysis and abstract interpretation. A theory of time systems would offer the ideal framework to pursue this development. It amounts to representing scheduling constraints, inferred from programs, as types or contract properties. It allows to formalize the target time model of the scheduler (the architecture, its middle-ware, its real-time system) and defines the basic concepts to verify assumptions made in one with promises offered by the other: contract verification or, in this case, synthesis.

### 3.5. Virtual Prototyping

Virtual Prototyping is the technology of developing realistic simulators from models of a system under design; that is, an emulated device that captures most, if not all, of the required properties of the real system, based on its specifications. A virtual prototype should be run and tested like the real device. Ideally, the real application software would be run on the virtual prototyping platform and produce the same results as the real device with the same sequence of outputs and reported performance measurements. This may be true to some extent only. Some trade-offs have often to be made between the accuracy of the virtual prototype, and time to develop accurate models.

<sup>0</sup>The Prelude language. LIFL and ONERA, 2012. <http://www.lifl.fr/~forget/prelude.html>

<sup>0</sup>PyECDAR, timed games for timed specifications. Inria, 2013. <https://project.inria.fr/pyecdar>

<sup>0</sup>A survey of hard real-time scheduling for multiprocessor systems. R. I. Davis and A. Burns. *ACM Computing Survey* 43(4), 2011.

<sup>0</sup>Buffer minimization in EDF scheduling of data-flow graphs. A. Bouakaz and J.-P. Talpin. *LCTES*, ACM, 2013.

<sup>0</sup>ADFG for the synthesis of hard real-time applications. A. Bouakaz, J.-P. Talpin, J. Vitek. *ACSD*, IEEE, June 2012.

<sup>0</sup>Design of SCJ Level 1 Applications Using Affine Abstract Clocks. A. Bouakaz and J.-P. Talpin. *SCOPES*, ACM, 2013.

<sup>0</sup>La vérification de programmes par interprétation abstraite. P. Cousot. Séminaire au Collège de France, 2008.

In order to speed-up simulation time, the virtual prototype must trade-off with something. Depending upon the application designer's goals, one may be interested in trading some loss of accuracy in exchange for simulation speed, which leads to constructing simulation models that focus on some design aspects and provide abstraction of others. A simulation model can provide an abstraction of the simulated hardware in three directions:

- *Computation abstraction.* A hardware component computes a high level function by carrying out a series of small steps executed by composing logical gates. In a virtual prototyping environment, it is often possible to compute the high level function directly by using the available computing resources on the simulation host machine, thus abstracting the hardware function.
- *Communication abstraction.* Hardware components communicate together using some wiring, and some protocol to transmit the data. Simulation of the communication and the particular protocol may be irrelevant for the purpose of virtual prototyping: communication can be abstracted into higher level data transmission functions.
- *Timing Abstraction.* In a cycle accurate simulator, there are multiple simulation tasks, and each task makes some progress on each clock cycle, but this slows down the simulation. In a virtual prototyping experiment, one may not need such precise timing information: coarser time abstractions can be defined allowing for faster simulation.

The cornerstone of a virtual prototyping platform is the component that simulates the processor(s) of the platform, and its associated peripherals. Such simulation can be *static* or *dynamic*.

A solution usually adopted to handle time in virtual prototyping is to manage hierarchical time scales, use component abstractions where possible to gain performance, use refinement to gain accuracy where needed. Localized time abstraction may not only yield faster simulation, but facilitate also verification and synthesis (e.g. synchronous abstractions of physically distributed systems). Such an approach requires computations and communications to be harmoniously discretized and abstracted from originally heterogeneous viewpoints onto a structuring, articulating, pivot model, for concerted reasoning about time and scheduling of events in a way that ensures global system specification correctness.

In the short term these component models could be based on libraries of predefined models of different levels of abstractions. Such abstractions are common in large programming workbench for hardware modeling, such as SystemC, but less so, because of the engineering required, for virtual prototyping platforms.

The approach of team TEA provides an additional ingredient in the form of rich component interfaces. It therefore dictates to further investigate the combined use of conventional virtual prototyping libraries, defined as executable abstractions of real hardware, with executable component simulators synthesised from rich interface specifications (using, e.g., conventional compiling techniques used for synchronous programs).

## ANTIQUÉ Project-Team

### 3. Research Program

#### 3.1. Semantics

Semantics plays a central role in verification since it always serves as a basis to express the properties of interest, that need to be verified, but also additional properties, required to prove the properties of interest, or which may make the design of static analysis easier.

For instance, if we aim for a static analysis that should prove the absence of runtime error in some class of programs, the concrete semantics should define properly what error states and non error states are, and how program executions step from a state to the next one. In the case of a language like C, this includes the behavior of floating point operations as defined in the IEEE 754 standard. When considering parallel programs, this includes a model of the scheduler, and a formalization of the memory model.

In addition to the properties that are required to express the proof of the property of interest, it may also be desirable that semantics describe program behaviors in a finer manner, so as to make static analyses easier to design. For instance, it is well known that, when a state property (such as the absence of runtime error) is valid, it can be established using only a state invariant (i.e., an invariant that ignores the order in which states are visited during program executions). Yet searching for trace invariants (i.e., that take into account some properties of program execution history) may make the static analysis significantly easier, as it will allow it to make finer case splits, directed by the history of program executions. To allow for such powerful static analyses, we often resort to a *non standard semantics*, which incorporates properties that would normally be left out of the concrete semantics.

#### 3.2. Abstract interpretation and static analysis

Once a reference semantics has been fixed and a property of interest has been formalized, the definition of a static analysis requires the choice of an *abstraction*. The abstraction ties a set of *abstract predicates* to the concrete ones, which they denote. This relation is often expressed with a *concretization function* that maps each abstract element to the concrete property it stands for. Obviously, a well chosen abstraction should allow expressing the property of interest, as well as all the intermediate properties that are required in order to prove it (otherwise, the analysis would have no chance to achieve a successful verification). It should also lend itself to an efficient implementation, with efficient data-structures and algorithms for the representation and the manipulation of abstract predicates. A great number of abstractions have been proposed for all kinds of concrete data types, yet the search for new abstractions is a very important topic in static analysis, so as to target novel kinds of properties, to design more efficient or more precise static analyses.

Once an abstraction is chosen, a set of *sound abstract transformers* can be derived from the concrete semantics and that account for individual program steps, in the abstract level and without forgetting any concrete behavior. A static analysis follows as a result of this step by step approximation of the concrete semantics, when the abstract transformers are all computable. This process defines an *abstract interpretation* [13]. The case of loops requires a bit more work as the concrete semantics typically relies on a fixpoint that may not be computable in finitely many iterations. To achieve a terminating analysis we then use *widening operators* [13], which over-approximates the concrete union and ensure termination.

A static analysis defined that way always terminates and produces sound over-approximations of the programs behaviors. Yet, these results may not be precise enough for verification. This is where the art of static analysis design comes into play through, among others:

- the use of more precise, yet still efficient enough abstract domains;
- the combination of application specific abstract domains;
- the careful choice of abstract transformers and widening operators.

### 3.3. Applications of the notion of abstraction in semantics

In the previous subsections, we sketched the steps in the design of a static analyzer to infer some family of properties, which should be implementable, and efficient enough to succeed in verifying non trivial systems.

Yet, the same principles can also be applied successfully to other goals. In particular, the abstract interpretation framework should be viewed a very general tool to *compare different semantics*, not necessarily with the goal of deriving a static analyzer. Such comparisons may be used in order to prove two semantics equivalent (i.e., one is an abstraction of the other and vice versa), or that a first semantics is strictly more expressive than another one (i.e., the latter can be viewed an abstraction of the former, where the abstraction actually makes some information redundant, which cannot be recovered). A classical example of such comparison is the classification of semantics of transition systems [12], which provides a better understanding of program semantics in general. For instance, this approach can be applied to get a better understanding of the semantics of a programming language, but also to select which concrete semantics should be used as a foundation for a static analysis, or to prove the correctness of a program transformation, compilation or optimization.

### 3.4. The analysis of biological models

One of our application domains, the analysis of biological models, is not a classical target of static analysis because it aims at analyzing models instead of programs. Yet, the analysis of biological models is closely intertwined with the other application fields of our group. Firstly, abstract interpretation provides a formal understanding of the abstraction process which is inherent to the modeling process. Abstract interpretation is also used to better understand the systematic approaches which are used in the systems biology field to capture the properties of models, until getting formal, fully automatic, and scalable methods. Secondly, abstract interpretation is used to offer various semantics with different grains of abstraction, and, thus, new methods to apprehend the overall behavior of the models. Conversely, some of the methods and abstractions which are developed for biological models are inspired by the analysis of concurrent systems and by security analysis. Lastly, the analysis of biological models raises issues about differential systems, stochastic systems, and hybrid systems. Any breakthrough in these directions will likely be very important to address the important challenge of the certification of critical systems in interaction with their physical environment.

**CELTIQUE Project-Team (section vide)**

## DEDUCTEAM Team

### 3. Research Program

#### 3.1. From proof-checking to Interoperability

A new turn with Deduction modulo was taken when the idea of reasoning modulo an arbitrary equivalence relation was applied to typed  $\lambda$ -calculi with dependent types, that permits to express proofs as algorithms, using the Brouwer-Heyting-Kolmogorov interpretation and the Curry-de Bruijn-Howard correspondence [27]. It was shown in 2007, that extending the simplest  $\lambda$ -calculus with dependent types, the  $\lambda\Pi$ -calculus, with an equivalence relation (more precisely a coningruence), led to a calculus we called the  $\lambda\Pi$ -calculus modulo, that permitted to simulate many other  $\lambda$ -calculi, such as the Calculus of Constructions, designed to express proofs in specific theories.

This led to the development of a general proof-checker based on the  $\lambda\Pi$ -calculus modulo [3], that could be used to verify proofs coming from different proof systems, such as Coq [26], HOL [33], etc. To emphasize this versatility of our proof-system, we called it Dedukti —“to deduce” in Esperanto. This system is currently developed together with companion systems, Coqine, Krajono, Holide, Focalide, and Zenonide, that permits to translate proofs from Coq, HOL, Focalize, and Zenon, to Dedukti. Other tools, such as Zenon Modulo, directly output proofs that can be checked by Dedukti. Dedukti proofs can also be exported to other systems, in particular to the MMT format [37].

A thesis, which is at the root of our research effort, and which was already formulated in [32] is that proof-checkers should be theory independent. This is for instance expressed in the title of our invited talk at Icalp 2012: *A theory independent Curry-De Bruijn-Howard correspondence*. Such a theory independent proof-checker is called a *Logical Framework*.

Using a single prover to check proofs coming from different provers naturally led to investigate how these proofs could interact one with another. This issue is of prime importance because developments in proof systems are getting bigger and, unlike other communities in computer science, the proof-checking community has given little effort in the direction of standardization and interoperability. On a longer term we believe that, for each proof, we should be able to identify the systems in which it can be expressed.

#### 3.2. Automated theorem proving

Deduction modulo has originally been proposed to solve a problem in automated theorem proving and some of the early work in this area focused on the design of an automated theorem proving method called *Resolution modulo*, but this method was so complex that it was never implemented. This method was simplified in 2010 [5] and it could then be implemented. This implementation that builds on the iProver effort [36] is called iProver modulo.

iProver modulo gave surprisingly good results [4], so that we use it now to search for proofs in many areas: in the theory of classes—also known as B set theory—, on finite structures, etc. Similar ideas have also been implemented for the tableau method with in particular several extensions of the Zenon automated theorem prover. More precisely, two extensions have been realized: the first one is called **SuperZenon** [35] [30] and is an extension to superdeduction (which is a variant of Deduction modulo), and the second one is called **ZenonModulo** [28], [29] and is an extension to Deduction modulo. Both extensions have been extensively tested over first-order problems (of the TPTP library), and also provide good results in terms of number of proved problems. In particular, these tools provide good performances in set theory, so that SuperZenon has been successfully applied to verify B proof rules of Atelier B (work in collaboration with Siemens). Similarly, we plan to apply ZenonModulo in the framework of the BWare project to verify B proof obligations coming from the modeling of industrial applications.



More generally, we believe that proof-checking and automated theorem proving have a lot to learn from each other, because a proof is both a static linguistic object justifying the truth of a proposition and a dynamic process of proving this proposition.

### **3.3. Models of computation**

The idea of Deduction modulo is that computation plays a major role in the foundations of mathematics. This led us to investigate the role played by computation in other sciences, in particular in physics. Some of this work can be seen as a continuation of Gandy's [31] on the fact that the physical Church-Turing thesis is a consequence of three principles of physics, two well-known: the homogeneity of space and time, and the existence of a bound on the velocity of information, and one more speculative: the existence of a bound on the density of information.

This led us to develop physically oriented models of computations.

## GALLIUM Project-Team

### 3. Research Program

#### 3.1. Programming languages: design, formalization, implementation

Like all languages, programming languages are the media by which thoughts (software designs) are communicated (development), acted upon (program execution), and reasoned upon (validation). The choice of adequate programming languages has a tremendous impact on software quality. By “adequate”, we mean in particular the following four aspects of programming languages:

- **Safety.** The programming language must not expose error-prone low-level operations (explicit memory deallocation, unchecked array access, etc) to programmers. Further, it should provide constructs for describing data structures, inserting assertions, and expressing invariants within programs. The consistency of these declarations and assertions should be verified through compile-time verification (e.g. static type-checking) and run-time checks.
- **Expressiveness.** A programming language should manipulate as directly as possible the concepts and entities of the application domain. In particular, complex, manual encodings of domain notions into programmatic notations should be avoided as much as possible. A typical example of a language feature that increases expressiveness is pattern matching for examination of structured data (as in symbolic programming) and of semi-structured data (as in XML processing). Carried to the extreme, the search for expressiveness leads to domain-specific languages, customized for a specific application area.
- **Modularity and compositionality.** The complexity of large software systems makes it impossible to design and develop them as one, monolithic program. Software decomposition (into semi-independent components) and software composition (of existing or independently-developed components) are therefore crucial. Again, this modular approach can be applied to any programming language, given sufficient fortitude by the programmers, but is much facilitated by adequate linguistic support. In particular, reflecting notions of modularity and software components in the programming language enables compile-time checking of correctness conditions such as type correctness at component boundaries.
- **Formal semantics.** A programming language should fully and formally specify the behaviours of programs using mathematical semantics, as opposed to informal, natural-language specifications. Such a formal semantics is required in order to apply formal methods (program proof, model checking) to programs.

Our research work in language design and implementation centers on the statically-typed functional programming paradigm, which scores high on safety, expressiveness and formal semantics, complemented with full imperative features and objects for additional expressiveness, and modules and classes for compositionality. The OCaml language and system embodies many of our earlier results in this area [49]. Through collaborations, we also gained experience with several domain-specific languages based on a functional core, including distributed programming (JoCaml), XML processing (XDuce, CDuce), reactive functional programming, and hardware modeling.

#### 3.2. Type systems

Type systems [52] are a very effective way to improve programming language reliability. By grouping the data manipulated by the program into classes called types, and ensuring that operations are never applied to types over which they are not defined (e.g. accessing an integer as if it were an array, or calling a string as if it were a function), a tremendous number of programming errors can be detected and avoided, ranging from the trivial (misspelled identifier) to the fairly subtle (violation of data structure invariants). These restrictions are also very effective at thwarting basic attacks on security vulnerabilities such as buffer overflows.

The enforcement of such typing restrictions is called type-checking, and can be performed either dynamically (through run-time type tests) or statically (at compile-time, through static program analysis). We favor static type-checking, as it catches bugs earlier and even in rarely-executed parts of the program, but note that not all type constraints can be checked statically if static type-checking is to remain decidable (i.e. not degenerate into full program proof). Therefore, all typed languages combine static and dynamic type-checking in various proportions.

Static type-checking amounts to an automatic proof of partial correctness of the programs that pass the compiler. The two key words here are *partial*, since only type safety guarantees are established, not full correctness; and *automatic*, since the proof is performed entirely by machine, without manual assistance from the programmer (beyond a few, easy type declarations in the source). Static type-checking can therefore be viewed as the poor man's formal methods: the guarantees it gives are much weaker than full formal verification, but it is much more acceptable to the general population of programmers.

### 3.2.1. *Type systems and language design.*

Unlike most other uses of static program analysis, static type-checking rejects programs that it cannot prove safe. Consequently, the type system is an integral part of the language design, as it determines which programs are acceptable and which are not. Modern typed languages go one step further: most of the language design is determined by the *type structure* (type algebra and typing rules) of the language and intended application area. This is apparent, for instance, in the XDuce and CDuce domain-specific languages for XML transformations [46], [43], whose design is driven by the idea of regular expression types that enforce DTDs at compile-time. For this reason, research on type systems – their design, their proof of semantic correctness (type safety), the development and proof of associated type-checking and inference algorithms – plays a large and central role in the field of programming language research, as evidenced by the huge number of type systems papers in conferences such as Principles of Programming Languages.

### 3.2.2. *Polymorphism in type systems.*

There exists a fundamental tension in the field of type systems that drives much of the research in this area. On the one hand, the desire to catch as many programming errors as possible leads to type systems that reject more programs, by enforcing fine distinctions between related data structures (say, sorted arrays and general arrays). The downside is that code reuse becomes harder: conceptually identical operations must be implemented several times (say, copying a general array and a sorted array). On the other hand, the desire to support code reuse and to increase expressiveness leads to type systems that accept more programs, by assigning a common type to broadly similar objects (for instance, the `Object` type of all class instances in Java). The downside is a loss of precision in static typing, requiring more dynamic type checks (downcasts in Java) and catching fewer bugs at compile-time.

*Polymorphic* type systems offer a way out of this dilemma by combining precise, descriptive types (to catch more errors statically) with the ability to abstract over their differences in pieces of reusable, generic code that is concerned only with their commonalities. The paradigmatic example is parametric polymorphism, which is at the heart of all typed functional programming languages. Many forms of polymorphic typing have been studied since then. Taking examples from our group, the work of Rémy, Vouillon and Garrigue on row polymorphism [55], integrated in OCaml, extended the benefits of this approach (reusable code with no loss of typing precision) to object-oriented programming, extensible records and extensible variants. Another example is the work by Pottier on subtype polymorphism, using a constraint-based formulation of the type system [53]. Finally, the notion of “coercion polymorphism” proposed by Cretin and Rémy [3] combines and generalizes both parametric and subtyping polymorphism.

### 3.2.3. *Type inference.*

Another crucial issue in type systems research is the issue of type inference: how many type annotations must be provided by the programmer, and how many can be inferred (reconstructed) automatically by the type-checker? Too many annotations make the language more verbose and bother the programmer with unnecessary details. Too few annotations make type-checking undecidable, possibly requiring heuristics,

which is unsatisfactory. OCaml requires explicit type information at data type declarations and at component interfaces, but infers all other types.

In order to be predictable, a type inference algorithm must be complete. That is, it must not find *one*, but *all* ways of filling in the missing type annotations to form an explicitly typed program. This task is made easier when all possible solutions to a type inference problem are *instances* of a single, *principal* solution.

Maybe surprisingly, the strong requirements – such as the existence of principal types – that are imposed on type systems by the desire to perform type inference sometimes lead to better designs. An illustration of this is row variables. The development of row variables was prompted by type inference for operations on records. Indeed, previous approaches were based on subtyping and did not easily support type inference. Row variables have proved simpler than structural subtyping and more adequate for type-checking record update, record extension, and objects.

Type inference encourages abstraction and code reuse. A programmer's understanding of his own program is often initially limited to a particular context, where types are more specific than strictly required. Type inference can reveal the additional generality, which allows making the code more abstract and thus more reusable.

### 3.3. Compilation

Compilation is the automatic translation of high-level programming languages, understandable by humans, to lower-level languages, often executable directly by hardware. It is an essential step in the efficient execution, and therefore in the adoption, of high-level languages. Compilation is at the interface between programming languages and computer architecture, and because of this position has had considerable influence on the design of both. Compilers have also attracted considerable research interest as the oldest instance of symbolic processing on computers.

Compilation has been the topic of much research work in the last 40 years, focusing mostly on high-performance execution (“optimization”) of low-level languages such as Fortran and C. Two major results came out of these efforts: one is a superb body of performance optimization algorithms, techniques and methodologies; the other is the whole field of static program analysis, which now serves not only to increase performance but also to increase reliability, through automatic detection of bugs and establishment of safety properties. The work on compilation carried out in the Gallium group focuses on a less investigated topic: compiler certification.

#### 3.3.1. Formal verification of compiler correctness.

While the algorithmic aspects of compilation (termination and complexity) have been well studied, its semantic correctness – the fact that the compiler preserves the meaning of programs – is generally taken for granted. In other terms, the correctness of compilers is generally established only through testing. This is adequate for compiling low-assurance software, themselves validated only by testing: what is tested is the executable code produced by the compiler, therefore compiler bugs are detected along with application bugs. This is not adequate for high-assurance, critical software which must be validated using formal methods: what is formally verified is the source code of the application; bugs in the compiler used to turn the source into the final executable can invalidate the guarantees so painfully obtained by formal verification of the source.

To establish strong guarantees that the compiler can be trusted not to change the behavior of the program, it is necessary to apply formal methods to the compiler itself. Several approaches in this direction have been investigated, including translation validation, proof-carrying code, and type-preserving compilation. The approach that we currently investigate, called *compiler verification*, applies program proof techniques to the compiler itself, seen as a program in particular, and use a theorem prover (the Coq system) to prove that the generated code is observationally equivalent to the source code. Besides its potential impact on the critical software industry, this line of work is also scientifically fertile: it improves our semantic understanding of compiler intermediate languages, static analyses and code transformations.

### **3.4. Interface with formal methods**

Formal methods collectively refer to the mathematical specification of software or hardware systems and to the verification of these systems against these specifications using computer assistance: model checkers, theorem provers, program analyzers, etc. Despite their costs, formal methods are gaining acceptance in the critical software industry, as they are the only way to reach the required levels of software assurance.

In contrast with several other Inria projects, our research objectives are not fully centered around formal methods. However, our research intersects formal methods in the following two areas, mostly related to program proofs using proof assistants and theorem provers.

#### ***3.4.1. Software-proof codesign***

The current industrial practice is to write programs first, then formally verify them later, often at huge costs. In contrast, we advocate a codesign approach where the program and its proof of correctness are developed in interaction, and we are interested in developing ways and means to facilitate this approach. One possibility that we currently investigate is to extend functional programming languages such as OCaml with the ability to state logical invariants over data structures and pre- and post-conditions over functions, and interface with automatic or interactive provers to verify that these specifications are satisfied. Another approach that we practice is to start with a proof assistant such as Coq and improve its capabilities for programming directly within Coq.

#### ***3.4.2. Mechanized specifications and proofs for programming languages components***

We emphasize mathematical specifications and proofs of correctness for key language components such as semantics, type systems, type inference algorithms, compilers and static analyzers. These components are getting so large that machine assistance becomes necessary to conduct these mathematical investigations. We have already mentioned using proof assistants to verify compiler correctness. We are also interested in using them to specify and reason about semantics and type systems. These efforts are part of a more general research topic that is gaining importance: the formal verification of the tools that participate in the construction and certification of high-assurance software.

## **MARELLE Project-Team**

### **3. Research Program**

#### **3.1. Type theory and formalization of mathematics**

The calculus of inductive constructions is a branch of type theory that serves as a foundation for theorem proving tools, especially the Coq proof assistant. It is powerful enough to formalize complex mathematics, based on algebraic structures and operations. This is especially important as we want to produce proofs of logical properties for these algebraic structures, a goal that is only marginally addressed in most scientific computation systems.

The calculus of inductive constructions also makes it possible to write algorithms as recursive functional programs which manipulate tree-like data structures. A third important characteristic of this calculus is that it is also a language for manipulating proofs. All this makes this calculus a tool of choice for our investigations. However, this language still is the object of improvements and part of our work focusses on these improvements.

#### **3.2. Verification of scientific algorithms**

To produce certified algorithms, we use the following approach: instead of attempting to prove properties of an existing program written in a conventional programming language such as C or Java, we produce new programs in the calculus of constructions whose correctness is an immediate consequence of their construction. This has several advantages. First, we work at a high level of abstraction, independently of the target implementation language. Secondly, we concentrate on specific characteristics of the algorithm, and abstract away from the rest (for instance, we abstract away from memory management or data implementation strategies). Therefore, we are able to address more high-level mathematics and to express more general properties without being overwhelmed by implementation details.

However, this approach also presents a few drawbacks. For instance, the calculus of constructions usually imposes that recursive programs should explicitly terminate for all inputs. For some algorithms, we need to use advanced concepts (for instance, well-founded relations) to make the property of termination explicit, and proofs of correctness become especially difficult in this setting.

#### **3.3. Programming language semantics**

To bridge the gap between our high-level descriptions of algorithms and conventional programming languages, we investigate the algorithms that are present in programming language implementations, for instance algorithms that are used in a compiler or a static analysis tool. When working on these algorithms, we usually base our work on the semantic description of the programming language. The properties that we attempt to prove for an algorithm are, for example, that an optimization respects the meaning of programs or that the programs produced are free of some unwanted behavior. In practice, we rely on this study of programming language semantics to propose extensions to theorem proving tools or to verify that compilers for conventional programming languages are exempt from bugs.

## MEXICO Project-Team

### 3. Research Program

#### 3.1. Concurrency

**Participants:** Benedikt Bollig, Thomas Chatain, Paul Gastin, Stefan Haar, Serge Haddad, Stefan Schwoon.

Concurrency; Semantics; Automatic Control ; Diagnosis ; Verification

**Concurrency:** Property of systems allowing some interacting processes to be executed in parallel.

**Diagnosis:** The process of deducing from a partial observation of a system aspects of the internal states or events of that system; in particular, *fault diagnosis* aims at determining whether or not some non-observable fault event has occurred.

**Conformance Testing:** Feeding dedicated input into an implemented system  $IS$  and deducing, from the resulting output of  $I$ , whether  $I$  respects a formal specification  $S$ .

##### 3.1.1. Introduction

It is well known that, whatever the intended form of analysis or control, a *global* view of the system state leads to overwhelming numbers of states and transitions, thus slowing down algorithms that need to explore the state space. Worse yet, it often blurs the mechanics that are at work rather than exhibiting them. Conversely, respecting concurrency relations avoids exhaustive enumeration of interleavings. It allows us to focus on ‘essential’ properties of non-sequential processes, which are expressible with causal precedence relations. These precedence relations are usually called causal (partial) orders. Concurrency is the explicit absence of such a precedence between actions that do not have to wait for one another. Both causal orders and concurrency are in fact essential elements of a specification. This is especially true when the specification is constructed in a distributed and modular way. Making these ordering relations explicit requires to leave the framework of state/interleaving based semantics. Therefore, we need to develop new dedicated algorithms for tasks such as conformance testing, fault diagnosis, or control for distributed discrete systems. Existing solutions for these problems often rely on centralized sequential models which do not scale up well.

##### 3.1.2. Diagnosis

**Participants:** Benedikt Bollig, Stefan Haar, Serge Haddad, Stefan Schwoon.

*Fault Diagnosis* for discrete event systems is a crucial task in automatic control. Our focus is on *event oriented* (as opposed to *state oriented*) model-based diagnosis, asking e.g. the following questions: given a - potentially large - *alarm pattern* formed of observations,

- what are the possible *fault scenarios* in the system that *explain* the pattern ?
- Based on the observations, can we deduce whether or not a certain - invisible - fault has actually occurred ?

Model-based diagnosis starts from a discrete event model of the observed system - or rather, its relevant aspects, such as possible fault propagations, abstracting away other dimensions. From this model, an extraction or unfolding process, guided by the observation, produces recursively the explanation candidates.

In asynchronous partial-order based diagnosis with Petri nets [51], [52], [56], one unfolds the *labelled product* of a Petri net model  $\mathcal{N}$  and an observed alarm pattern  $\mathcal{A}$ , also in Petri net form. We obtain an acyclic net giving partial order representation of the behaviors compatible with the alarm pattern. A recursive online procedure filters out those runs (*configurations*) that explain *exactly*  $\mathcal{A}$ . The Petri-net based approach generalizes to dynamically evolving topologies, in dynamical systems modeled by graph grammars, see [35]

### 3.1.2.1. Observability and Diagnosability

Diagnosis algorithms have to operate in contexts with low observability, i.e., in systems where many events are invisible to the supervisor. Checking *observability* and *diagnosability* for the supervised systems is therefore a crucial and non-trivial task in its own right. Analysis of the relational structure of occurrence nets allows us to check whether the system exhibits sufficient visibility to allow diagnosis. Developing efficient methods for both verification of *diagnosability checking* under concurrency, and the *diagnosis* itself for distributed, composite and asynchronous systems, is an important field for *MEXICO*.

### 3.1.2.2. Distribution

Distributed computation of unfoldings allows one to factor the unfolding of the global system into smaller *local* unfoldings, by local supervisors associated with sub-networks and communicating among each other. In [52], [37], elements of a methodology for distributed computation of unfoldings between several supervisors, underwritten by algebraic properties of the category of Petri nets have been developed. Generalizations, in particular to Graph Grammars, are still to be done.

Computing diagnosis in a distributed way is only one aspect of a much vaster topic, that of *distributed diagnosis* (see [48], [60]). In fact, it involves a more abstract and often indirect reasoning to conclude whether or not some given invisible fault has occurred. Combination of local scenarios is in general not sufficient: the global system may have behaviors that do not reveal themselves as faulty (or, dually, non-faulty) on any local supervisor's domain (compare [34], [40]). Rather, the local diagnosers have to join all *information* that is available to them locally, and then deduce collectively further information from the combination of their views. In particular, even the *absence* of fault evidence on all peers may allow to deduce fault occurrence jointly, see [64], [65]. Automating such procedures for the supervision and management of distributed and locally monitored asynchronous systems is a long-term goal to which *MEXICO* hopes to contribute.

### 3.1.3. Contextual nets

**Participant:** Stefan Schwoon.

Assuring the correctness of concurrent systems is notoriously difficult due to the many unforeseeable ways in which the components may interact and the resulting state-space explosion. A well-established approach to alleviate this problem is to model concurrent systems as Petri nets and analyse their unfoldings, essentially an acyclic version of the Petri net whose simpler structure permits easier analysis [50].

However, Petri nets are inadequate to model concurrent read accesses to the same resource. Such situations often arise naturally, for instance in concurrent databases or in asynchronous circuits. The encoding tricks typically used to model these cases in Petri nets make the unfolding technique inefficient. Contextual nets, which explicitly do model concurrent read accesses, address this problem. Their accurate representation of concurrency makes contextual unfoldings up to exponentially smaller in certain situations. An abstract algorithm for contextual unfoldings was first given in [36]. In recent work, we further studied this subject from a theoretical and practical perspective, allowing us to develop concrete, efficient data structures and algorithms and a tool (Cunf) that improves upon existing state of the art. This work led to the PhD thesis of César Rodríguez in 2014.

Contextual unfoldings deal well with two sources of state-space explosion: concurrency and shared resources. Recently, we proposed an improved data structure, called *contextual merged processes* (CMP) to deal with a third source of state-space explosion, i.e. sequences of choices. The work on CMP [66] is currently at an abstract level. In the short term, we want to put this work into practice, requiring some theoretical groundwork, as well as programming and experimentation.

Another well-known approach to verifying concurrent systems is *partial-order reduction*, exemplified by the tool SPIN. Although it is known that both partial-order reduction and unfoldings have their respective strengths and weaknesses, we are not aware of any conclusive comparison between the two techniques. Spin comes with a high-level modeling language having an explicit notion of processes, communication channels, and variables. Indeed, the reduction techniques implemented in Spin exploit the specific properties of these features. On the other side, while there exist highly efficient tools for unfoldings, Petri nets are a relatively general low-level



formalism, so these techniques do not exploit properties of higher language features. Our work on contextual unfoldings and CMPs represents a first step to make unfoldings exploit richer models. In the long run, we wish raise the unfolding technique to a suitable high-level modelling language and develop appropriate tool support.

### 3.1.4. Dynamic and parameterized concurrent systems

**Participants:** Benedikt Bollig, Paul Gastin.

In the past few years, our research has focused on concurrent systems where the architecture, which provides a set of processes and links between them, is *static* and *fixed in advance*. However, the assumption that the set of processes is fixed somehow seems to hinder the application of formal methods in practice. It is not appropriate in areas such as mobile computing or ad-hoc networks. In concurrent programming, it is actually perfectly natural to design a program, and claim its correctness, independently of the number of processes that participate in its execution. There are, essentially, two kinds of systems that fall into this category. When the process architecture is static but unknown, it is a parameter of the system; we then call a system *parameterized*. When, on the other hand, the process architecture is generated at runtime (i.e., process creation is a communication primitive), we say that a system is *dynamic*. Though parameterized and dynamic systems have received increasing interest in recent years, there is, by now, no canonical approach to modeling and verifying such systems. Our research program aims at the development of *a theory of parameterized and dynamic concurrent systems*. More precisely, our goal is a *unifying* theory that lays algebraic, logical, and automata-theoretic foundations to support and facilitate the study of parameterized and dynamic concurrent systems. Such theories indeed exist in non-parameterized settings where the number of processes and the way they are connected are fixed in advance. However, parameterized and dynamic systems lack such foundations and often restrict to very particular models with specialized verification techniques.

### 3.1.5. Testing

**Participants:** Benedikt Bollig, Paul Gastin, Stefan Haar.

#### 3.1.5.1. Introduction

The gap between specification and implementation is at the heart of research on formal testing. The general *conformance testing problem* can be defined as follows: Does an implementation  $\mathcal{M}'$  conform a given specification  $\mathcal{M}$ ? Here, both  $\mathcal{M}$  and  $\mathcal{M}'$  are assumed to have input and output channels. The formal model  $\mathcal{M}$  of the specification is entirely known and can be used for analysis. On the other hand, the implementation  $\mathcal{M}'$  is unknown but interacts with the environment through observable input and output channels. So the behavior of  $\mathcal{M}'$  is partially controlled by input streams, and partially observable via output streams. The Testing problem consists in computing, from the knowledge of  $\mathcal{M}$ , *input streams* for  $\mathcal{M}'$  such that observation of the resulting output streams from  $\mathcal{M}'$  allows to determine whether  $\mathcal{M}'$  conforms to  $\mathcal{M}$  as intended.

In this project, we focus on distributed or asynchronous versions of the conformance testing problem. There are two main difficulties. First, due to the distributed nature of the system, it may not be possible to have a unique global observer for the outcome of a test. Hence, we may need to use *local* observers which will record only *partial views* of the execution. Due to this, it is difficult or even impossible to reconstruct a coherent global execution. The second difficulty is the lack of global synchronization in distributed asynchronous systems. Up to now, models were described with I/O automata having a centralized control, hence inducing global synchronizations.

#### 3.1.5.2. Asynchronous Testing

Since 2006 and in particular during his sabbatical stay at the University of Ottawa, Stefan Haar has been working with Guy-Vincent Jourdan and Gregor v. Bochmann of UOttawa and Claude Jard of IRISA on asynchronous testing. In the synchronous (sequential) approach, the model is described by an I/O automaton with a centralized control and transitions labeled with individual input or output actions. This approach has known limitations when inputs and outputs are distributed over remote sites, a feature that is characteristic of, e.g., web computing. To account for concurrency in the system, they have developed in [58], [41] asynchronous conformance testing for automata with transitions labeled with (finite) partial orders of I/O. Intuitively, this is

a “big step” semantics where each step allows concurrency but the system is synchronized before the next big step. This is already an important improvement on the synchronous setting. The non-trivial challenge is now to cope with fully asynchronous specifications using models with decentralized control such as Petri nets.

### 3.1.5.3. Near Future

Completion of asynchronous testing in the setting without any big-step synchronization, and an improved understanding of the relations and possible interconnections between local (i.e. distributed) and asynchronous (centralized) testing. This has been the objective of the *TECSTES* project (2011-2014), funded by a DIGITEO *DIM/LSC* grant, and which involved Hernán Ponce de León and Stefan Haar of *MExICo*, and Delphine Longuet at LRI, University Paris-Sud/Orsay. We have extended several well known conformance (ioco style) relations for sequential models to models that can handle concurrency (labeled event structures). Two semantics (interleaving and partial order) were presented for every relation. With the interleaving semantics, the relations we obtained boil down to the same relations defined for labeled transition systems, since they focus on sequences of actions. The only advantage of using labeled event structures as a specification formalism for testing remains in the conciseness of the concurrent model with respect to a sequential one. As far as testing is concerned, the benefit is low since every interleaving has to be tested. By contrast, under the partial order semantics, the relations we obtain allow to distinguish explicitly implementations where concurrent actions are implemented concurrently, from those where they are interleaved, i.e. implemented sequentially. Therefore, these relations will be of interest when designing distributed systems, since the natural concurrency between actions that are performed in parallel by different processes can be taken into account. In particular, the fact of being unable to control or observe the order between actions taking place on different processes will not be considered as an impediment for testing. We have developed a complete testing framework for concurrent systems, which included the notions of test suites and test cases. We studied what kind of systems are testable in such a framework, and we have proposed sufficient conditions for obtaining a complete test suite as well as an algorithm to construct a test suite with such properties.

A mid-to long term goal (which may or may not be addressed by *MExICo* depending on the availability of staff for this subject) is the comprehensive formalization of testing and testability in asynchronous systems with distributed architecture and test protocols.

## 3.2. Interaction

**Participants:** Benedikt Bollig, Thomas Chatain, Paul Gastin, Stefan Haar, Serge Haddad.

### 3.2.1. Introduction

Systems and services exhibit non-trivial *interaction* between specialized and heterogeneous components. This interplay is challenging for several reasons. On one hand, a coordinated interplay of several components is required, though each has only a limited, partial view of the system’s configuration. We refer to this problem as *distributed synthesis* or *distributed control*. An aggravating factor is that the structure of a component might be semi-transparent, which requires a form of *grey box management*.

Interaction, one of the main characteristics of systems under consideration, often involves an environment that is not under the control of cooperating services. To achieve a common goal, the services need to agree upon a strategy that allows them to react appropriately regardless of the interactions with the environment. Clearly, the notions of opponents and strategies fall within *game theory*, which is naturally one of our main tools in exploring interaction. We will apply to our problems techniques and results developed in the domains of distributed games and of games with partial information. We will consider also new problems on games that arise from our applications.

### 3.2.2. Distributed Control

**Participants:** Benedikt Bollig, Thomas Chatain, Paul Gastin, Stefan Haar.

Program synthesis, as introduced by Church [47] aims at deriving directly an implementation from a specification, allowing the implementation to be correct by design. When the implementation is already at hand but choices remain to be resolved at run time then the problem becomes controller synthesis. Both program and controller synthesis have been extensively studied for sequential systems. In a distributed setting, we need to synthesize a distributed program or distributed controllers that interact locally with the system components. The main difficulty comes from the fact that the local controllers/programs have only a partial view of the entire system. This is also an old problem largely considered undecidable in most settings [63], [59], [62], [53], [55].

Actually, the main undecidability sources come from the fact that this problem was addressed in a synchronous setting using global runs viewed as sequences. In a truly distributed system where interactions are asynchronous we have recently obtained encouraging decidability results [54], [45]. This is a clear witness where concurrency may be exploited to obtain positive results. It is essential to specify expected properties directly in terms of causality revealed by partial order models of executions (MSCs or Mazurkiewicz traces). We intend to develop this line of research with the ambitious aim to obtain decidability for all natural systems and specifications. More precisely, we will identify natural hypotheses both on the architecture of our distributed system and on the specifications under which the distributed program/controller synthesis problem is decidable. This should open the way to important applications, e.g., for distributed control of embedded systems.

### 3.2.3. Adaptation and Grey box management

**Participants:** Stefan Haar, Serge Haddad.

Contrary to mainframe systems or monolithic applications of the past, we are experiencing and using an increasing number of services that are performed not by one provider but rather by the interaction and cooperation of many specialized components. As these components come from different providers, one can no longer assume all of their internal technologies to be known (as it is the case with proprietary technology). Thus, in order to compose e.g. orchestrated services over the web, to determine violations of specifications or contracts, to adapt existing services to new situations etc, one needs to analyze the interaction behavior of *boxes* that are known only through their public interfaces. For their semi-transparent-semi-opaque nature, we shall refer to them as **grey boxes**. While the concrete nature of these boxes can range from vehicles in a highway section to hotel reservation systems, the tasks of *grey box management* have universal features allowing for generalized approaches with formal methods. Two central issues emerge:

- **Abstraction:** From the designer point of view, there is a need for a trade-off between transparency (no abstraction) in order to integrate the box in different contexts and opacity (full abstraction) for security reasons.
- **Adaptation:** Since a grey box gives a partial view about the behavior of the component, even if it is not immediately useable in some context, the design of an adaptor is possible. Thus the goal is the synthesis of such an adaptor from a formal specification of the component and the environment.

Our work on direct modeling and handling of "grey boxes" via modal models (see [49]) was halted when Dorsaf El-Hog stopped her PhD work to leave academia, and has not resumed for lack of staff. However, it should be noted that semi-transparent system management in a larger sense remains an active field for the team, witness in particular our work on diagnosis and testing.

## 3.3. Management of Quantitative Behavior

**Participants:** Benedikt Bollig, Thomas Chatain, Paul Gastin, Stefan Haar, Serge Haddad.

### 3.3.1. Introduction

Besides the logical functionalities of programs, the *quantitative* aspects of component behavior and interaction play an increasingly important role.

- *Real-time* properties cannot be neglected even if time is not an explicit functional issue, since transmission delays, parallelism, etc. can lead to time-outs striking, and thus change even the logical course of processes. Again, this phenomenon arises in telecommunications and web services, but also in transport systems.
- In the same contexts, *probabilities* need to be taken into account, for many diverse reasons such as unpredictable functionalities, or because the outcome of a computation may be governed by race conditions.
- Last but not least, constraints on *cost* cannot be ignored, be it in terms of money or any other limited resource, such as memory space or available CPU time.

Traditional mainframe systems were proprietary and (essentially) localized; therefore, impact of delays, unforeseen failures, etc. could be considered under the control of the system manager. It was therefore natural, in verification and control of systems, to focus on *functional* behavior entirely.

With the increase in size of computing system and the growing degree of compositionality and distribution, quantitative factors enter the stage:

- calling remote services and transmitting data over the web creates *delays*;
- remote or non-proprietary components are not “deterministic”, in the sense that their behavior is uncertain.

*Time* and *probability* are thus parameters that management of distributed systems must be able to handle; along with both, the *cost* of operations is often subject to restrictions, or its minimization is at least desired. The mathematical treatment of these features in distributed systems is an important challenge, which *MExICO* is addressing; the following describes our activities concerning probabilistic and timed systems. Note that cost optimization is not a current activity but enters the picture in several intended activities.

### 3.3.2. Probabilistic distributed Systems

**Participants:** Stefan Haar, Serge Haddad, Claudine Picaronny.

#### 3.3.2.1. Non-sequential probabilistic processes

Practical fault diagnosis requires to select explanations of *maximal likelihood*. For partial-order based diagnosis, this leads therefore to the question what the probability of a given partially ordered execution is. In Benveniste et al. [39], [32], we presented a model of stochastic processes, whose trajectories are partially ordered, based on local branching in Petri net unfoldings; an alternative and complementary model based on Markov fields is developed in [57], which takes a different view on the semantics and overcomes the first model’s restrictions on applicability.

Both approaches abstract away from real time progress and randomize choices in *logical* time. On the other hand, the relative speed - and thus, indirectly, the real-time behavior of the system’s local processes - are crucial factors determining the outcome of probabilistic choices, even if non-determinism is absent from the system.

In another line of research [43] we have studied the likelihood of occurrence of non-sequential runs under random durations in a stochastic Petri net setting. It remains to better understand the properties of the probability measures thus obtained, to relate them with the models in logical time, and exploit them e.g. in *diagnosis*.

#### 3.3.2.2. Distributed Markov Decision Processes

**Participant:** Serge Haddad.

Distributed systems featuring non-deterministic and probabilistic aspects are usually hard to analyze and, more specifically, to optimize. Furthermore, high complexity theoretical lower bounds have been established for models like partially observed Markovian decision processes and distributed partially observed Markovian decision processes. We believe that these negative results are consequences of the choice of the models rather than the intrinsic complexity of problems to be solved. Thus we plan to introduce new models in which the associated optimization problems can be solved in a more efficient way. More precisely, we start by studying connection protocols weighted by costs and we look for online and offline strategies for optimizing the mean cost to achieve the protocol. We have been cooperating on this subject with the SUMO team at Inria Rennes; in the joint work [33]; there, we strive to synthesize for a given MDP a control so as to guarantee a specific stationary behavior, rather than - as is usually done - so as to maximize some reward.

### 3.3.3. Large scale probabilistic systems

Addressing large-scale probabilistic systems requires to face state explosion, due to both the discrete part and the probabilistic part of the model. In order to deal with such systems, different approaches have been proposed:

- Restricting the synchronization between the components as in queuing networks allows to express the steady-state distribution of the model by an analytical formula called a product-form [38].
- Some methods that tackle with the combinatory explosion for discrete-event systems can be generalized to stochastic systems using an appropriate theory. For instance symmetry based methods have been generalized to stochastic systems with the help of aggregation theory [46].
- At last simulation, which works as soon as a stochastic operational semantic is defined, has been adapted to perform statistical model checking. Roughly speaking, it consists to produce a confidence interval for the probability that a random path fulfills a formula of some temporal logic [67].

We want to contribute to these three axes: (1) we are looking for product-forms related to systems where synchronization are more involved (like in Petri nets), see [2]; (2) we want to adapt methods for discrete-event systems that require some theoretical developments in the stochastic framework and, (3) we plan to address some important limitations of statistical model checking like the expressiveness of the associated logic and the handling of rare events.

### 3.3.4. Real time distributed systems

Nowadays, software systems largely depend on complex timing constraints and usually consist of many interacting local components. Among them, railway crossings, traffic control units, mobile phones, computer servers, and many more safety-critical systems are subject to particular quality standards. It is therefore becoming increasingly important to look at networks of timed systems, which allow real-time systems to operate in a distributed manner.

Timed automata are a well-studied formalism to describe reactive systems that come with timing constraints. For modeling distributed real-time systems, networks of timed automata have been considered, where the local clocks of the processes usually evolve at the same rate [61] [44]. It is, however, not always adequate to assume that distributed components of a system obey a global time. Actually, there is generally no reason to assume that different timed systems in the networks refer to the same time or evolve at the same rate. Any component is rather determined by local influences such as temperature and workload.

#### 3.3.4.1. Implementation of Real-Time Concurrent Systems

**Participants:** Thomas Chatain, Stefan Haar, Serge Haddad.

This was one of the tasks of the ANR ImpRo.

Formal models for real-time systems, like timed automata and time Petri nets, have been extensively studied and have proved their interest for the verification of real-time systems. On the other hand, the question of using these models as specifications for designing real-time systems raises some difficulties. One of those comes from the fact that the real-time constraints introduce some artifacts and because of them some syntactically correct models have a formal semantics that is clearly unrealistic. One famous situation is the case of Zeno executions, where the formal semantics allows the system to do infinitely many actions in finite time. But there are other problems, and some of them are related to the distributed nature of the system. These are the ones we address here.

One approach to implementability problems is to formalize either syntactical or behavioral requirements about what should be considered as a reasonable model, and reject other models. Another approach is to adapt the formal semantics such that only realistic behaviors are considered.

These techniques are preliminaries for dealing with the problem of implementability of models. Indeed implementing a model may be possible at the cost of some transformation, which make it suitable for the target device. By the way these transformations may be of interest for the designer who can now use high-level features in a model of a system or protocol, and rely on the transformation to make it implementable.

We aim at formalizing and automating translations that preserve both the timed semantics and the concurrent semantics. This effort is crucial for extending concurrency-oriented methods for logical time, in particular for exploiting partial order properties. In fact, validation and management - in a broad sense - of distributed systems is not realistic *in general* without understanding and control of their real-time dependent features; the link between real-time and logical-time behaviors is thus crucial for many aspects of *MEXICO*'s work.

### 3.3.5. *Weighted Automata and Weighted Logics*

**Participants:** Benedikt Bollig, Paul Gastin.

Time and probability are only two facets of quantitative phenomena. A generic concept of adding weights to qualitative systems is provided by the theory of weighted automata [31]. They allow one to treat probabilistic or also reward models in a unified framework. Unlike finite automata, which are based on the Boolean semiring, weighted automata build on more general structures such as the natural or real numbers (equipped with the usual addition and multiplication) or the probabilistic semiring. Hence, a weighted automaton associates with any possible behavior a weight beyond the usual Boolean classification of “acceptance” or “non-acceptance”. Automata with weights have produced a well-established theory and come, e.g., with a characterization in terms of rational expressions, which generalizes the famous theorem of Kleene in the unweighted setting. Equipped with a solid theoretical basis, weighted automata finally found their way into numerous application areas such as natural language processing and speech recognition, or digital image compression.

What is still missing in the theory of weighted automata are satisfactory connections with verification-related issues such as (temporal) logic and bisimulation that could lead to a general approach to corresponding satisfiability and model-checking problems. A first step towards a more satisfactory theory of weighted systems was done in [42]. That paper, however, does not give definite answers to all the aforementioned problems. It identifies directions for future research that we will be tackling.

## PARSIFAL Project-Team

### 3. Research Program

#### 3.1. General overview

There are two broad approaches for computational specifications. In the *computation as model* approach, computations are encoded as mathematical structures containing nodes, transitions, and state. Logic is used to *describe* these structures, that is, the computations are used as models for logical expressions. Intensional operators, such as the modals of temporal and dynamic logics or the triples of Hoare logic, are often employed to express propositions about the change in state.

The *computation as deduction* approach, in contrast, expresses computations logically, using formulas, terms, types, and proofs as computational elements. Unlike the model approach, general logical apparatus such as cut-elimination or automated deduction becomes directly applicable as tools for defining, analyzing, and animating computations. Indeed, we can identify two main aspects of logical specifications that have been very fruitful:

- *Proof normalization*, which treats the state of a computation as a proof term and computation as normalization of the proof terms. General reduction principles such as  $\beta$ -reduction or cut-elimination are merely particular forms of proof normalization. Functional programming is based on normalization [69], and normalization in different logics can justify the design of new and different functional programming languages [41].
- *Proof search*, which views the state of a computation as a structured collection of formulas, known as a *sequent*, and proof search in a suitable sequent calculus as encoding the dynamics of the computation. Logic programming is based on proof search [75], and different proof search strategies can be used to justify the design of new and different logic programming languages [73].

While the distinction between these two aspects is somewhat informal, it helps to identify and classify different concerns that arise in computational semantics. For instance, confluence and termination of reductions are crucial considerations for normalization, while unification and strategies are important for search. A key challenge of computational logic is to find means of uniting or reorganizing these apparently disjoint concerns.

An important organizational principle is structural proof theory, that is, the study of proofs as syntactic, algebraic and combinatorial objects. Formal proofs often have equivalences in their syntactic representations, leading to an important research question about *canonicity* in proofs – when are two proofs “essentially the same?” The syntactic equivalences can be used to derive normal forms for proofs that illuminate not only the proofs of a given formula, but also its entire proof search space. The celebrated *focusing* theorem of Andreoli [43] identifies one such normal form for derivations in the sequent calculus that has many important consequences both for search and for computation. The combinatorial structure of proofs can be further explored with the use of *deep inference*; in particular, deep inference allows access to simple and manifestly correct cut-elimination procedures with precise complexity bounds.

Type theory is another important organizational principle, but most popular type systems are generally designed for either search or for normalization. To give some examples, the Coq system [85] that implements the Calculus of Inductive Constructions (CIC) is designed to facilitate the expression of computational features of proofs directly as executable functional programs, but general proof search techniques for Coq are rather primitive. In contrast, the Twelf system [80] that is based on the LF type theory (a subsystem of the CIC), is based on relational specifications in canonical form (*i.e.*, without redexes) for which there are sophisticated automated reasoning systems such as meta-theoretic analysis tools, logic programming engines, and inductive theorem provers. In recent years, there has been a push towards combining search and normalization in the same type-theoretic framework. The Beluga system [81], for example, is an extension of the LF type theory with a purely computational meta-framework where operations on inductively defined LF objects can be expressed as functional programs.

The Parsifal team investigates both the search and the normalization aspects of computational specifications using the concepts, results, and insights from proof theory and type theory.

### 3.2. Inductive and co-inductive reasoning

The team has spent a number of years in designing a strong new logic that can be used to reason (inductively and co-inductively) on syntactic expressions containing bindings. This work is based on earlier work by McDowell, Miller, and Tiu [71] [70] [76] [86], and on more recent work by Gacek, Miller, and Nadathur [4] [56]. The Parsifal team, along with our colleagues in Minneapolis, Canberra, Singapore, and Cachem, have been building two tools that exploit the novel features of this logic. These two systems are the following.

- Abella, which is an interactive theorem prover for the full logic.
- Bedwyr, which is a model checker for the “finite” part of the logic.

We have used these systems to provide formalize reasoning of a number of complex formal systems, ranging from programming languages to the  $\lambda$ -calculus and  $\pi$ -calculus.

Since 2014, the Abella system has been extended with a number of new features. A number of new significant examples have been implemented in Abella and an extensive tutorial for it has been written [1].

### 3.3. Developing a foundational approach to defining proof evidence

The team is developing a framework for defining the semantics of proof evidence. With this framework, implementers of theorem provers can output proof evidence in a format of their choice: they will only need to be able to formally define that evidence’s semantics. With such semantics provided, proof checkers can then check alleged proofs for correctness. Thus, anyone who needs to trust proofs from various provers can put their energies into designing trustworthy checkers that can execute the semantic specification.

In order to provide our framework with the flexibility that this ambitious plan requires, we have based our design on the most recent advances within the theory of proofs. For a number of years, various team members have been contributing to the design and theory of *focused proof systems* [45] [48] [49] [50] [59] [67] [68] and we have adopted such proof systems as the corner stone for our framework.

We have also been working for a number of years on the implementation of computational logic systems, involving, for example, both unification and backtracking search. As a result, we are also building an early and reference implementation of our semantic definitions.

### 3.4. Deep inference

Deep inference [61], [63] is a novel methodology for presenting deductive systems. Unlike traditional formalisms like the sequent calculus, it allows rewriting of formulas deep inside arbitrary contexts. The new freedom for designing inference rules creates a richer proof theory. For example, for systems using deep inference, we have a greater variety of normal forms for proofs than in sequent calculus or natural deduction systems. Another advantage of deep inference systems is the close relationship to categorical proof theory. Due to the deep inference design one can directly read off the morphism from the derivations. There is no need for a counter-intuitive translation.

The following research problems are investigated by members of the Parsifal team:

- Find deep inference system for richer logics. This is necessary for making the proof theoretic results of deep inference accessible to applications as they are described in the previous sections of this report.
- Investigate the possibility of focusing proofs in deep inference. As described before, focusing is a way to reduce the non-determinism in proof search. However, it is well investigated only for the sequent calculus. In order to apply deep inference in proof search, we need to develop a theory of focusing for deep inference.



### 3.5. Proof nets and atomic flows

Proof nets and atomic flows are abstract (graph-like) presentations of proofs such that all "trivial rule permutations" are quotiented away. Ideally the notion of proof net should be independent from any syntactic formalism, but most notions of proof nets proposed in the past were formulated in terms of their relation to the sequent calculus. Consequently we could observe features like "boxes" and explicit "contraction links". The latter appeared not only in Girard's proof nets [58] for linear logic but also in Robinson's proof nets [83] for classical logic. In this kind of proof nets every link in the net corresponds to a rule application in the sequent calculus.

Only recently, due to the rise of deep inference, new kinds of proof nets have been introduced that take the formula trees of the conclusions and add additional "flow-graph" information (see e.g., [6], [5] and [62]). On one side, this gives new insights in the essence of proofs and their normalization. But on the other side, all the known correctness criteria are no longer available.

This directly leads to the following research questions investigated by members of the Parsifal team:

- Finding (for classical logic) a notion of proof nets that is deductive, i.e., can effectively be used for doing proof search. An important property of deductive proof nets must be that the correctness can be checked in linear time. For the classical logic proof nets by Lamarche and Straßburger [6] this takes exponential time (in the size of the net).
- Studying the normalization of proofs in classical logic using atomic flows. Although there is no correctness criterion they allow to simplify the normalization procedure for proofs in deep inference, and additionally allow to get new insights in the complexity of the normalization.

### 3.6. Cost Models and Abstract Machines for Functional Programs

In the *proof normalization* approach, computation is usually reformulated as the evaluation of functional programs, expressed as terms in a variation over the  $\lambda$ -calculus. Thanks to its higher-order nature, this approach provides very concise and abstract specifications. Its strength is however also its weakness: the abstraction from physical machines is pushed to a level where it is no longer clear how to measure the complexity of an algorithm.

Models like Turing machines or RAM rely on atomic computational steps and thus admit quite obvious cost models for time and space. The  $\lambda$ -calculus instead relies on a single non-atomic operation,  $\beta$ -reduction, for which costs in terms of time and space are far from evident.

Nonetheless, it turns out that the number of  $\beta$ -steps is a reasonable time cost model, i.e., it is polynomially related to those of Turing machines and RAM. For the special case of *weak evaluation* (i.e., reducing only  $\beta$ -steps that are not under abstractions)—which is used to model functional programming languages—this is a relatively old result due to Blleloch and Greiner [46] (1995). It is only very recently (2014) that the strong case—used in the implementation models of proof assistants—has been solved by Accattoli and Dal Lago [42].

With the recent recruitment of Accattoli, the team's research has expanded in this direction. The topics under investigations are:

1. *Complexity of Abstract Machines.* Bounding and comparing the overhead of different abstract machines for different evaluation schemas (weak/strong call-by-name/value/need  $\lambda$ -calculi) with respect to the cost model. The aim is the development of a complexity-aware theory of the implementation of functional programs.
2. *Reasonable Space Cost Models.* Essentially nothing is known about reasonable space cost models. It is known, however, that environment-based execution model—which are the mainstream technology for functional programs—do not provide an answer. We are exploring the use of the non-standard implementation models provided by Girard's Geometry of Interaction to address this question.

## PL.R2 Project-Team

### 3. Research Program

#### 3.1. Proof theory and the Curry-Howard correspondence

##### 3.1.1. *Proofs as programs*

Proof theory is the branch of logic devoted to the study of the structure of proofs. An essential contributor to this field is Gentzen [57] who developed in 1935 two logical formalisms that are now central to the study of proofs. These are the so-called “natural deduction”, a syntax that is particularly well-suited to simulate the intuitive notion of reasoning, and the so-called “sequent calculus”, a syntax with deep geometric properties that is particularly well-suited for proof automation.

Proof theory gained a remarkable importance in computer science when it became clear, after genuine observations first by Curry in 1958 [52], then by Howard and de Bruijn at the end of the 60’s [70], [89], that proofs had the very same structure as programs: for instance, natural deduction proofs can be identified as typed programs of the ideal programming language known as  $\lambda$ -calculus.

This proofs-as-programs correspondence has been the starting point to a large spectrum of researches and results contributing to deeply connect logic and computer science. In particular, it is from this line of work that Coquand and Huet’s Calculus of Constructions [49], [50] stemmed out – a formalism that is both a logic and a programming language and that is at the source of the Coq system [87].

##### 3.1.2. *Towards the calculus of constructions*

The  $\lambda$ -calculus, defined by Church [48], is a remarkably succinct model of computation that is defined via only three constructions (abstraction of a program with respect to one of its parameters, reference to such a parameter, application of a program to an argument) and one reduction rule (substitution of the formal parameter of a program by its effective argument). The  $\lambda$ -calculus, which is Turing-complete, i.e. which has the same expressiveness as a Turing machine (there is for instance an encoding of numbers as functions in  $\lambda$ -calculus), comes with two possible semantics referred to as call-by-name and call-by-value evaluations. Of these two semantics, the first one, which is the simplest to characterise, has been deeply studied in the last decades [44].

To explain the Curry-Howard correspondence, it is important to distinguish between intuitionistic and classical logic: following Brouwer at the beginning of the 20<sup>th</sup> century, classical logic is a logic that accepts the use of reasoning by contradiction while intuitionistic logic proscribes it. Then, Howard’s observation is that the proofs of the intuitionistic natural deduction formalism exactly coincide with programs in the (simply typed)  $\lambda$ -calculus.

A major achievement has been accomplished by Martin-Löf who designed in 1971 a formalism, referred to as modern type theory, that was both a logical system and a (typed) programming language [80].

In 1985, Coquand and Huet [49], [50] in the Formel team of Inria-Rocquencourt explored an alternative approach based on Girard-Reynolds’ system  $F$  [58], [83]. This formalism, called the Calculus of Constructions, served as logical foundation of the first implementation of Coq in 1984. Coq was called CoC at this time.

##### 3.1.3. *The Calculus of Inductive Constructions*

The first public release of CoC dates back to 1989. The same project-team developed the programming language Caml (nowadays called OCaml and coordinated by the Gallium team) that provided the expressive and powerful concept of algebraic data types (a paragon of it being the type of lists). In CoC, it was possible to simulate algebraic data types, but only through a not-so-natural not-so-convenient encoding.

In 1989, Coquand and Paulin [51] designed an extension of the Calculus of Constructions with a generalisation of algebraic types called inductive types, leading to the Calculus of Inductive Constructions (CIC) that started to serve as a new foundation for the Coq system. This new system, which got its current definitive name Coq, was released in 1991.

In practice, the Calculus of Inductive Constructions derives its strength from being both a logic powerful enough to formalise all common mathematics (as set theory is) and an expressive richly-typed functional programming language (like ML but with a richer type system, no effects and no non-terminating functions).

## 3.2. The development of Coq

Since 1984, about 40 persons have contributed to the development of Coq, out of which 7 persons have contributed to bring the system to the place it is now. First Thierry Coquand through his foundational theoretical ideas, then Gérard Huet who developed the first prototypes with Thierry Coquand and who headed the Coq group until 1998, then Christine Paulin who was the main actor of the system based on the CIC and who headed the development group from 1998 to 2006. On the programming side, important steps were made by Chet Murthy who raised Coq from the prototypical state to a reasonably scalable system, Jean-Christophe Filliâtre who turned to concrete the concept of a small trustful certification kernel on which an arbitrary large system can be set up, Bruno Barras and Hugo Herbelin who, among other extensions, reorganised Coq on a new smoother and more uniform basis able to support a new round of extensions for the next decade.

The development started from the Formel team at Rocquencourt but, after Christine Paulin got a position in Lyon, it spread to École Normale Supérieure de Lyon. Then, the task force there globally moved to the University of Orsay when Christine Paulin got a new position there. On the Rocquencourt side, the part of Formel involved in ML moved to the Cristal team (now Gallium) and Formel got renamed into Coq. Gérard Huet left the team and Christine Paulin started to head a Coq team bilocalised at Rocquencourt and Orsay. Gilles Dowek became the head of the team which was renamed into LogiCal. Following Gilles Dowek who got a position at École Polytechnique, LogiCal moved to the new Inria Saclay research center. It then split again, giving birth to ProVal. At the same time, the Marelle team (formerly Lemme, formerly Croap) which has been a long partner of the Formel team, invested more and more energy in the formalisation of mathematics in Coq, while contributing importantly to the development of Coq, in particular nowadays for what regards user interfaces.

After various other spreadings resulting from where the wind pushed former PhD students, the development of Coq got multi-site with the development now realised by employees of Inria, the CNAM and Paris 7.

We next briefly describe the main components of Coq.

### 3.2.1. The underlying logic and the verification kernel

The architecture adopts the so-called de Bruijn principle: the well-delimited *kernel* of Coq ensures the correctness of the proofs validated by the system. The kernel is rather stable with modifications tied to the evolution of the underlying Calculus of Inductive Constructions formalism. The kernel includes an interpreter of the programs expressible in the CIC and this interpreter exists in two flavours: a customisable lazy evaluation machine written in OCaml and a call-by-value bytecode interpreter written in C dedicated to efficient computations. The kernel also provides a module system.

### 3.2.2. Programming and specification languages

The concrete user language of Coq, called *Gallina*, is a high-level language built on top of the CIC. It includes a type inference algorithm, definitions by complex pattern-matching, implicit arguments, mathematical notations and various other high-level language features. This high-level language serves both for the development of programs and for the formalisation of mathematical theories. Coq also provides a large set of commands. Gallina and the commands together forms the *Vernacular* language of Coq.

### 3.2.3. Standard library

The standard library is written in the vernacular language of Coq. There are libraries for various arithmetical structures and various implementations of numbers (Peano numbers, implementation of  $\mathbb{N}$ ,  $\mathbb{Z}$ ,  $\mathbb{Q}$  with binary digits, implementation of  $\mathbb{N}$ ,  $\mathbb{Z}$ ,  $\mathbb{Q}$  using machine words, axiomatisation of  $\mathbb{R}$ ). There are libraries for lists, list of a specified length, sorts, and for various implementations of finite maps and finite sets. There are libraries on relations, sets, orders.

### 3.2.4. Tactics

The tactics are the methods available to conduct proofs. This includes the basic inference rules of the CIC, various advanced higher level inference rules and all the automation tactics. Regarding automation, there are tactics for solving systems of equations, for simplifying ring or field expressions, for arbitrary proof search, for semi-decidability of first-order logic and so on. There is also a powerful and popular untyped scripting language for combining tactics into more complex tactics.

Note that all tactics of Coq produce proof certificates that are checked by the kernel of Coq. As a consequence, possible bugs in proof methods do not hinder the confidence in the correctness of the Coq checker. Note also that the CIC being a programming language, tactics can have their core written (and certified) in the own language of Coq if needed.

### 3.2.5. Extraction

Extraction is a component of Coq that maps programs (or even computational proofs) of the CIC to functional programs (in OCaml, Scheme or Haskell). Especially, a program certified by Coq can further be extracted to a program of a full-fledged programming language then benefiting of the efficient compilation, linking tools, profiling tools, ... of the target software.

## 3.3. Dependently typed programming languages

Dependently typed programming (shortly DTP) is an emerging concept referring to the diffuse and broadening tendency to develop programming languages with type systems able to express program properties finer than the usual information of simply belonging to specific data-types. The type systems of dependently-typed programming languages allow to express properties *dependent* of the input and the output of the program (for instance that a sorting program returns a list of same size as its argument). Typical examples of such languages were the Cayenne language, developed in the late 90's at Chalmers University in Sweden and the DML language developed at Boston. Since then, various new tools have been proposed, either as typed programming languages whose types embed equalities ( $\Omega$ mega at Portland, ATS at Boston, ...) or as hybrid logic/programming frameworks (Agda at Chalmers University, Twelf at Carnegie, Delphin at Yale, OpTT at U. Iowa, Epigram at Nottingham, ...).

DTP contributes to a general movement leading to the fusion between logic and programming. Coq, whose language is both a logic and a programming language which moreover can be extracted to pure ML code plays a role in this movement and some frameworks combining logic and programming have been proposed on top of Coq (Concoqtion at Rice and Colorado, Ynot at Harvard, Why in the ProVal team at Inria). It also connects to Hoare logic, providing frameworks where pre- and post-conditions of programs are tied with the programs.

DTP approached from the programming language side generally benefits of a full-fledged language (e.g. supporting effects) with efficient compilation. DTP approached from the logic side generally benefits of an expressive specification logic and of proof methods so as to certify the specifications. The weakness of the approach from logic however is generally the weak support for effects or partial functions.

### 3.3.1. Type-checking and proof automation

In between the decidable type systems of conventional data-types based programming languages and the full expressiveness of logically undecidable formulae, an active field of research explores a spectrum of decidable or semi-decidable type systems for possible use in dependently typed programming languages. At the beginning of the spectrum, this includes, for instance, the system  $F$ 's extension  $ML_F$  of the ML type

system or the generalisation of abstract data types with type constraints (G.A.D.T.) such as found in the Haskell programming language. At the other side of the spectrum, one finds arbitrary complex type specification languages (e.g. that a sorting function returns a list of type “sorted list”) for which more or less powerful proof automation tools exist – generally first-order ones.

### 3.4. Around and beyond the Curry-Howard correspondence

For two decades, the Curry-Howard correspondence has been limited to the intuitionistic case but since 1990, an important stimulus spurred on the community following Griffin’s discovery that this correspondence was extensible to classical logic. The community then started to investigate unexplored potential connections between computer science and logic. One of these fields is the computational understanding of Gentzen’s sequent calculus while another one is the computational content of the axiom of choice.

#### 3.4.1. Control operators and classical logic

Indeed, a significant extension of the Curry-Howard correspondence has been obtained at the beginning of the 90’s thanks to the seminal observation by Griffin [59] that some operators known as control operators were typable by the principle of double negation elimination ( $\neg\neg A \Rightarrow A$ ), a principle that enables classical reasoning.

Control operators are used to jump from one location of a program to another. They were first considered in the 60’s by Landin [76] and Reynolds [82] and started to be studied in an abstract way in the 80’s by Felleisen *et al* [55], leading to Parigot’s  $\lambda\mu$ -calculus [81], a reference calculus that is in close Curry-Howard correspondence with classical natural deduction. In this respect, control operators are fundamental pieces to establish a full connection between proofs and programs.

#### 3.4.2. Sequent calculus

The Curry-Howard interpretation of sequent calculus started to be investigated at the beginning of the 90’s. The main technicality of sequent calculus is the presence of *left introduction* inference rules, for which two kinds of interpretations are applicable. The first approach interprets left introduction rules as construction rules for a language of patterns but it does not really address the problem of the interpretation of the implication connective. The second approach, started in 1994, interprets left introduction rules as evaluation context formation rules. This line of work led in 2000 to the design by Hugo Herbelin and Pierre-Louis Curien of a symmetric calculus exhibiting deep dualities between the notion of programs and evaluation contexts and between the standard notions of call-by-name and call-by-value evaluation semantics.

#### 3.4.3. Abstract machines

Abstract machines came as an intermediate evaluation device, between high-level programming languages and the computer microprocessor. The typical reference for call-by-value evaluation of  $\lambda$ -calculus is Landin’s SECD machine [75] and Krivine’s abstract machine for call-by-name evaluation [72], [71]. A typical abstract machine manipulates a state that consists of a program in some environment of bindings and some evaluation context traditionally encoded into a “stack”.

#### 3.4.4. Delimited control

Delimited control extends the expressiveness of control operators with effects: the fundamental result here is a completeness result by Filinski [56]: any side-effect expressible in monadic style (and this covers references, exceptions, states, dynamic bindings, ...) can be simulated in  $\lambda$ -calculus equipped with delimited control.

### 3.5. Effective higher-dimensional algebra

#### 3.5.1. Higher-dimensional algebra

Like ordinary categories, higher-dimensional categorical structures originate in algebraic topology. Indeed,  $\infty$ -groupoids have been initially considered as a unified point of view for all the information contained in the

homotopy groups of a topological space  $X$ : the *fundamental*  $\infty$ -groupoid  $\Pi(X)$  of  $X$  contains the elements of  $X$  as 0-dimensional cells, continuous paths in  $X$  as 1-cells, homotopies between continuous paths as 2-cells, and so on. This point of view translates a topological problem (to determine if two given spaces  $X$  and  $Y$  are homotopically equivalent) into an algebraic problem (to determine if the fundamental groupoids  $\Pi(X)$  and  $\Pi(Y)$  are equivalent).

In the last decades, the importance of higher-dimensional categories has grown fast, mainly with the new trend of *categorification* that currently touches algebra and the surrounding fields of mathematics. Categorification is an informal process that consists in the study of higher-dimensional versions of known algebraic objects (such as higher Lie algebras in mathematical physics [43]) and/or of “weakened” versions of those objects, where equations hold only up to suitable equivalences (such as weak actions of monoids and groups in representation theory [54]).

Since a few years, the categorification process has reached logic, with the introduction of homotopy type theory. After a preliminary result that had identified categorical structures in type theory [69], it has been observed recently that the so-called “identity types” are naturally equipped with a structure of  $\infty$ -groupoid: the 1-cells are the proofs of equality, the 2-cells are the proofs of equality between proofs of equality, and so on. The striking resemblance with the fundamental  $\infty$ -groupoid of a topological space led to the conjecture that homotopy type theory could serve as a replacement of set theory as a foundational language for different fields of mathematics, and homotopical algebra in particular.

### 3.5.2. Higher-dimensional rewriting

Higher-dimensional categories are algebraic structures that contain, in essence, computational aspects. This has been recognised by Street [86], and independently by Burroni [47], when they have introduced the concept of *computad* or *polygraph* as combinatorial descriptions of higher categories. Those are directed presentations of higher-dimensional categories, generalising word and term rewriting systems.

In the recent years, the algebraic structure of polygraph has led to a new theory of rewriting, called *higher-dimensional rewriting*, as a unifying point of view for usual rewriting paradigms, namely abstract, word and term rewriting [73], [79], [60], [61], and beyond: Petri nets [63] and formal proofs of classical and linear logic have been expressed in this framework [62]. Higher-dimensional rewriting has developed its own methods to analyse computational properties of polygraphs, using in particular algebraic tools such as derivations to prove termination, which in turn led to new tools for complexity analysis [46].

### 3.5.3. Squier theory

The homotopical properties of higher categories, as studied in mathematics, are in fact deeply related to the computational properties of their polygraphic presentations. This connection has its roots in a tradition of using rewriting-like methods in algebra, and more specifically in the work of Anick [41] and Squier [85], [84] in the 1980s: Squier has proved that, if a monoid  $M$  can be presented by a *finite*, *terminating* and *confluent* rewriting system, then its third integral homology group  $H_3(M, \mathbb{Z})$  is finitely generated and the monoid  $M$  has *finite derivation type* (a property of homotopical nature). This allowed him to conclude that finite convergent rewriting systems were not a universal solution to decide the word problem of finitely generated monoids. Since then, Yves Guiraud and Philippe Malbos have shown that this connection was part of a deeper unified theory when formulated in the higher-dimensional setting [9], [10], [66], [67], [68].

In particular, the computational content of Squier’s proof has led to a constructive methodology to produce, from a convergent presentation, *coherent presentations* and *polygraphic resolutions* of algebraic structures, such as monoids [9] and algebras [65]. A coherent presentation of a monoid  $M$  is a 3-dimensional combinatorial object that contains not only a presentation of  $M$  (generators and relations), but also higher-dimensional cells, each of which corresponding to two fundamentally different proofs of the same equality: this is, in essence, the same as the proofs of equality of proofs of equality in homotopy type theory. When this process of “unfolding” proofs of equalities is pursued in every dimension, one gets a polygraphic resolution of the starting monoid  $M$ . This object has the following desirable qualities: it is free and homotopically equivalent to  $M$  (in the canonical model structure of higher categories [74], [42]). A polygraphic resolution of an algebraic object  $X$  is a faithful formalisation of  $X$  on which one can perform computations, such as homotopical

or homological invariants of  $X$ . In particular, this has led to new algorithms and proofs in representation theory [7], and in homological algebra [64], [65].

## SUMO Project-Team

### 3. Research Program

#### 3.1. Analysis and verification of quantitative systems

The overall objective of this axis is to develop the quantitative aspects of formal methods while maintaining the tractability of verification objectives and progressing toward the management of large systems. This covers the development of relevant modeling formalisms, to nicely weave time, costs and probabilities with existing models for concurrency. We plan to further study time(d) Petri nets, networks of timed automata (with synchronous or asynchronous communications), stochastic automata, partially observed Markov decision processes, etc. A second objective is to develop verification methods for such quantitative systems. This covers several aspects: quantitative verification questions (compute an optimal scheduling policy), boolean questions on quantitative features (deciding whether some probability is greater than a threshold), robustness issues (will a system have the same behaviors if some parameter is slightly altered), etc. Our goal is to explore the frontier between decidable and undecidable problems, or more pragmatically tractable and untractable problems. Of course, there is a tradeoff between the expressivity and the tractability of a model. Models that incorporate distributed aspects, probabilities, time, etc, are typically untractable. In such a case, abstraction or approximation techniques are a work around that we will explore.

Here are some more detailed topics that we place in our agenda

- analysis of diagnosability and opacity properties for stochastic systems
- verification of time(d) Petri nets
- robustness analysis for timed or/and stochastic systems
- abstraction techniques for quantitative systems

#### 3.2. Control of quantitative systems

The main objective of this research axis is to explore the quantitative and/or distributed extensions of classical control problems. We envision control in its widest meaning of driving a system in order to guarantee or enforce some extra property (i.e. not guaranteed by the system alone), in a partially or totally observed setting. This property can either be logical (e.g. reachability or safety) or quantitative (e.g. reach some performance level). These problems have of course an offline facet (e.g. controller design, existence of a policy/strategy) and an online facet (e.g. algorithm to select some optimal action at runtime).

Our objectives comprise classical controller synthesis for discrete event systems, with extensions to temporal/stochastic/reward settings. They also cover maintaining or maximizing extra properties as diagnosability or opacity, for example in stochastic systems. We also target further analysis of POMDPs (partially observed Markov decision processes), and multi-agent versions of policy synthesis relying on tools from game theory. We aim at addressing some control problems motivated by industrial applications, that raise issues like the optimal control of timed and stochastic discrete event systems, with concerns like robustness to perturbations and multicriteria optimization. Finally, we also plan to work on modular testing, and on runtime enforcement techniques, in order to guarantee extra logical and temporal properties to event flows.

#### 3.3. Management of large or distributed systems

The generic terms of “supervision” or “management” of distributed systems cover problems like control, diagnosis, sensor placement, planning, optimization, (state) estimation, parameter identification, testing, etc. This research axis examines how classical settings for such problems can scale up to large or distributed systems. Our work will be driven by considerations like : how to take advantage of modularity, how to design approximate management algorithms, how to design relevant abstractions to make large systems more tractable, how to deal with models of unknown size, how to design mechanisms to obtain relevant models, etc.



As more specific objectives, let us mention:

- Parametric systems. How to verify properties of distributed systems with an unknown number of components.
- Approximate management methods. We will explore the extension of ideas developed for Bayesian inference in large scale stochastic systems (such as turbo-algorithms for example) to the field of modular dynamic systems. When component interactions are sparse, even if exact management methods are unaccessible (for diagnosis, planning, control, etc.), good approximations based on local computations may be accessible.
- Model abstraction. We will explore techniques to design more tractable abstractions of stochastic dynamic systems defined on large sets of variables.
- Self-modeling, which consists in managing large scale systems that are known by their building rules, but which specific managed instance is only discovered at runtime, and on the fly. The model of the managed system is built on-line, following the needs of the management algorithms.
- Distributed control. We will tackle issues related to asynchronous communications between local controllers, and to abstraction techniques allowing to address large systems.
- Test and enforcement. We will tackle coverage issues for the test of large systems, and the test and enforcement of properties for timed models, or for systems handling data.

### 3.4. Data driven systems

Data-driven systems are systems whose behavior depends both on explicit workflows (scheduling and durations of tasks, calls to possibly distant services,...) and on the data processed by the system (stored data, parameters of a request, results of a request,...). This family of systems covers workflows that convey data (business processes or information systems), transactional systems (web stores), large databases managed with rules (banking systems), collaborative environments (crowds, health systems), etc. These systems are distributed, modular, and open: they integrate components and sub-services distributed over the web and accept requests from clients. Our objective is to provide validation and supervision tools for such systems. To achieve this goal, we have to solve several challenging tasks:

- provide realistic models, and sound automated abstraction techniques, to reason on models that are reasonable abstractions of real systems. These models should be able to encompass modularity, distribution, in a context where workflows and data aspects are tightly connected.
- address design of data driven systems in a declarative way: declarative models are another way to handle data-driven systems. Rather than defining the explicit workflows and their effects on data, rule-based models state how actions are enacted in terms of the shape (pattern matching) or value of the current data. We think that distributed rewriting rules or attributed grammars can provide a practical yet formal framework for maintenance, by providing a solution to update mandatory documentation during the lifetime of an artifact.
- provide tractable solutions for validation of models. Frequent issues are safety questions (can a system reach some bad configuration?), but also liveness (workflows progress), ... These questions should not only remain decidable on our models, but also with efficient computational methods.
- address QoS management in large reconfigurable systems: Data driven distributed systems often have constraints in terms of QoS. This QoS questions adrese performance issues, but also data quality. This calls for an analysis of quantitative features and for reconfiguration techniques to meet desired QoS.

## TOCCATA Project-Team

### 3. Research Program

#### 3.1. Introduction

In the former ProVal project, we have been working on the design of methods and tools for deductive verification of programs. One of our original skills was the ability to conduct proofs by using automatic provers and proof assistants at the same time, depending on the difficulty of the program, and specifically the difficulty of each particular verification condition. We thus believe that we are in a good position to propose a bridge between the two families of approaches of deductive verification presented above. Establishing this bridge is one of the goals of the Toccata project: we want to provide methods and tools for deductive program verification that can offer both a high amount of proof automation and a high guarantee of validity. Toward this objective, a new axis of research was proposed: the development of *certified* analysis tools that are themselves formally proved correct.

The reader should be aware that the word “certified” in this scientific programme means “verified by a formal specification and a formal proof that the program meets this specification”. This differs from the standard meaning of “certified” in an industrial context where it means a conformance to some rigorous process and/or norm. We believe this is the right term to use, as it was used for the *Certified Compiler* project [100], the new conference series *Certified Programs and Proofs*, and more generally the important topics of *proof certificates*.

In industrial applications, numerical calculations are very common (e.g. control software in transportation). Typically they involve floating-point numbers. Some of the members of Toccata have an internationally recognized expertise on deductive program verification involving floating-point computations. Our past work includes a new approach for proving behavioral properties of numerical C programs using Frama-C/Jessie [41], various examples of applications of that approach [61], the use of the Gappa solver for proving numerical algorithms [118], an approach to take architectures and compilers into account when dealing with floating-point programs [62], [111]. We also contributed to the Handbook of Floating-Point Arithmetic [110]. A representative case study is the analysis and the proof of both the method error and the rounding error of a numerical analysis program solving the one-dimension acoustic wave equation [4] [54]. Our experience led us to a conclusion that verification of numerical programs can benefit a lot from combining automatic and interactive theorem proving [56], [61]. Certification of numerical programs is the other main axis of Toccata.

Our scientific programme is structured into four objectives:

1. deductive program verification;
2. automated reasoning;
3. formalization and certification of languages, tools and systems;
4. proof of numerical programs.

We detail these objectives below.

#### 3.2. Deductive Program Verification

Permanent researchers: A. Charguéraud, S. Conchon, J.-C. Filliâtre, C. Marché, G. Melquiond, A. Paskevich

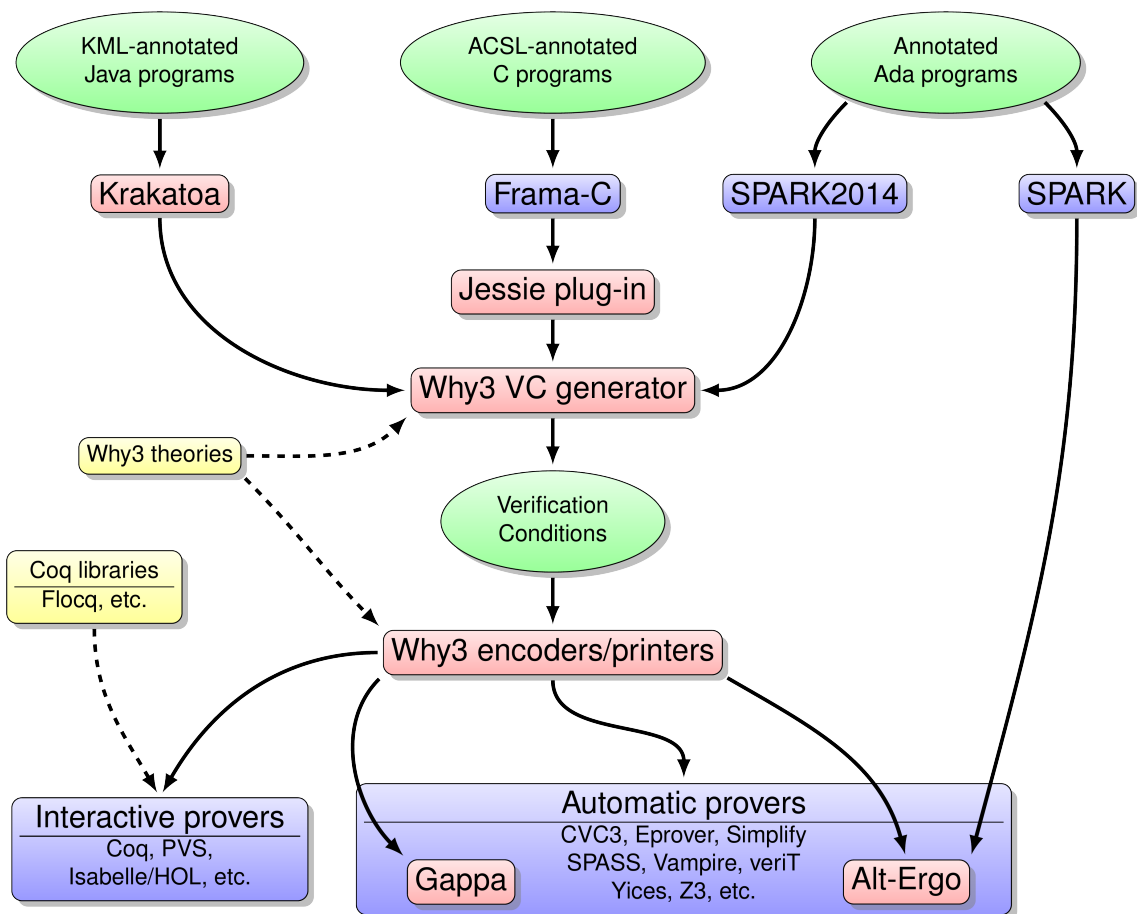


Figure 1. The Why3 ecosystem

### 3.2.1. The Why3 Ecosystem

This ecosystem is central in our work; it is displayed on Figure 1. The boxes in red background correspond to the tools we develop in the Toccata team.

- The initial design of Why3 was presented in 2012 [49], [89]. In the past years, the main improvements concern the specification language (such as support for higher-order logic functions [67]) and the support for provers. Several new interactive provers are now supported: PVS 6 (used at NASA), Isabelle2014 (planned to be used in the context of Ada program via Spark), and Mathematica. We also added support for new automated provers: CVC4, Metitarski, Metis, Beagle, Princess, and Yices2. More technical improvements are the design of a Coq tactic to call provers via Why3 from Coq, and the design of a proof session mechanism [48]. Why3 was presented during several invited talks [88], [87], [84], [85].
- At the level of the C front-end of Why3 (via Frama-C), we have proposed an approach to add a notion of refinement on C programs [117], and an approach to reason about pointer programs with a standard logic, via *separation predicates* [47]
- The Ada front-end of Why3 has mainly been developed during the past three years, leading to the release of SPARK2014 [95] (<http://www.spark-2014.org/>)
- In collaboration with J. Almeida, M. Barbosa, J. Pinto, and B. Vieira (University do Minho, Braga, Portugal), J.-C. Filliâtre has developed a method for certifying programs involving cryptographic methods. It uses Why as an intermediate language [40].
- With M. Pereira and S. Melo de Sousa (Universidade da Beira Interior, Covilhã, Portugal), J.-C. Filliâtre has developed an environment for proving ARM assembly code. It uses Why3 as an intermediate VC generator. It was presented at the Inforum conference [114] (best student paper).

### 3.2.2. Concurrent Programming

- S. Conchon and A. Mebsout, in collaboration with F. Zaïdi (VALS team, LRI), A. Goel and S. Krstić (Strategic Cad Labs, INTEL) have proposed a new model-checking approach for verifying safety properties of array-based systems. This is a syntactically restricted class of parametrized transition systems with states represented as arrays indexed by an arbitrary number of processes. Cache coherence protocols and mutual exclusion algorithms are typical examples of such systems. It was first presented at CAV 2012 [7] and detailed further [77]. It was applied to the verification of programs with fences [73]. The core algorithm has been extended with a mechanism for inferring invariants. This new algorithm, called BRAB, is able to automatically infer invariants strong enough to prove industrial cache coherence protocols. BRAB computes over-approximations of backward reachable states that are checked to be unreachable in a finite instance of the system. These approximations (candidate invariants) are then model-checked together with the original safety properties. Completeness of the approach is ensured by a mechanism for backtracking on spurious traces introduced by too coarse approximations [74], [106].
- In the context of the ERC DeepSea project <sup>0</sup>, A. Charguéraud and his co-authors have developed a unifying semantics for various different paradigms of parallel computing (fork-join, async-finish, and futures), and published a conference paper describing this work [16]. Besides, A. Charguéraud and his co-authors have polished their previous work on granularity control for parallel algorithms using user-provided complexity functions, and produced a journal article [11].

### 3.2.3. Case Studies

- To provide an easy access to the case studies that we develop using Why3 and its front-ends, we have published a *gallery of verified programs* on our web page <http://toccata.lri.fr/gallery/>. Part of these examples are the solutions to the competitions VerifyThis 2011 [63], VerifyThis 2012 [3], and the competition VScomp 2011 [90].

<sup>0</sup>Arthur Charguéraud is involved 40% of his time in the ERC DeepSea project, which is hosted at Inria Paris Rocquencourt (team Gallium).

- Other case studies that led to publications are the design of a library of data-structures based on AVLs [66], and the verification a two-lines C program (solving the  $N$ -queens puzzle) using Why3 [86].
- A. Charguéraud, with F. Pottier (Inria Paris), extended their formalization of the correctness and asymptotic complexity of the classic Union Find data structure, which features the bound expressed in terms of the inverse Ackermann function [39]. The proof, conducted using CFML extended with time credits, was refined using a slightly more complex potential function, allowing to derive a simpler and richer interface for the data structure. A journal article is in preparation.

For other case studies, see also sections of numerical programs and formalization of languages and tools.

### 3.2.4. Project-team Positioning

Several research groups in the world develop their own approaches, techniques, and tools for deductive verification. With respect to all these related approaches and tools, our originality is our will to use more sophisticated specification languages (with inductive definitions, higher-order features and such) and the ability to use a large set of various theorem provers, including the use of interactive theorem proving to deal with complex functional properties.

- The RiSE team <sup>0</sup> at Microsoft Research Redmond, USA, partly in collaboration with team “programming methodology” team <sup>0</sup> at ETH Zurich develop tools that are closely related to ours: Boogie and Dafny are direct competitors of Why3, VCC is a direct competitor of Frama-C/Jessie.
- The KeY project <sup>0</sup> (several teams, mainly at Karlsruhe and Darmstadt, Germany, and Göteborg, Sweden) develops the KeY tool for Java program verification [38], based on dynamic logic, and has several industrial users. They use a specific modal logic (dynamic logic) for modeling programs, whereas we use standard logic, so as to be able to use off-the-shelf automated provers.
- The “software engineering” group at Augsburg, Germany, develops the KIV system <sup>0</sup>, which was created more than 20 years ago (1992) and is still well maintained and efficient. It provides a semi-interactive proof environment based on algebraic-style specifications, and is able to deal with several kinds of imperative style programs. They have a significant industrial impact.
- The VeriFast system <sup>0</sup> aims at verifying C programs specified in Separation Logic. It is developed at the Catholic University at Leuven, Belgium. We do not usually use separation logic (so as to use off-the-shelf provers) but alternative approaches (e.g. static memory separation analysis).
- The Mobius Program Verification Environment <sup>0</sup> is a joint effort for the verification of Java source annotated with JML, combining static analysis and runtime checking. The tool ESC/Java2 <sup>0</sup> is a VC generator similar to Krakatoa, that builds on top of Boogie. It is developed by a community led by University of Copenhagen, Denmark. Again, our specificity with respect to them is the consideration of more complex specification languages and interactive theorem proving.
- The Lab for Automated Reasoning and Analysis <sup>0</sup> at EPFL, develop methods and tools for verification of Java (Jahob) and Scala (Leon) programs. They share with us the will and the ability to use several provers at the same time.
- The TLA environment <sup>0</sup>, developed by Microsoft Research and the Inria team Veridis, aims at the verification of concurrent programs using mathematical specifications, model checking, and interactive or automated theorem proving.
- The F\* project <sup>0</sup>, developed by Microsoft Research and the Inria Prosecco team, aims at providing a rich environment for developing programs and proving them.

<sup>0</sup><http://research.microsoft.com/en-us/groups/rise/default.aspx>

<sup>0</sup><http://www.pm.inf.ethz.ch/>

<sup>0</sup><http://www.key-project.org/>

<sup>0</sup><http://www.isse.uni-augsburg.de/en/software/kiv/>

<sup>0</sup><http://people.cs.kuleuven.be/~bart.jacobs/verifast/>

<sup>0</sup><http://kindsoftware.com/products/opensource/Mobius/>

<sup>0</sup><http://kindsoftware.com/products/opensource/ESCJava2/>

<sup>0</sup><http://lara.epfl.ch/w/>

<sup>0</sup><http://research.microsoft.com/en-us/um/people/lamport/TLA/TLA.html>

<sup>0</sup><http://research.microsoft.com/en-us/projects/fstar/>

The KeY and KIV environments mentioned above are partly based on interactive theorem provers. There are other approaches on top of general-purpose proof assistants for proving programs that are not purely functional:

- The Ynot project<sup>0</sup> is a Coq library for writing imperative programs specified in separation logic. It was developed at Harvard University, until the end of the project in 2010. Ynot had similar goals as CFML, although Ynot requires programs to be written in monadic style inside Coq, whereas CFML applies directly on programs written in OCaml syntax, translating them into logical formulae.
- Front-ends to Isabelle were developed to deal with simple sequential imperative programs [116] or C programs [113]. The L4-verified project [96] is built on top of Isabelle.

### 3.3. Automated Reasoning

Permanent researchers: S. Conchon, G. Melquiond, A. Paskevich

#### 3.3.1. Generalities on Automated Reasoning

- J. C. Blanchette and A. Paskevich have designed an extension to the TPTP TFF (Typed First-order Form) format of theorem proving problems to support rank-1 polymorphic types (also known as ML-style parametric polymorphism) [1]. This extension, named TFF1, has been incorporated in the TPTP standard.
- S. Conchon defended his *habilitation à diriger des recherches* in December 2012. The memoir [70] provides a useful survey of the scientific work of the past 10 years, around the SMT solving techniques, that led to the tools Alt-Ergo and Cubicle as they are nowadays.

#### 3.3.2. Quantifiers and Triggers

- C. Dross, J. Kanig, S. Conchon, and A. Paskevich have proposed a generic framework for adding a decision procedure for a theory or a combination of theories to an SMT prover. This mechanism is based on the notion of instantiation patterns, or *triggers*, which restrict instantiation of universal premises and can effectively prevent a combinatorial explosion. A user provides an axiomatization with triggers, along with a proof of completeness and termination in the proposed framework, and obtains in return a sound, complete and terminating solver for his theory. A prototype implementation was realized on top of Alt-Ergo. As a case study, a feature-rich axiomatization of doubly-linked lists was proved complete and terminating [82]. C. Dross defended her PhD thesis in April 2014 [83]. The main results of the thesis are: (1) a formal semantics of the notion of *triggers* typically used to control quantifier instantiation in SMT solvers, (2) a general setting to show how a first-order axiomatization with triggers can be proved correct, complete, and terminating, and (3) an extended DPLL(T) algorithm to integrate a first-order axiomatization with triggers as a decision procedure for the theory it defines. Significant case studies were conducted on examples coming from SPARK programs, and on the benchmarks on B set theory constructed within the BWare project.

#### 3.3.3. Reasoning Modulo Theories

- S. Conchon, É. Contejean and M. Iguernelala have presented a modular extension of ground AC-completion for deciding formulas in the combination of the theory of equality with user-defined AC symbols, uninterpreted symbols and an arbitrary signature-disjoint Shostak theory X [72]. This work extends the results presented in [71] by showing that a simple preprocessing step allows to get rid of a full AC-compatible reduction ordering, and to simply use a partial multiset extension of a *non-necessarily AC-compatible* ordering.
- S. Conchon, M. Iguernelala, and A. Mebsout have designed a collaborative framework for reasoning modulo simple properties of non-linear arithmetic [76]. This framework has been implemented in the Alt-Ergo SMT solver.

---

<sup>0</sup><http://ynot.cs.harvard.edu/>

- S. Conchon, G. Melquiond and C. Roux have described a dedicated procedure for a theory of floating-point numbers which allows reasoning on approximation errors. This procedure is based on the approach of the Gappa tool: it performs saturation of consequences of the axioms, in order to refine bounds on expressions. In addition to the original approach, bounds are further refined by a constraint solver for linear arithmetic [78]. This procedure has been implemented in Alt-Ergo.
- In collaboration with A. Mahboubi (Inria project-team Typical), and G. Melquiond, the group involved in the development of Alt-Ergo have implemented and proved the correctness of a novel decision procedure for quantifier-free linear integer arithmetic [2]. This algorithm tries to bridge the gap between projection and branching/cutting methods: it interleaves an exhaustive search for a model with bounds inference. These bounds are computed provided an oracle capable of finding constant positive linear combinations of affine forms. An efficient oracle based on the Simplex procedure has been designed. This algorithm is proved sound, complete, and terminating and is implemented in Alt-Ergo.
- Most of the results above are detailed in M. Iguernelala's PhD thesis [93].

### 3.3.4. Applications

- We have been quite successful in the application of Alt-Ergo to industrial development: qualification by Airbus France, integration of Alt-Ergo into the Spark Pro toolset.
- In the context of the BWare project, aiming at using Why3 and Alt-Ergo for discharging proof obligations generated by Atelier B, we made progress into several directions. The method of translation of B proof obligations into Why3 goals was first presented at ABZ'2012 [109]. Then, new drivers have been designed for Why3, in order to use new back-end provers Zenon modulo and iProver modulo. A notion of rewrite rule was introduced into Why3, and a transformation for simplifying goals before sending them to back-end provers was designed. Intermediate results obtained so far in the project were presented both at the French conference AFADL [81] and at ABZ'2014 [80].

On the side of Alt-Ergo, recent developments have been made to efficiently discharge proof obligations generated by Atelier B. This includes a new plugin architecture to facilitate experiments with different SAT engines, new heuristics to handle quantified formulas, and important modifications in its internal data structures to boost performances of core decision procedures. Benchmarks realized on more than 10,000 proof obligations generated from industrial B projects show significant improvements [75].

- Hybrid automata interleave continuous behaviors (described by differential equations) with discrete transitions. D. Ishii and G. Melquiond have worked on an automated procedure for verifying safety properties (that is, global invariants) of such systems [94].

### 3.3.5. Project-team Positioning

Automated Theorem Proving is a large community, but several sub-groups can be identified:

- The SMT-LIB community gathers people interested in reasoning modulo theories. In this community, only a minority of participants are interested in supporting first-order quantifiers at the same time as theories. SMT solvers that support quantifiers are Z3 (Microsoft Research Redmond, USA), CVC3 and its successor CVC4<sup>0</sup>.
- The TPTP community gathers people interested in first-order theorem proving.
- Other Inria teams develop provers: veriT by team Veridis, and Psyche by team Parsifal.
- Other groups develop provers dedicated to very specific cases, such as Metitarski<sup>0</sup> at Cambridge, UK, which aims at proving formulas on real numbers, in particular involving special functions such as log or exp. The goal is somewhat similar to our CoqInterval library, cf objective 4.

<sup>0</sup><http://cvc4.cs.nyu.edu/web/>

<sup>0</sup><http://www.cl.cam.ac.uk/~lp15/papers/Arith/>

It should be noticed that a large number of provers mentioned above are connected to Why3 as back-ends.

### 3.4. Formalization and Certification of Languages, Tools and Systems

Permanent researchers: S. Boldo, A. Charguéraud, C. Marché, G. Melquiond, C. Paulin

#### 3.4.1. Real Numbers, Real Analysis, Probabilities

- S. Boldo, C. Lelay, and G. Melquiond have worked on the Coquelicot library, designed to be a user-friendly Coq library about real analysis [59], [60]. An easier way of writing formulas and theorem statements is achieved by relying on total functions in place of dependent types for limits, derivatives, integrals, power series, and so on. To help with the proof process, the library comes with a comprehensive set of theorems and some automation. We have exercised the library on several use cases: on an exam at university entry level [98], for the definitions and properties of Bessel functions [97], and for the solution of the one-dimensional wave equation [99]. We have also conducted a survey on the formalization of real arithmetic and real analysis in various proof systems [12].
- Watermarking techniques are used to help identify copies of publicly released information. They consist in applying a slight and secret modification to the data before its release, in a way that should remain recognizable even in (reasonably) modified copies of the data. Using the Coq ALEA library, which formalizes probability theory and probabilistic programs, D. Baelde together with P. Courtieu, D. Gross-Amblard from Rennes and C. Paulin have established new results about the robustness of watermarking schemes against arbitrary attackers [42]. The technique for proving robustness is adapted from methods commonly used for cryptographic protocols and our work illustrates the strengths and particularities of the ALEA style of reasoning about probabilistic programs.

#### 3.4.2. Formalization of Languages, Semantics

- P. Herms, together with C. Marché and B. Monate (CEA List), has developed a certified VC generator, using Coq. The program for VC calculus and its specifications are both written in Coq, but the code is crafted so that it can be extracted automatically into a stand-alone executable. It is also designed in a way that allows the use of arbitrary first-order theorem provers to discharge the generated obligations [92]. On top of this generic VC generator, P. Herms developed a certified VC generator for C source code annotated using ACSL. This work is the main result of his PhD thesis [91].
- A. Tafat and C. Marché have developed a certified VC generator using Why3 [102], [103]. The challenge was to formalize the operational semantics of an imperative language, and a corresponding weakest precondition calculus, without the possibility to use Coq advanced features such as dependent types or higher-order functions. The classical issues with local bindings, names and substitutions were solved by identifying appropriate lemmas. It was shown that Why3 can offer a significantly higher amount of proof automation compared to Coq.
- A. Charguéraud, together with Alan Schmitt (Inria Rennes) and Thomas Wood (Imperial College), has developed an interactive debugger for JavaScript. The interface, accessible as a webpage in a browser, allows to execute a given JavaScript program, following step by step the formal specification of JavaScript developed in prior work on *JsCert* [50]. Concretely, the tool acts as a double-debugger: one can visualize both the state of the interpreted program and the state of the interpreter program. This tool is intended for the JavaScript committee, VM developers, and other experts in JavaScript semantics.
- M. Clochard, C. Marché, and A. Paskevich have developed a general setting for developing programs involving binders, using Why3. This approach was successfully validated on two case studies: a verified implementation of untyped lambda-calculus and a verified tableaux-based theorem prover [69].



- M. Clochard, J.-C. Filliâtre, C. Marché, and A. Paskevich have developed a case study on the formalization of semantics of programming languages using Why3 [67]. This case study aims at illustrating recent improvements of Why3 regarding the support for higher-order logic features in the input logic of Why3, and how these are encoded into first-order logic, so that goals can be discharged by automated provers. This case study also illustrates how reasoning by induction can be done without need for interactive proofs, via the use of *lemma functions*.
- M. Clochard and L. Gondelman have developed a formalization of a simple compiler in Why3 [68]. It compiles a simple imperative language into assembler instructions for a stack machine. This case study was inspired by a similar example developed using Coq and interactive theorem proving. The aim is to improve significantly the degree of automation in the proofs. This is achieved by the formalization of a Hoare logic and a Weakest Precondition Calculus on assembly programs, so that the correctness of compilation is seen as a formal specification of the assembly instructions generated.

### 3.4.3. Project-team Positioning

The objective of formalizing languages and algorithms is very general, and it is pursued by several Inria teams. One common trait is the use of the Coq proof assistant for this purpose: Pi.r2 (development of Coq itself and its meta-theory), Gallium (semantics and compilers of programming languages), Marelle (formalization of mathematics), SpecFun (real arithmetic), Celtique (formalization of static analyzers).

Other environments for the formalization of languages include

- ACL2 system <sup>0</sup>: an environment for writing programs with formal specifications in first-order logic based on a Lisp engine. The proofs are conducted using a prover based on the Boyer-Moore approach. It is a rather old system but still actively maintained and powerful, developed at University of Texas at Austin. It has a strong industrial impact.
- Isabelle environment <sup>0</sup>: both a proof assistant and an environment for developing pure applicative programs. It is developed jointly at University of Cambridge, UK, Technische Universität München, Germany, and to some extent by the VALS team at LRI, Université Paris-Sud. It features highly automated tactics based on ATP systems (the Sledgehammer tool).
- The team “Trustworthy Systems” at NICTA in Australia <sup>0</sup> aims at developing highly trustable software applications. They developed a formally verified micro-kernel called seL4 [96], using a home-made layer to deal with C programs on top of the Isabelle prover.
- The PVS system <sup>0</sup> is an environment for both programming and proving (purely applicative) programs. It is developed at the Computer Science Laboratory of SRI international, California, USA. A major user of PVS is the team LFM <sup>0</sup> at NASA Langley, USA, for the certification of programs related to air traffic control.

In the Toccata team, we do not see these alternative environments as competitors, even though, for historical reasons, we are mainly using Coq. Indeed both Isabelle and PVS are available as back-ends of Why3.

## 3.5. Proof of Numerical Programs

Permanent researchers: S. Boldo, C. Marché, G. Melquiond

- Linked with objective 1 (Deductive Program Verification), the methodology for proving numerical C programs has been presented by S. Boldo in her habilitation [52] and as invited speaker [53]. An application is the formal verification of a numerical analysis program. S. Boldo, J.-C. Filliâtre, and G. Melquiond, with F. Clément and P. Weis (POMDAPI team, Inria Paris - Rocquencourt), and M. Mayero (LIPN), completed the formal proof of the second-order centered finite-difference scheme for the one-dimensional acoustic wave [55][4].

<sup>0</sup><http://www.cs.utexas.edu/~moore/acl2/>

<sup>0</sup><http://isabelle.in.tum.de/>

<sup>0</sup><http://ssrg.nicta.com.au/projects/TS/>

<sup>0</sup><http://pvs.csl.sri.com/>

<sup>0</sup><http://shemesh.larc.nasa.gov/fm/fm-main-team.html>

- Several challenging floating-point algorithms have been studied and proved. This includes an algorithm by Kahan for computing the area of a triangle: S. Boldo proved an improvement of its error bound and new investigations in case of underflow [51]. This includes investigations about quaternions. They should be of norm 1, but due to the round-off errors, a drift of this norm is observed over time. C. Marché determined a bound on this drift and formally proved it correct [9]. P. Roux formally verified an algorithm for checking that a matrix is semi-definite positive [115]. The challenge here is that testing semi-definiteness involves algebraic number computations, yet it needs to be implemented using only approximate floating-point operations.
- Because of compiler optimizations (or bugs), the floating-point semantics of a program might change once compiled, thus invalidating any property proved on the source code. We have investigated two ways to circumvent this issue, depending on whether the compiler is a black box. When it is, T. Nguyen has proposed to analyze the assembly code it generates and to verify it is correct [112]. On the contrary, S. Boldo and G. Melquiond (in collaboration with J.-H. Jourdan and X. Leroy) have added support for floating-point arithmetic to the CompCert compiler and formally proved that none of the transformations the compiler applies modify the floating-point semantics of the program [58], [57].
- Linked with objectives 2 (Automated Reasoning) and 3 (Formalization and Certification of Languages, Tools and Systems), G. Melquiond has implemented an efficient Coq library for floating-point arithmetic and proved its correctness in terms of operations on real numbers [107]. It serves as a basis for an interval arithmetic on which Taylor models have been formalized. É. Martin-Dorel and G. Melquiond have integrated these models into CoqInterval [15]. This Coq library is dedicated to automatically proving the approximation properties that occur when formally verifying the implementation of mathematical libraries (libm).
- Double rounding occurs when the target precision of a floating-point computation is narrower than the working precision. In some situations, this phenomenon incurs a loss of accuracy. P. Roux has formally studied when it is innocuous for basic arithmetic operations [115]. É. Martin-Dorel and G. Melquiond (in collaboration with J.-M. Muller) have formally studied how it impacts algorithms used for error-free transformations [105]. These works were based on the Flocq formalization of floating-point arithmetic for Coq.
- By combining multi-precision arithmetic, interval arithmetic, and massively-parallel computations, G. Melquiond (in collaboration with G. Nowak and P. Zimmermann) has computed enough digits of the Mersenne-Graham constant to invalidate a 30-year old conjecture about its closed form [108].

### 3.5.1. Project-team Positioning

This objective deals both with formal verification and floating-point arithmetic, which is quite uncommon. Therefore our competitors/peers are few. We may only cite the works by J. Duracz and M. Konečný, Aston University in Birmingham, UK.

The Inria team AriC (Grenoble - Rhône-Alpes) is closer to our research interests, but they are lacking manpower on the formal proof side; we have numerous collaborations with them. The Inria team Caramel (Nancy - Grand Est) also shares some research interests with us, though fewer; again, they do not work on the formal aspect of the verification; we have some occasional collaborations with them.

There are many formalization efforts from chip manufacturers, such as AMD (using the ACL2 proof assistant) and Intel (using the Forte proof assistants) but the algorithms they consider are quite different from the ones we study. The works on the topic of floating-point arithmetic from J. Harrison at Intel using HOL Light are really close to our research interests, but they seem to be discontinued.

A few deductive program verification teams are willing to extend their tools toward floating-point programs. This includes the KeY project and SPARK. We have an ongoing collaboration with the latter, in the context of the ProofInUse project.

Deductive verification is not the only way to prove programs. Abstract interpretation is widely used, and several teams are interested in floating-point arithmetic. This includes the Inria team Antique (Paris - Rocquencourt) and a CEA List team, who have respectively developed the Astrée and Fluctuat tools. This approach targets a different class of numerical algorithms than the ones we are interested in.

Other people, especially from the SMT community (*cf* objective 2), are also interested in automatically proving formulas about floating-point numbers, notably at Oxford University. They are mainly focusing on pure floating-point arithmetic though and do not consider them as approximation of real numbers.

Finally, it can be noted that numerous teams are working on the verification of numerical programs, but assuming the computations are real rather than floating-point ones. This is out of the scope of this objective.

## VERIDIS Project-Team

### 3. Research Program

#### 3.1. Automated and Interactive Theorem Proving

The VeriDis team gathers experts in techniques and tools for automatic deduction and interactive theorem proving, and specialists in methods and formalisms designed for the development of trustworthy concurrent and distributed systems and algorithms. Our common objective is twofold: first, we wish to advance the state of the art in automated and interactive theorem proving, and their combinations. Second, we work on making the resulting technology available for the computer-aided verification of distributed systems and protocols. In particular, our techniques and tools are intended to support sound methods for the development of trustworthy distributed systems that scale to algorithms relevant for practical applications.

VeriDis members from Saarbrücken are developing SPASS [10], one of the leading automated theorem provers for first-order logic based on the superposition calculus [39]. The group also studies general frameworks for the combination of theories such as the locality principle [52] and automated reasoning mechanisms these induce.

In a complementary approach to automated deduction, VeriDis members from Nancy work on techniques for integrating reasoners for specific theories. They develop veriT [1], an SMT (Satisfiability Modulo Theories [41]) solver that combines decision procedures for different fragments of first-order logic and that integrates an automatic theorem prover for full first-order logic. The veriT solver is designed to produce detailed proofs; this makes it particularly suitable as a component of a robust cooperation of deduction tools.

Finally, VeriDis members design effective quantifier elimination methods and decision procedures for algebraic theories, supported by their efficient implementation in the Redlog system [4].

An important objective of this line of work is the integration of theories in automated deduction. Typical theories of interest, including fragments of arithmetic, are not expressible in first-order logic. We therefore explore efficient, modular techniques for integrating semantic and syntactic reasoning methods, develop novel combination results and techniques for quantifier instantiation. These problems are addressed from both sides, e.g. by embedding a decision procedure into the superposition framework or by allowing an SMT solver to accept axiomatizations for plug-in theories. We also develop specific decision procedures for theories such as non-linear real arithmetic that are important when reasoning about certain classes of (e.g., real-time) systems but that also have interesting applications beyond verification.

We rely on interactive theorem provers for reasoning about specifications at a high level of abstraction when fully automatic verification is not (yet) feasible. An interactive proof platform should help verification engineers lay out the proof structure at a sufficiently high level of abstraction; powerful automatic plug-ins should then discharge the resulting proof steps. Members of VeriDis have ample experience in the specification and subsequent machine-assisted, interactive verification of algorithms. In particular, we participate in a project at the joint Microsoft Research-Inria Centre in Saclay on the development of methods and tools for the formal proof of TLA<sup>+</sup> [45] specifications. Our prover relies on a declarative proof language, and calls upon several automatic backends [3]. Trust in the correctness of the overall proof can be ensured when the backends provide justifications that can be checked by the trusted kernel of a proof assistant. During the development of a proof, most obligations that are passed to the prover actually fail – for example, because necessary information is not present in the context or because the invariant is too weak, and we are interested in explaining failed proof attempts to the user, in particular through the construction of counter-models.

### 3.2. Formal Methods for Developing and Analyzing Algorithms and Systems

Theorem provers are not used in isolation, but they support the application of sound methodologies for modeling and verifying systems. In this respect, members of VeriDis have gained expertise and recognition in making contributions to formal methods for concurrent and distributed algorithms and systems [2], [9], and in applying them to concrete use cases. In particular, the concept of *refinement* [38], [40], [48] in state-based modeling formalisms is central to our approach because it allows us to present a rational (re)construction of system development. An important goal in designing such methods is to establish precise proof obligations many of which can be discharged by automatic tools. This requires taking into account specific characteristics of certain classes of systems and tailoring the model to concrete computational models. Our research in this area is supported by carrying out case studies for academic and industrial developments. This activity benefits from and influences the development of our proof tools.

In this line of work, we investigate specific development and verification patterns for particular classes of algorithms, in order to reduce the work associated with their verification. We are also interested in applications of formal methods and their associated tools to the development of systems that underlie specific certification requirements in the sense of, e.g., Common Criteria. Finally, we are interested in the adaptation of model checking techniques for verifying actual distributed programs, rather than high-level models.

Today, the formal verification of a new algorithm is typically the subject of a PhD thesis, if it is addressed at all. This situation is not sustainable given the move towards more and more parallelism in mainstream systems: algorithm developers and system designers must be able to productively use verification tools for validating their algorithms and implementations. On a high level, the goal of VeriDis is to make formal verification standard practice for the development of distributed algorithms and systems, just as symbolic model checking has become commonplace in the development of embedded systems and as security analysis for cryptographic protocols is becoming standard practice today. Although the fundamental problems in distributed programming are well-known, they pose new challenges in the context of modern system paradigms, including ad-hoc and overlay networks or peer-to-peer systems, and they must be integrated for concrete applications.

## CARTE Team

### 3. Research Program

#### 3.1. Computer Virology

From a historical point of view, the first official virus appeared in 1983 on Vax-PDP 11. At the same time, a series of papers was published which always remains a reference in computer virology: Thompson [71], Cohen [42] and Adleman [29]. The literature which explains and discusses practical issues is quite extensive [46], [48]. However, there are only a few theoretical/scientific studies, which attempt to give a model of computer viruses.

A virus is essentially a self-replicating program inside an adversary environment. Self-replication has a solid background based on works on fixed point in  $\lambda$ -calculus and on studies of von Neumann [76]. More precisely we establish in [38] that Kleene's second recursion theorem [60] is the cornerstone from which viruses and infection scenarios can be defined and classified. The bottom line of a virus behavior is

1. a virus infects programs by modifying them,
2. a virus copies itself and can mutate,
3. it spreads throughout a system.

The above scientific foundation justifies our position to use the word virus as a generic word for self-replicating malwares. There is yet a difference. A malware has a payload, and virus may not have one. For example, a worm is an autonomous self-replicating malware and so falls into our definition. In fact, the current malware taxonomy (virus, worms, trojans, ...) is unclear and subject to debate.

#### 3.2. Computation over continuous structures

Classical recursion theory deals with computability over discrete structures (natural numbers, finite symbolic words). There is a growing community of researchers working on the extension of this theory to continuous structures arising in mathematics. One goal is to give foundations of numerical analysis, by studying the limitations of machines in terms of computability or complexity, when computing with real numbers. Classical questions are : if a function  $f : \mathbb{R} \rightarrow \mathbb{R}$  is computable in some sense, are its roots computable? in which time? Another goal is to investigate the possibility of designing new computation paradigms, transcending the usual discrete-time, discrete-space computer model initiated by the Turing machine that is at the base of modern computers.

While the notion of a computable function over discrete data is captured by the model of Turing machines, the situation is more delicate when the data are continuous, and several non-equivalent models exist. In this case, let us mention computable analysis, which relates computability to topology [45], [74]; the Blum-Shub-Smale model (BSS), where the real numbers are treated as elementary entities [37]; the General Purpose Analog Computer (GPAC) introduced by Shannon [69] with continuous time.

#### 3.3. Rewriting

The rewriting paradigm is now widely used for specifying, modelizing, programming and proving. It allows one to easily express deduction systems in a declarative way, and to express complex relations on infinite sets of states in a finite way, provided they are countable. Programming languages and environments with a rewriting based semantics have been developed ; see ASF+SDF [39], MAUDE [41], and TOM [66].

For basic rewriting, many techniques have been developed to prove properties of rewrite systems like confluence, completeness, consistency or various notions of termination. Proof methods have also been proposed for extensions of rewriting such as equational extensions, consisting of rewriting modulo a set of axioms, conditional extensions where rules are applied under certain conditions only, typed extensions, where rules are applied only if there is a type correspondence between the rule and the term to be rewritten, and constrained extensions, where rules are enriched by formulas to be satisfied [32], [44], [70].

An interesting aspect of the rewriting paradigm is that it allows automatable or semi-automatable correctness proofs for systems or programs: the properties of rewriting systems as those cited above are translatable to the deduction systems or programs they formalize and the proof techniques may directly apply to them.

Another interesting aspect is that it allows characteristics or properties of the modeled systems to be expressed as equational theorems, often automatically provable using the rewriting mechanism itself or induction techniques based on completion [43]. Note that the rewriting and the completion mechanisms also enable transformation and simplification of formal systems or programs.

Applications of rewriting-based proofs to computer security are various. Approaches using rule-based specifications have recently been proposed for detection of computer viruses [72], [73]. For several years, in our team, we have also been working in this direction. We already proposed an approach using rewriting techniques to abstract program behaviors for detecting suspicious or malicious programs [34], [35].

## COMETE Project-Team

### 3. Research Program

#### 3.1. Probability and information theory

**Participants:** Konstantinos Chatzikokolakis, Catuscia Palamidessi, Ehab Elsalamouny, Tymofii Prokopenko, Joris Lamare.

Much of the research of Comète focuses on security and privacy. In particular, we are interested in the problem of the leakage of secret information through public observables.

Ideally we would like systems to be completely secure, but in practice this goal is often impossible to achieve. Therefore, we need to reason about the amount of information leaked, and the utility that it can have for the adversary, i.e. the probability that the adversary is able to exploit such information.

The recent tendency is to use an information theoretic approach to model the problem and define the leakage in a quantitative way. The idea is to consider the system as an information-theoretic *channel*. The input represents the secret, the output represents the observable, and the correlation between the input and output (*mutual information*) represents the information leakage.

Information theory depends on the notion of entropy as a measure of uncertainty. From the security point of view, this measure corresponds to a particular model of attack and a particular way of estimating the security threat (vulnerability of the secret). Most of the proposals in the literature use Shannon entropy, which is the most established notion of entropy in information theory. We, however, consider also other notions, in particular Rényi min-entropy, which seems to be more appropriate for security in common scenarios like one-try attacks.

#### 3.2. Expressiveness of Concurrent Formalisms

**Participants:** Catuscia Palamidessi, Frank Valencia.

We study computational models and languages for distributed, probabilistic and mobile systems, with a particular attention to expressiveness issues. We aim at developing criteria to assess the expressive power of a model or formalism in a distributed setting, to compare existing models and formalisms, and to define new ones according to an intended level of expressiveness, also taking into account the issue of (efficient) implementability.

#### 3.3. Concurrent constraint programming

**Participants:** Michell Guzman, Yamil Salim Perchy, Frank Valencia.

Concurrent constraint programming (ccp) is a well established process calculus for modeling systems where agents interact by posting and asking information in a store, much like in users interact in *social networks*. This information is represented as first-order logic formulae, called constraints, on the shared variables of the system (e.g.,  $X > 42$ ). The most distinctive and appealing feature of ccp is perhaps that it unifies in a single formalism the operational view of processes based upon process calculi with a declarative one based upon first-order logic. It also has an elegant denotational semantics that interprets processes as closure operators (over the set of constraints ordered by entailment). In other words, any ccp process can be seen as an idempotent, increasing, and monotonic function from stores to stores. Consequently, ccp processes can be viewed as: computing agents, formulae in the underlying logic, and closure operators. This allows ccp to benefit from the large body of techniques of process calculi, logic and domain theory.



Our research in ccp develops along the following two lines:

1. **(a)** The study of a bisimulation semantics for ccp. The advantage of bisimulation, over other kinds of semantics, is that it can be efficiently verified.
2. **(b)** The extension of ccp with constructs to capture emergent systems such as those in social networks and cloud computing.

### 3.4. Model checking

**Participants:** Konstantinos Chatzikokolakis, Catuscia Palamidessi.

Model checking addresses the problem of establishing whether a given specification satisfies a certain property. We are interested in developing model-checking techniques for verifying concurrent systems of the kind explained above. In particular, we focus on security and privacy, i.e., on the problem of proving that a given system satisfies the intended security or privacy properties. Since the properties we are interested in have a probabilistic nature, we use probabilistic automata to model the protocols. A challenging problem is represented by the fact that the interplay between nondeterminism and probability, which in security presents subtleties that cannot be handled with the traditional notion of a scheduler,

## DICE Team

### 3. Research Program

#### 3.1. Introduction

Our goal is to address technological issues as well as investigate their impact on society. We believe that addressing both directions simultaneously is essential. More precisely, we focus on the following two objectives:

- Technologies for global intermediation platforms, at reach for unbounded number of users;
- Trans-disciplinary investigations on the global impact of the new intermediation means.

We focus on intermediation platforms, for their increasingly fundamental role in our societies. Intermediation platforms are online systems which offer services to their users, which are well-tuned with their expectation, thanks to the knowledge the platform has accumulated on usage. Search engines and social networks are fundamental examples of intermediation platforms. More generally, intermediation platforms intermediate between producers of services and consumers of services in two-sided markets, with generally one side subsidizing the other. Intermediation will generalise beyond people to things, such as producers or consumers of energy for instance. The capacity to intermediate "in the cloud" with no presence in the physical world in which the market is deployed, by working purely on data with algorithms and in particular learning techniques, is at the heart of the revolution which reshapes our societies.

Platforms ensure a gatekeeping function, always in direct contact with their users, providing them with the most relevant information or contact. They also generate an ecosystem. To do so, platforms allow existing industries as well as new applications proposed by developers to build new services on top of their API. Their impact goes far beyond the Web, while they disrupt step by step all sectors of the economy, transportation, press, education, to name a few.

So far as computer science is concerned, we focus on the technologies used for intermediation, which are at the basis of the largest existing online systems. For the transdisciplinary questions, we focus mostly on the new equilibria that is resulting from the evolution of power balances due mostly to intermediation platforms.

#### 3.2. Intermediation technologies

DICE focuses on intermediation platforms because of the central role they play in the emerging economy.

Intermediation platforms connect users to one another, or users to services with a very high accuracy. They rely on both technological and social innovations. These innovations were unthinkable only a decade ago, when platforms such as Facebook started. They allow communication and interaction between billions of users, gathered in the same digital space, both producers and consumers of data and services. State-of-the-art intermediation platforms include Facebook, Google, Twitter, GitHub, as well as Wikipedia, StackOverflow or Quora. These systems share a common design and their market penetration follows the same pattern. They are built around an initial minimal viable product based on a somehow naive low-tech implementation, which evolves after a few years of improvement to Web giants. Their domination now contributes to standardize the web industry, that means in particular:

- Gatekeeping, a direct relation with users together with services satisfying users' needs;
- Continuous data flows mapped to users' profiles;
- Search engines associating, in a relevant manner, producers, consumers and services.

These common characteristics lead to new software architectural standards, which are shared by all these systems, and used in the peripheral services developed in the ecosystem on top of their API:

- Authentication systems: openId, OAuth, ...
- Object graphs: opengraph, follower/followee scheme, ...
- DataFlow engines: Twitter storm, Google millwheel, ...
- Databases: noSql, keyValues stores, ...
- Application development: javascript, dart, MEAN (Mongo, Express, Angular, Node),...

These architectural components impact the whole digital world. DICE targets systems that use standard architecture services but preserve some aspects we consider as disruptive ones: *data concentration*, *data symmetry* and *computational subsidiarity*. Our current research activity includes the following directions:

- Peer-to-peer design for preserving users' primary data;
- Third parties based organic systems providing subsidiary data computation hosted at peer sites;
- In-Browser applications that impact mobile device and demonstrate instantaneous usability;
- Flow-based computing enabling a stream based exchange of information between peers at runtime.

### 3.3. Economy of intermediation

The recent neologism *uberization* coined after the name of *Uber*, a young intermediation platform, may summarize the effects of the digital revolution. This revolution is impacting all sectors of our societies such as organizations, education, energy, transportation and health, to name a few. This revolution results in a serie of what Schumpeter calls *creative destruction*. As traditional sectors disappear, new ones are created. Our societies, which did not anticipate the depth of the changes, have to struggle to adapt to the pace of the development of the industry. Legal reforms in various important sectors including taxation are at stake. Some countries, more reactive than others, are clearly leading the changes, exploiting the benefits for businesses and the capacity to generate information and value, while others are trying to catch up with the global trends.

Data form the bricks of the information society, and their flows between users and services constitute the blood of the industry. We focus in DICE on the strategic role of data in this revolution, and in particular on the systems that harvest the data and concentrate it. In particular, we focus on *intermediation platforms*. Doing so, we investigate the issues they raise and the disruptions they entail.

We are especially interested in the global political impact of intermediation platforms. The settlement of the *right to be forgotten* in Europe, for instance, exemplifies the new roles platforms are playing: they are both targets of complaints from institutions and mandatory partners in the governance of the world in the digital era. Indeed, they deeply revolutionize the relations between governments and citizens. If privacy is the focus of considerable attention, together with the state surveillance, in Europe in particular, it is only one aspect of the new knowledge made available. Social media produce considerable knowledge not only on individuals, but on populations as well, their economic fate, their political orientation, etc. On the other hand, open data from governments allow citizens to monitor the action of their governments, as well as to contribute to it. The digital revolution, with the capacity to access information in ways unthinkable in the recent past, modifies completely the balance of powers between citizens, states and corporations.

We investigate the digital world, and more precisely the power relations, from an interdisciplinary perspective. We simultaneously quantify power relations by studying data flows and the rise of intermediation platforms and produce an economical, political and ethical analysis of this new state of affairs. Namely, we show that areas such as the US or China dominate the digital world when others, such as Europe, do not succeed in proposing widely used intermediation platforms. This situation generates several conflicts between countries and companies and prevents *weak* countries from promoting their values and policies.

A new trend is emerging in the humanities, around in particular the digital studies, which promote the cooperation between computer scientists and specialists of social sciences. Among them, the Berkman center for Internet and Society in Harvard, the Medialab at MIT, or the Web Science Institute in the UK have gained strong visibility. They address positive as well as negative externalities of IT for societies, that is the new potentials offered as well as their risks. The Center for Information Technology Research in the Interest of Society in Berkeley also addresses fundamental political impacts on democracy, which can be enhanced by open data as well as another philosophy of political power as currently implemented in the State of California for instance. The Open Data Institute in the UK is also a leading center for political issues in Europe. France should catch up on these research trends, at the intersection of different scientific fields.

## PESTO Project-Team

### 3. Research Program

#### 3.1. Modelling

Before being able to analyse and properly design security protocols, it is essential to have a model with a precise semantics of the protocols themselves, the attacker and its capabilities, as well as the properties a protocol needs to ensure.

Most current languages for protocol specification are quite basic and do not provide support for global state, loops, or complex data structures such as lists, or Merkle trees. As an example we may cite Hardware Security Modules that rely on a notion of *mutable global state* which does not arise in traditional protocols, see e.g. the discussion by Herzog [46].

Similarly, the properties a protocol should satisfy are generally not precisely defined, and stating the “right” definitions is often a challenging task in itself. In the case of authentication, many protocol attacks were due to the lack of a precise meaning, cf [44]. While the case of authentication has been widely studied, the recent digitalisation of all kinds of transactions and services, introduces a plethora of new properties, including for instance anonymity in e-voting, untraceability of RFID tokens, verifiability of computations that are outsourced, as well as sanitisation of data in social networks. We expect that many privacy anonymity properties may be modelled as particular observational equivalences in process calculi [40], or indistinguishability between cryptographic games [2], sanitisation of data may also rely on information-theoretic measures.

We also need to take into account that the attacker model changes. While historically the attacker was considered to control the communication network, we may nowadays argue that even (part of) the host executing the software may be compromised through, e.g., malware. This situation motivates the use of secure elements and multi-factor authentication with out-of-band channels. A typical example occurs in e-commerce: to validate an online payment a user needs to enter an additional code sent by the bank via sms to the user’s mobile phone. Such protocols require the possession of a physical device in addition to the knowledge of a password which could have been leaked on an untrusted platform. The fact that data needs to be copied by a human requires these data to be *short*, and hence amenable to brute-force attacks by an attacker or guessing.

#### 3.2. Analysis

##### 3.2.1. Generic proof techniques

Most automated tools for verifying security properties rely on techniques stemming from automated deduction. Often existing techniques do however not apply directly, or do not scale up due to the state explosion problems. For instance, the use of Horn clause resolution techniques requires dedicated resolution methods [34][3]. Another example is unification modulo equational theory, which is a key technique in several tools, e.g. [43]. Security protocols, however require to consider particular equational theories that are not naturally studied in classical automated reasoning. Sometimes, even new concepts have been introduced. One example is the finite variant property [38], which is used in several tools, e.g., *Akiss* [3], Maude-NPA [43] and Tamarin [47]. Another example is the notion of asymmetric unification [42] which is a variant of unification used in Maude-NPA to perform important *syntactic* pruning techniques of the search space, even when reasoning modulo an equational theory. For each of these topics we need to design efficient decision procedures for a variety of equational theories.

### 3.2.2. Dedicated procedures and tools

We will also design dedicated techniques for automated protocol verification. While existing techniques for security protocol verification are efficient and have reached maturity for verification of confidentiality and authentication properties (or more generally safety properties), our goal is to go beyond these properties and the standard attacker models, verifying the properties and attacker models identified in Section 3.1. This includes techniques that

- can analyse *indistinguishability* properties, including for instance anonymity and unlinkability properties, but also properties stated in simulation-based (also known as universally composable) frameworks, which express the security of a protocol as an ideal (correct by design) system;
- take into account protocols that rely on *mutable global state* which does not arise in traditional protocols, but is essential when verifying tamper-resistant hardware devices, e.g., the RSA PKCS#11 standard, IBM's CCA and the trusted platform module (TPM);
- consider attacker models for protocols relying on *weak secrets* that need to be copied or remembered by a human, such as multi-factor authentication.

These goals are beyond the scope of most current analysis tools and require both theoretical advances in the area of verification, as well as the design of new efficient verification tools.

## 3.3. Design

Given our experience in formal analysis of security protocols, including both protocol proofs and findings of flaws, it is tempting to use our experience to design protocols with security in mind and security proofs. This part includes both provably secure design techniques, as well as the development of new protocols.

### 3.3.1. General design techniques

Design techniques will include *composition results* that allow one to design protocols in a modular way [39], [36]. Composition results come in many flavours: they may allow one to compose protocols with different objectives, e.g. compose a key exchange protocol with a protocol that requires a shared key or rely on a protocol for secure channel establishment, compose different protocols in parallel that may re-use some key material, or compose different sessions of a same protocol.

Another area where composition is of particular importance is Service Oriented Computing, where an "orchestrator" must combine some available component services, while guaranteeing some security properties. In this context, we will work on the automated synthesis of the orchestrator or monitors for enforcing the security goals. These problems require to study new classes of automata that communicate with structured messages.

### 3.3.2. New protocol design

We will also design new protocols. Application areas that seem of particular importance are:

- External hardware devices such as security APIs that allow one for flexible key management, including key revocation, and their integration in security protocols. The security *fiasco* of the PKCS#11 standard [35], [41] witnesses the need for new protocols in this area.
- Election systems that provide strong security guarantees. We already work (in collaboration with the Caramba team) on a prototype implementation of an e-voting system, Belenios (<http://belenios.gforge.inria.fr>).
- Mechanisms for publishing personal information (e.g. on social networks) in a controlled way.

**PRIVATICS Project-Team (section vide)**

## PROSECCO Project-Team

### 3. Research Program

#### 3.1. Symbolic verification of cryptographic applications

Despite decades of experience, designing and implementing cryptographic applications remains dangerously error-prone, even for experts. This is partly because cryptographic security is an inherently hard problem, and partly because automated verification tools require carefully-crafted inputs and are not widely applicable. To take just the example of TLS, a widely-deployed and well-studied cryptographic protocol designed, implemented, and verified by security experts, the lack of a formal proof about all its details has regularly led to the discovery of major attacks (including several in 2014) on both the protocol and its implementations, after many years of unsuspecting use.

As a result, the automated verification for cryptographic applications is an active area of research, with a wide variety of tools being employed for verifying different kinds of applications.

In previous work, the we have developed the following three approaches:

- ProVerif: a symbolic prover for cryptographic protocol models
- Tookan: an attack-finder for PKCS#11 hardware security devices
- F7: a security typechecker for cryptographic applications written in F#

##### 3.1.1. Verifying cryptographic protocols with ProVerif

Given a model of a cryptographic protocol, the problem is to verify that an active attacker, possibly with access to some cryptographic keys but unable to guess other secrets, cannot thwart security goals such as authentication and secrecy [42]; it has motivated a serious research effort on the formal analysis of cryptographic protocols, starting with [40] and eventually leading to effective verification tools, such as our tool ProVerif.

To use ProVerif, one encodes a protocol model in a formal language, called the applied pi-calculus, and ProVerif abstracts it to a set of generalized Horn clauses. This abstraction is a small approximation: it just ignores the number of repetitions of each action, so ProVerif is still very precise, more precise than, say, tree automata-based techniques. The price to pay for this precision is that ProVerif does not always terminate; however, it terminates in most cases in practice, and it always terminates on the interesting class of *tagged protocols* [36]. ProVerif also distinguishes itself from other tools by the variety of cryptographic primitives it can handle, defined by rewrite rules or by some equations, and the variety of security properties it can prove: secrecy [34], [25], correspondences (including authentication) [35], and observational equivalences [33]. Observational equivalence means that an adversary cannot distinguish two processes (protocols); equivalences can be used to formalize a wide range of properties, but they are particularly difficult to prove. Even if the class of equivalences that ProVerif can prove is limited to equivalences between processes that differ only by the terms they contain, these equivalences are useful in practice and ProVerif is the only tool that proves equivalences for an unbounded number of sessions.

Using ProVerif, it is now possible to verify large parts of industrial-strength protocols, such as TLS [30], JFK [26], and Web Services Security [32], against powerful adversaries that can run an unlimited number of protocol sessions, for strong security properties expressed as correspondence queries or equivalence assertions. ProVerif is used by many teams at the international level, and has been used in more than 30 research papers (references available at <http://proverif.inria.fr/proverif-users.html>).



### 3.1.2. Verifying security APIs using Tookan

Security application programming interfaces (APIs) are interfaces that provide access to functionality while also enforcing a security policy, so that even if a malicious program makes calls to the interface, certain security properties will continue to hold. They are used, for example, by cryptographic devices such as smartcards and Hardware Security Modules (HSMs) to manage keys and provide access to cryptographic functions whilst keeping the keys secure. Like security protocols, their design is security critical and very difficult to get right. Hence formal techniques have been adapted from security protocols to security APIs.

The most widely used standard for cryptographic APIs is RSA PKCS#11, ubiquitous in devices from smartcards to HSMs. A 2003 paper highlighted possible flaws in PKCS#11 [37], results which were extended by formal analysis work using a Dolev-Yao style model of the standard [38]. However at this point it was not clear to what extent these flaws affected real commercial devices, since the standard is underspecified and can be implemented in many different ways. The Tookan tool, developed by Steel in collaboration with Bortolozzo, Centenaro and Focardi, was designed to address this problem. Tookan can reverse engineer the particular configuration of PKCS#11 used by a device under test by sending a carefully designed series of PKCS#11 commands and observing the return codes. These codes are used to instantiate a Dolev-Yao model of the device's API. This model can then be searched using a security protocol model checking tool to find attacks. If an attack is found, Tookan converts the trace from the model checker into the sequence of PKCS#11 queries needed to make the attack and executes the commands directly on the device. Results obtained by Tookan are remarkable: of 18 commercially available PKCS#11 devices tested, 10 were found to be susceptible to at least one attack.

### 3.1.3. Verifying cryptographic applications using F7 and F\*

Verifying the implementation of a protocol has traditionally been considered much harder than verifying its model. This is mainly because implementations have to consider real-world details of the protocol, such as message formats, that models typically ignore. This leads to a situation that a protocol may have been proved secure in theory, but its implementation may be buggy and insecure. However, with recent advances in both program verification and symbolic protocol verification tools, it has become possible to verify fully functional protocol implementations in the symbolic model.

One approach is to extract a symbolic protocol model from an implementation and then verify the model, say, using ProVerif. This approach has been quite successful, yielding a verified implementation of TLS in F# [30]. However, the generated models are typically quite large and whole-program symbolic verification does not scale very well.

An alternate approach is to develop a verification method directly for implementation code, using well-known program verification techniques such as typechecking. F7 [28] is a refinement typechecker for F#, developed jointly at Microsoft Research Cambridge and Inria. It implements a dependent type-system that allows us to specify security assumptions and goals as first-order logic annotations directly inside the program. It has been used for the modular verification of large web services security protocol implementations [31]. F\* (see below) is an extension of F7 with higher-order kinds and a certifying typechecker. Both F7 and F\* have a growing user community. The cryptographic protocol implementations verified using F7 and F\* already represent the largest verified cryptographic applications to our knowledge.

## 3.2. Computational verification of cryptographic applications

Proofs done by cryptographers in the computational model are mostly manual. Our goal is to provide computer support to build or verify these proofs. In order to reach this goal, we have already designed the automatic tool CryptoVerif, which generates proofs by sequences of games. Much work is still needed in order to develop this approach, so that it is applicable to more protocols. We also plan to design and implement techniques for proving implementations of protocols secure in the computational model, by generating them from CryptoVerif specifications that have been proved secure, or by automatically extracting CryptoVerif models from implementations.

A different approach is to directly verify cryptographic applications in the computational model by typing. A recent work [41] shows how to use refinement typechecking in F7 to prove computational security for protocol implementations. In this method, henceforth referred to as computational F7, typechecking is used as the main step to justify a classic game-hopping proof of computational security. The correctness of this method is based on a probabilistic semantics of F# programs and crucially relies on uses of type abstraction and parametricity to establish strong security properties, such as indistinguishability.

In principle, the two approaches, typechecking and game-based proofs, are complementary. Understanding how to combine these approaches remains an open and active topic of research.

An alternative to direct computation proofs is to identify the cryptographic assumptions under which symbolic proofs, which are typically easier to derive automatically, can be mapped to computational proofs. This line of research is sometimes called computational soundness and the extent of its applicability to real-world cryptographic protocols is an active area of investigation.

### 3.3. F\*: A Higher-Order Effectful Language Designed for Program Verification

F\* [43] is a verification system for ML programs developed collaboratively by Inria and Microsoft Research. ML types are extended with logical predicates that can conveniently express precise specifications for programs (pre- and post- conditions of functions as well as stateful invariants), including functional correctness and security properties. The F\* typechecker implements a weakest-precondition calculus to produce first-order logic formulas that are automatically discharged using the Z3 SMT solver. The original F\* implementation has been successfully used to verify nearly 50,000 lines of code, including cryptographic protocol implementations, web browser extensions, cloudhosted web applications, and key parts of the F\* typechecker and compiler (itself written in F\*). F\* has also been used for formalizing the semantics of other languages, including JavaScript and a compiler from a subset of F\* to JavaScript, and TS\*, a secure subset of TypeScript. Programs verified with F\* can be extracted to F#, OCaml, C, and JavaScript and then efficiently executed and integrated into larger code bases.

The latest version of F\* is written entirely in F\*, and bootstraps in OCaml and F#. It is open source and under active development on GitHub. A detailed description of this new F\* version is available in a POPL 2016 paper [20] and a POPL 2017 one [6]. We continue to evolve and develop F\* and we use it to develop large case studies of verified cryptographic applications, such as miTLS.

### 3.4. Efficient Formally Secure Compilers to a Tagged Architecture

Severe low-level vulnerabilities abound in today's computer systems, allowing cyber-attackers to remotely gain full control. This happens in big part because our programming languages, compilers, and architectures were designed in an era of scarce hardware resources and too often trade off security for efficiency. The semantics of mainstream low-level languages like C is inherently insecure, and even for safer languages, establishing security with respect to a high-level semantics does not guarantee the absence of low-level attacks. Secure compilation using the coarse-grained protection mechanisms provided by mainstream hardware architectures would be too inefficient for most practical scenarios.

We aim to leverage emerging hardware capabilities for fine-grained protection to build the first, efficient secure compilers for realistic programming languages, both low-level (the C language) and high-level (ML and F\*, a dependently-typed variant). These compilers will provide a secure semantics for all programs and will ensure that high-level abstractions cannot be violated even when interacting with untrusted low-level code. To achieve this level of security without sacrificing efficiency, our secure compilers will target a tagged architecture, which associates a metadata tag to each word and efficiently propagates and checks tags according to software-defined rules. We will experimentally evaluate and carefully optimize the efficiency of our secure compilers on realistic workloads and standard benchmark suites. We will use property-based testing and formal verification to provide high confidence that our compilers are indeed secure. Formally, we will construct machine-checked proofs of full abstraction with respect to a secure high-level semantics. This strong property complements

compiler correctness and ensures that no machine-code attacker can do more harm to securely compiled components than a component in the secure source language already could.

### 3.5. Provably secure web applications

Web applications are fast becoming the dominant programming platform for new software, probably because they offer a quick and easy way for developers to deploy and sell their *apps* to a large number of customers. Third-party web-based apps for Facebook, Apple, and Google, already number in the hundreds of thousands and are likely to grow in number. Many of these applications store and manage private user data, such as health information, credit card data, and GPS locations. To protect this data, applications tend to use an ad hoc combination of cryptographic primitives and protocols. Since designing cryptographic applications is easy to get wrong even for experts, we believe this is an opportune moment to develop security libraries and verification techniques to help web application programmers.

As a typical example, consider commercial password managers, such as LastPass, RoboForm, and 1Password. They are implemented as browser-based web applications that, for a monthly fee, offer to store a user's passwords securely on the web and synchronize them across all of the user's computers and smartphones. The passwords are encrypted using a master password (known only to the user) and stored in the cloud. Hence, no-one except the user should ever be able to read her passwords. When the user visits a web page that has a login form, the password manager asks the user to decrypt her password for this website and automatically fills in the login form. Hence, the user no longer has to remember passwords (except her master password) and all her passwords are available on every computer she uses.

Password managers are available as browser extensions for mainstream browsers such as Firefox, Chrome, and Internet Explorer, and as downloadable apps for Android and Apple phones. So, seen as a distributed application, each password manager application consists of a web service (written in PHP or Java), some number of browser extensions (written in JavaScript), and some smartphone apps (written in Java or Objective C). Each of these components uses a different cryptographic library to encrypt and decrypt password data. How do we verify the correctness of all these components?

We propose three approaches. For client-side web applications and browser extensions written in JavaScript, we propose to build a static and dynamic program analysis framework to verify security invariants. To this end, we have developed two security-oriented type systems for JavaScript, Defensive JavaScript [29] [29] and TS\* [45], and used them to guarantee security properties for a number of JavaScript applications. For Android smartphone apps and web services written in Java, we propose to develop annotated JML cryptography libraries that can be used with static analysis tools like ESC/Java to verify the security of application code. For clients and web services written in F# for the .NET platform, we propose to use F\* to verify their correctness. We also propose to translate verified F\* web applications to JavaScript via a verified compiler that preserves the semantics of F\* programs in JavaScript.

### 3.6. Design and Verification of next-generation protocols: identity, blockchains, and messaging

Building on our work on verifying and re-designing pre-existing protocols like TLS and Web Security in general, with the resources provided by the NEXTLEAP project, we are working on both designing and verifying new protocols in rapidly emerging areas like identity, blockchains, and secure messaging. These are all areas where existing protocols, such as the heavily used OAuth protocol, are in need of considerable re-design in order to maintain privacy and security properties. Other emerging areas, such as blockchains and secure messaging, can have modifications to existing pre-standard proposals or even a complete 'clean slate' design. As shown by Prosecco's work, newer standards, such as IETF OAuth, W3C Web Crypto, and W3C Web Authentication API, can have vulnerabilities fixed before standardization is complete and heavily deployed. We hope that the tools used by Prosecco can shape the design of new protocols even before they are shipped to standards bodies.

## TAMIS Team

### 3. Research Program

#### 3.1. Axis 1: Vulnerability analysis

This axis proposes different techniques to discover vulnerabilities in systems. The outcomes of this axis are (a) new techniques to discover system vulnerabilities as well as to analyze them, and (b) to understand the importance of the hardware support.

Most existing approaches used at the engineering level rely on testing and fuzzing. Such techniques consist in simulating the system for various input values, and then checking that the result conforms to a given standard. The problem being the large set of inputs to be potentially tested. Existing solutions propose to extract significant sets by mutating a finite set of inputs. Other solutions, especially concolic testing developed at Microsoft, propose to exploit symbolic executions to extract constraints on new values. We build on those existing work, and extend them with recent techniques based on dissimilarity distances and learning. We also account for the execution environment, and study techniques based on the combination of timing attacks with fuzzing techniques to discover and classify classes of behavior of the system under test.

Techniques such as model checking and static analysis have been used for verifying several types of requirements such as safety and reliability. Recently, several works have attempted to adapt model checking to the detection of security issues. It has clearly been identified that this required to work at the level of binary code. Applying formal techniques to such code requires the development of disassembly techniques to obtain a semantically well-defined model. One of the biggest issues faced with formal analysis is the state space explosion problem. This problem is amplified in our context as representations of data (such as stack content) definitively blow up the state space. We propose to use statistical model checking (SMC) of rare events to efficiently identify problematic behaviors.

We also seek to understand vulnerabilities at the architecture and hardware levels. Particularly, we evaluate vulnerabilities of the interfaces and how an adversary could use them to get access to core assets in the system. One particular mechanism to be investigated is the DMA and the so-called Trustzone. An ad-hoc technique to defend against adversarial DMA-access to memory is to keep key material exclusively in registers. This implies co-analyzing machine code and an accurate hardware model.

#### 3.2. Axis 2: Malware analysis

Axis 1 is concerned with vulnerabilities. Such vulnerabilities can be exploited by an attacker in order to introduce malicious behaviors in a system. Another method to identify vulnerabilities is to analyze malware that exploits them. However, modern malware has a wide variety of analysis avoidance techniques. In particular, attackers obfuscate the code leading to a security exploit. For doing so, recent black hat research suggests hiding constants in program choices via polynomials. Such techniques hinder forensic analysis by making detailed analysis labor intensive and time consuming. The objective of research axis 2 is to obtain a full tool chain for malware analysis starting from (a) the observability of the malware via deobfuscation, and (b) the analysis of the resulting binary file. A complementary objective is to understand how hardware attacks can be exploited by malwares.

We first investigate obfuscation techniques. Several solutions exist to mitigate the packer problem. As an example, we try to reverse the packer and remove the environment evaluation in such a way that it performs the same actions and outputs the resulting binary for further analysis. There is a wide range of techniques to obfuscate malware, which includes flattening and virtualization. We will produce a taxonomy of both techniques and tools. We will first give a particular focus to control flow obfuscation via mixed Boolean algebra, which is highly deployed for malware obfuscation. We recently showed that a subset of them can be broken via SAT-solving and synthesis. Then, we will expand our research to other obfuscation techniques.

Once the malware code has been unpacked/deobfuscated, the resulting binary still needs to be fully understood. Advanced malware often contains multiple stages, multiple exploits and may unpack additional features based on its environment. Ensuring that one understands all interesting execution paths of a malware sample is related to enumerating all of the possible execution paths when checking a system for vulnerabilities. The main difference is that in one case we are interested in finding vulnerabilities and in the other in finding exploitative behavior that may mutate. Still, some of the techniques of Axis 1 can be helpful in analyzing malware. The main challenge for axis 2 is thus to adapt the tools and techniques to deal with binary programs as inputs, as well as the logic used to specify malware behavior, including behavior with potentially rare occurrences. Another challenge is to take mutation into account, which we plan to do by exploiting mining algorithms.

Most recent attacks against hardware are based on fault injection which dynamically modifies the semantics of the code. We demonstrated the possibility to obfuscate code using constraint solver in such a way that the code becomes intentionally hostile while hit by a laser beam. This new form of obfuscation opens a new challenge for secure devices where malicious programs can be designed and uploaded that defeat comprehensive static analysis tools or code reviews, due to their multi-semantic nature. We have shown on several products that such an attack cannot be mitigated with the current defenses embedded in Java cards. In this research, we first aim at extending the work on fault injection, then at developing new techniques to analyze such hostile code. This is done by proposing formal models of fault injection, and then reusing results from our work on obfuscation/deobfuscation.

### **3.3. Axis 3: Building a secure network stack**

To evaluate the techniques developed in Axes 1 and 2, we analyze concrete systems developed not only with industry partners, but also within the team. By using our own systems, we can co-evolve best-practices, while externally developed systems provide realistic challenges especially with respect to analyzing obfuscated malware in the hardware or complex vulnerabilities. In this context, Christian Grothoff (ARP Inria) is currently developing a new Internet, which is supposed to be more secure. This introduces interesting challenges both in terms of vulnerability and malware analysis, and hence should be a great opportunity to mix the competences of all the members of the team.

More precisely, this system intends to challenge the idea that network security is an administrative task, where network administrators shield users with passwords, firewalls, intrusion detection systems and policies. Instead, we want to eliminate administrators that have power over user's data, and as such administrators themselves are liabilities, and because a network design that permits administrative intrusion inherently adds vulnerabilities. Instead, the system should ensure secure communication mechanisms without trusted third parties.

Key challenges we work on include (a) improving scalable secure ad-hoc decentralized routing, including key-value lookup, unicast and multicast communication, (b) protecting meta-data in the overlay using advanced decentralized onion routing, (c) a unified public-key infrastructure and identity management solution that is suitable to replace the Web-of-Trust, X.509, DNSSEC and other legacy methods for naming and identifying services, (d) secure synchronous and asynchronous messaging at scale, providing decentralized alternatives to common online social applications and addressing challenges in protocol evolution and compatibility. Finally, we are currently working on GNU Taler, a new secure privacy-preserving payment system where users never have to authenticate. This system in particular can be used as a concrete test case for the methods developed in the team.

To support this research work, we develop a framework named GNUnet. It provides a clear separation into layers, which facilitates testing and verifying the various components. However, we see that often existing formal verification techniques still do not scale to typical subsystems encountered in practice. Our objective is thus to exploit efficient and scalable formal techniques proposed in Axis 1 together with engineering skills in order to guide the validation (message synchronization, data protection, ...) and reach the best compromise. An additional complication is that we need a validation process that not merely covers the software itself, but also all of its dependencies (such as database, cryptographic libraries and networking libraries). For the Taler-specific hardware, we are envisioning an NFC-powered device, which creates new

challenges in terms of securing cryptographic computations in a setting where the adversary has control over the power supply. In such a case, the attacker can drive the environment and modify the behavior of the system as we have shown in Axis 2. Providing the control of the environment is a new vector for attackers.