

Inria

RESEARCH CENTER

FIELD

Algorithmics, Programming, Software and Architecture

Activity Report 2019

Section Scientific Foundations

Edition: 2020-03-21

ALGORITHMICS, COMPUTER ALGEBRA AND CRYPTOLOGY

1. ARIC Project-Team	5
2. AROMATH Project-Team	8
3. CARAMBA Project-Team	10
4. CASCADE Project-Team	13
5. DATASHAPE Project-Team	15
6. GAMBLE Project-Team	17
7. GRACE Project-Team	21
8. LFANT Project-Team	24
9. OURAGAN Project-Team	27
10. POLSYS Project-Team	30
11. SECRET Project-Team	34
12. SPECFUN Project-Team	36

ARCHITECTURE, LANGUAGES AND COMPILATION

13. CAIRN Project-Team	41
14. CAMUS Project-Team	44
15. CASH Project-Team	47
16. CORSE Project-Team	57
17. PACAP Project-Team	58

EMBEDDED AND REAL-TIME SYSTEMS

18. HYCOMES Project-Team	65
19. Kairos Project-Team	71
20. KOPERNIC Team	74
21. PARKAS Project-Team	76
22. SPADES Project-Team	79
23. TEA Project-Team	81

PROOFS AND VERIFICATION

24. ANTIQUE Project-Team	85
25. CAMBIUM Project-Team	88
26. CELTIQUE Project-Team (section vide)	89
27. CONVECS Project-Team	90
28. DEDUCTTEAM Project-Team	94
29. GALLINETTE Project-Team	95
30. MEXICO Project-Team	104
31. MOCQUA Team	109
32. PARSIFAL Project-Team	111
33. PIR2 Project-Team	114
34. STAMP Project-Team	121
35. SUMO Project-Team	122
36. TOCCATA Project-Team	125
37. VERIDIS Project-Team	129

SECURITY AND CONFIDENTIALITY

38. CIDRE Project-Team	131
39. COMETE Project-Team	133
40. DATASPHERE Team	135
41. PESTO Project-Team	136
42. PRIVATICS Project-Team (section vide)	138
43. PROSECCO Project-Team	139
44. TAMIS Project-Team	143

ARIC Project-Team

3. Research Program

3.1. Efficient and certified approximation methods

3.1.1. *Safe numerical approximations*

The last twenty years have seen the advent of computer-aided proofs in mathematics and this trend is getting more and more important. They request: fast and stable numerical computations; numerical results with a guarantee on the error; formal proofs of these computations or computations with a proof assistant. One of our main long-term objectives is to develop a platform where one can study a computational problem on all (or any) of these three levels of rigor. At this stage, most of the necessary routines are not easily available (or do not even exist) and one needs to develop *ad hoc* tools to complete the proof. We plan to provide more and more algorithms and routines to address such questions. Possible applications lie in the study of mathematical conjectures where exact mathematical results are required (e.g., stability of dynamical systems); or in more applied questions, such as the automatic generation of efficient and reliable numerical software for function evaluation. On a complementary viewpoint, numerical safety is also critical in robust space mission design, where guidance and control algorithms become more complex in the context of increased satellite autonomy. We will pursue our collaboration with specialists of that area whose questions bring us interesting focus on relevant issues.

3.1.2. *Floating-point computing*

Floating-point arithmetic is currently undergoing a major evolution, in particular with the recent advent of a greater diversity of available precisions on a same system (from 8 to 128 bits) and of coarser-grained floating-point hardware instructions. This new arithmetic landscape raises important issues at the various levels of computing, that we will address along the following three directions.

3.1.2.1. *Floating-point algorithms, properties, and standardization*

One of our targets is the design of building blocks of computing (e.g., algorithms for the basic operations and functions, and algorithms for complex or double-word arithmetic). Establishing properties of these building blocks (e.g., the absence of “spurious” underflows/overflows) is also important. The IEEE 754 standard on floating-point arithmetic (whose next version, a rather minor revision, will be released soon) will have to undergo a major revision within a few years: first because advances in technology or new needs make some of its features obsolete, and because new features need standardization. We aim at playing a leading role in the preparation of the next standard.

3.1.2.2. *Error bounds*

We will pursue our studies in rounding error analysis, in particular for the “low precision–high dimension” regime, where traditional analyses become ineffective and where improved bounds are thus most needed. For this, the structure of both the data and the errors themselves will have to be exploited. We will also investigate the impact of mixed-precision and coarser-grained instructions (such as small matrix products) on accuracy analyses.

3.1.2.3. *High performance kernels*

Most directions in the team are concerned with optimized and high performance implementations. We will pursue our efforts concerning the implementation of well optimized floating-point kernels, with an emphasis on numerical quality, and taking into account the current evolution in computer architectures (the increasing width of SIMD registers, and the availability of low precision formats). We will focus on computing kernels used within other axes in the team such as, for example, extended precision linear algebra routines within the FPLLL and HPLLL libraries.

3.2. Lattices: algorithms and cryptology

We intend to strengthen our assessment of the cryptographic relevance of problems over lattices, and to broaden our studies in two main (complementary) directions: hardness foundations and advanced functionalities.

3.2.1. *Hardness foundations*

Recent advances in cryptography have broadened the scope of encryption functionalities (e.g., encryption schemes allowing to compute over encrypted data or to delegate partial decryption keys). While simple variants (e.g., identity-based encryption) are already practical, the more advanced ones still lack efficiency. Towards reaching practicality, we plan to investigate simpler constructions of the fundamental building blocks (e.g., pseudorandom functions) involved in these advanced protocols. We aim at simplifying known constructions based on standard hardness assumptions, but also at identifying new sources of hardness from which simple constructions that are naturally suited for the aforementioned advanced applications could be obtained (e.g., constructions that minimize critical complexity measures such as the depth of evaluation). Understanding the core source of hardness of today's standard hard algorithmic problems is an interesting direction as it could lead to new hardness assumptions (e.g., tweaked version of standard ones) from which we could derive much more efficient constructions. Furthermore, it could open the way to completely different constructions of advanced primitives based on new hardness assumptions.

3.2.2. *Cryptanalysis*

Lattice-based cryptography has come much closer to maturity in the recent past. In particular, NIST has started a standardization process for post-quantum cryptography, and lattice-based proposals are numerous and competitive. This dramatically increases the need for cryptanalysis: Do the underlying hard problems suffer from structural weaknesses? Are some of the problems used easy to solve, e.g., asymptotically? Are the chosen concrete parameters meaningful for concrete cryptanalysis? In particular, how secure would they be if all the known algorithms and implementations thereof were pushed to their limits? How would these concrete performances change in case (full-fledged) quantum computers get built?

On another front, the cryptographic functionalities reachable under lattice hardness assumptions seem to get closer to an intrinsic ceiling. For instance, to obtain cryptographic multilinear maps, functional encryption and indistinguishability obfuscation, new assumptions have been introduced. They often have a lattice flavour, but are far from standard. Assessing the validity of these assumptions will be one of our priorities in the mid-term.

3.2.3. *Advanced cryptographic primitives*

In the design of cryptographic schemes, we will pursue our investigations on functional encryption. Despite recent advances, efficient solutions are only available for restricted function families. Indeed, solutions for general functions are either way too inefficient for practical use or they rely on uncertain security foundations like the existence of circuit obfuscators (or both). We will explore constructions based on well-studied hardness assumptions and which are closer to being usable in real-life applications. In the case of specific functionalities, we will aim at more efficient realizations satisfying stronger security notions.

Another direction we will explore is multi-party computation via a new approach exploiting the rich structure of class groups of quadratic fields. We already showed that such groups have a positive impact in this field by designing new efficient encryption switching protocols from the additively homomorphic encryption we introduced earlier. We want to go deeper in this direction that raises interesting questions such as how to design efficient zero-knowledge proofs for groups of unknown order, how to exploit their structure in the context of 2-party cryptography (such as two-party signing) or how to extend to the multi-party setting.

In the context of the PROMETHEUS H2020 project, we will keep seeking to develop new quantum-resistant privacy-preserving cryptographic primitives (group signatures, anonymous credentials, e-cash systems, etc). This includes the design of more efficient zero-knowledge proof systems that can interact with lattice-based cryptographic primitives.

3.3. Algebraic computing and high performance kernels

The connections between algorithms for structured matrices and for polynomial matrices will continue to be developed, since they have proved to bring progress to fundamental questions with applications throughout computer algebra. The new fast algorithm for the bivariate resultant opens an exciting area of research which should produce improvements to a variety of questions related to polynomial elimination. Obviously, we expect to produce results in that area.

For definite summation and integration, we now have fast algorithms for single integrals of general functions and sequences and for multiple integrals of rational functions. The long-term objective of that part of computer algebra is an efficient and general algorithm for multiple definite integration and summation of general functions and sequences. This is the direction we will take, starting with single definite sums of general functions and sequences (leading in particular to a faster variant of Zeilberger's algorithm). We also plan to investigate geometric issues related to the presence of apparent singularities and how they seem to play a role in the complexity of the current 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 knots. 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 relies 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 since 2014, notably for non-prime fields, 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 (whose last commit is from August 2018). In Lausanne, T. Kleinjung develops his own code base, which is unfortunately not public.

The work plan 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 for Cryptology

The challenges associated with 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. With the standardization of TLS 1.3 in 2018 [34], the curves `x25519` and `x448` have entered the base specification of standard. These curves were designed by academia and offer an excellent compromise between efficiency and security.

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. Symmetric Cryptography

Since the recruiting of Marine Minier in September 2016 as a Professor at the Université de Lorraine, and of Virginie Lallemand as a CNRS researcher in October 2018, a new research domain has emerged in the CARAMBA team: symmetric key cryptology. Accompanied in this adventure by non-permanent team members, we are tackling problems related to both design and analysis. A large part of our recent researches has been motivated by the Lightweight Cryptography Standardization Process of the NIST⁰ that embodies a crucial challenge of the last decade: finding ciphers that are suitable for resource-constrained devices.

On a general note, the working program of CARAMBA in symmetric cryptography is defined as follows:

- Develop automatic tools based on constraint programming to help finding optimum attack parameters. The effort will be focused on the AES standard and on recent lightweight cipher proposals.
- Contribute to the security and performance analysis effort required to sort out the candidates for the NIST Lightweight Cryptography Standardization Process.
- Study how to protect services execution on dedicated platforms using white-box cryptography and software obfuscation methods.

3.4. Computer Arithmetic

Computer arithmetic is part of the common background of all team members, and is naturally ubiquitous in our application domains. 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 advanced 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.

3.5. 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.

⁰National Institute of Standard and Technology.

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 basis algorithms that can achieve large speedups compared to generic implementations [29], [28].

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 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 Cryptography in general. On the narrower topic of the Elliptic Curve Discrete Logarithm Problem on small characteristic finite fields, the highly structured polynomial systems that are involved match well our expertise on the topic of polynomial systems. Once a very hot topic in 2015, activity on this precise problem seems to have slowed down. Yet, the conjunction of skills that we have may lead to results in this direction in the future.
- The hiring of Marine Minier is likely to lead the team to study particular polynomial systems in contexts 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 [29], [28]. 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. Quantum-Safe 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 computing discrete logarithms. This is 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, which also offer features that other alternative public-key cryptosystems (such as MQ-based, code-based or hash-based schemes) cannot provide.

3.2. Advanced Encryption

Fully Homomorphic Encryption (FHE) has become a very active research area since 2009, when IBM announced the discovery of a FHE scheme by Craig Gentry. FHE allows to perform any computation on encrypted data, yielding the result encrypted under the same key. This enables outsourcing computation in the Cloud, on encrypted data, so the Cloud provider does not learn any information. However, FHE does not allow to share the result.

Functional encryption is another recent tool that allows an authority to deliver functional decryption keys, for any function f of his choice, so that when applied to the encryption of a message m , the functional decryption key yields $f(m)$. Since m can be a large vector, f can be an aggregation or statistical function: on encrypted data, one can get the result $f(m)$ in clear.

While this functionality has initially been defined in theory, our team has been very active in designing concrete instantiations for practical purposes.

3.3. Security amidst Concurrency on the Internet

Cryptographic protocols that are secure when executed in isolation can become 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, 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, and
2. 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 and the Blockchain

Electronic cash (e-cash) was first proposed in the 1980s but has never been deployed on a large scale. Other means of digital payments are instead largely replacing physical cash, but they do not respect the citizens' right to privacy, which includes their right of anonymous payments of moderate sums. Recently, so-called decentralized currencies, such as Bitcoin, have become a third type of payments in addition to physical cash, and card and other (non-anonymous) electronic payments. The continuous growth of popularity and usage of this new kind of currencies, also called "cryptocurrencies", have triggered a renewed interest in cryptographic e-cash.

On the one hand, our group investigates "centralized" e-cash, in keeping with the current economic model that has money be issued by (central) banks (while cryptocurrencies use money distribution as an incentive for participation in the system, on which its stability hinges). Of particular interest among centralized e-cash schemes is transferable e-cash, which allows users to transfer coins between each other without interacting with a third party (or the blockchain). Existing efficient e-cash schemes are not transferable, as they require coins to be deposited at the bank after having been used in a payment. Our goal is to propose efficient transferable e-cash schemes.

Another direction concerns (decentralized) cryptocurrencies, whose adoption has grown tremendously over the last few years. While in Bitcoin all transactions are publicly posted on the so-called "blockchain", other cryptocurrencies such as *Zcash* respect user privacy, whose security guarantees we have analyzed. Apart from privacy, two pressing challenges for cryptocurrencies, and blockchains in general, are sustainability and scalability. Regarding the former, we are addressing the electricity waste caused by the concept of "proof of work" used by all major cryptocurrencies by proposing alternatives; for the latter, we are working on proposals that avoid the need for all data having to be stored on the blockchain forever.

Blockchains have meanwhile found many other applications apart from electronic money. Together with Microsoft Research, our group investigates decentralized means of authentication that uses cryptography to guarantee privacy.

DATASHAPE Project-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 underlie 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 software⁰ 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.

⁰<https://www.r-project.org/>

GAMBLE Project-Team

3. Research Program

3.1. Non-linear computational geometry



Figure 1. Two views of the Whitney umbrella (on the left, the “stick” of the umbrella, i.e., the negative z -axis, is missing). Right picture from [\[Wikipedia\]](#), left picture from [\[Lachaud et al.\]](#).

As mentioned above, curved objects are ubiquitous in real world problems and in computer science and, despite this fact, there are very few problems on curved objects that admit robust and efficient algorithmic solutions without first discretizing the curved objects into meshes. Meshing curved objects induces a loss of accuracy which is sometimes not an issue but which can also be most problematic depending on the application. In addition, discretization induces a combinatorial explosion which could cause a loss in efficiency compared to a direct solution on the curved objects (as our work on quadrics has demonstrated with flying colors [\[50\]](#), [\[51\]](#), [\[52\]](#), [\[54\]](#), [\[58\]](#)). But it is also crucial to know that even the process of computing meshes that approximate curved objects is far from being resolved. As a matter of fact there is no algorithm capable of computing in practice meshes with certified topology of even rather simple singular 3D surfaces, due to the high constants in the theoretical complexity and the difficulty of handling degenerate cases. Part of the difficulty comes from the unintuitive fact that the structure of an algebraic object can be quite complicated, as depicted in the Whitney umbrella (see [Figure 1](#)), surface of equation $x^2 = y^2z$ on which the origin (the “special” point of the surface) is a vertex of the arrangement induced by the surface while the singular locus is simply the whole z -axis. Even in 2D, meshing an algebraic curve with the correct topology, that is in other words producing a correct drawing of the curve (without knowing where the domain of interest is), is a very difficult problem on which we have recently made important contributions [\[37\]](#), [\[38\]](#), [\[59\]](#).

It is thus to be understood that producing practical robust and efficient algorithmic solutions to geometric problems on curved objects is a challenge on all and even the most basic problems. The basicness and fundamentality of two problems we mentioned above on the intersection of 3D quadrics and on the drawing in a topologically certified way of plane algebraic curves show rather well that the domain is still in its infancy. And it should be stressed that these two sets of results were not anecdotal but flagship results produced during the lifetime of the VEGAS team (the team preceding GAMBLE).

There are many problems in this theme that are expected to have high long-term impacts. Intersecting NURBS (Non-uniform rational basis splines) in a certified way is an important problem in computer-aided design and manufacturing. As hinted above, meshing objects in a certified way is important when topology matters. The 2D case, that is essentially drawing plane curves with the correct topology, is a fundamental problem with

far-reaching applications in research or R&D. Notice that on such elementary problems it is often difficult to predict the reach of the applications; as an example, we were astonished by the scope of the applications of our software on 3D quadric intersection⁰ which was used by researchers in, for instance, photochemistry, computer vision, statistics and mathematics.

3.2. Non-Euclidean computational geometry



Figure 2. Left: 3D mesh of a gyroid (triply periodic surface) [61]. Right: Simulation of a periodic Delaunay triangulation of the hyperbolic plane [33].

Triangulations, in particular Delaunay triangulations, in the *Euclidean space* \mathbb{R}^d have been extensively studied throughout the 20th century and they are still a very active research topic. Their mathematical properties are now well understood, many algorithms to construct them have been proposed and analyzed (see the book of Aurenhammer *et al.* [32]). Some members of GAMBLE have been contributing to these algorithmic advances (see, e.g. [36], [68], [47], [35]); they have also contributed robust and efficient triangulation packages through the state-of-the-art Computational Geometry Algorithms Library CGAL whose impact extends far beyond computational geometry. Application fields include particle physics, fluid dynamics, shape matching, image processing, geometry processing, computer graphics, computer vision, shape reconstruction, mesh generation, virtual worlds, geophysics, and medical imaging.⁰

It is fair to say that little has been done on non-Euclidean spaces, in spite of the large number of questions raised by application domains. Needs for simulations or modeling in a variety of domains⁰ ranging from the infinitely small (nuclear matter, nano-structures, biological data) to the infinitely large (astrophysics) have led us to consider 3D periodic Delaunay triangulations, which can be seen as Delaunay triangulations in the 3D *flat torus*, quotient of \mathbb{R}^3 under the action of some group of translations [42]. This work has already yielded a fruitful collaboration with astrophysicists [55], [69] and new collaborations with physicists are emerging. To the best of our knowledge, our CGAL package [41] is the only publicly available software that computes Delaunay triangulations of a 3D flat torus, in the special case where the domain is cubic. This case, although restrictive, is already useful.⁰ We have also generalized this algorithm to the case of general d -dimensional compact flat manifolds [43]. As far as non-compact manifolds are concerned, past approaches, limited to the two-dimensional case, have stayed theoretical [60].

Interestingly, even for the simple case of triangulations on the *sphere*, the software packages that are currently available are far from offering satisfactory solutions in terms of robustness and efficiency [40].

⁰QI: [web](#).

⁰See [Projects using CGAL](#) for details.

⁰See [CGAL Prospective Workshop on Geometric Computing in Periodic Spaces](#), [Subdivide and Tile: Triangulating spaces for understanding the world](#), [Computational geometry in non-Euclidean spaces](#), [Shape Up 2015 : Exercises in Materials Geometry and Topology](#)

⁰See examples at [Projects using CGAL](#)

Moreover, while our solution for computing triangulations in hyperbolic spaces can be considered as ultimate [33], the case of *hyperbolic manifolds* has hardly been explored. Hyperbolic manifolds are quotients of a hyperbolic space by some group of hyperbolic isometries. Their triangulations can be seen as hyperbolic periodic triangulations. Periodic hyperbolic triangulations and meshes appear for instance in geometric modeling [62], neuromathematics [45], or physics [65]. Even the case of the Bolza surface (a surface of genus 2, whose fundamental domain is the regular octagon in the hyperbolic plane) shows mathematical difficulties [34], [57].

3.3. Probability in computational geometry

In most computational geometry papers, algorithms are analyzed in the worst-case setting. This often yields too pessimistic complexities that arise only in pathological situations that are unlikely to occur in practice. On the other hand, probabilistic geometry provides analyses with great precision [63], [64], [39], but using hypotheses with much more randomness than in most realistic situations. We are developing new algorithmic designs improving state-of-the-art performance in random settings that are not overly simplified and that can thus reflect many realistic situations.

Twelve years ago, smooth analysis was introduced by Spielman and Teng analyzing the simplex algorithm by averaging on some noise on the data [67] (and they won the Gödel prize). In essence, this analysis smoothes the complexity around worst-case situations, thus avoiding pathological scenarios but without considering unrealistic randomness. In that sense, this method makes a bridge between full randomness and worst case situations by tuning the noise intensity. The analysis of computational geometry algorithms within this framework is still embryonic. To illustrate the difficulty of the problem, we started working in 2009 on the smooth analysis of the size of the convex hull of a point set, arguably the simplest computational geometry data structure; then, only one very rough result from 2004 existed [46] and we only obtained in 2015 breakthrough results, but still not definitive [49], [48], [53].

Another example of a problem of different flavor concerns Delaunay triangulations, which are rather ubiquitous in computational geometry. When Delaunay triangulations are computed for reconstructing meshes from point clouds coming from 3D scanners, the worst-case scenario is, again, too pessimistic and the full randomness hypothesis is clearly not adapted. Some results exist for “good samplings of generic surfaces” [31] but the big result that everybody wishes for is an analysis for random samples (without the extra assumptions hidden in the “good” sampling) of possibly non-generic surfaces.

Trade-offs between full randomness and worst case may also appear in other forms such as dependent distributions, or random distributions conditioned to be in some special configurations. Simulating these kinds of geometric distributions is currently out of reach for more than a few hundred points [56] although it has practical applications in physics or networks.

3.4. Discrete geometric structures

Our work on discrete geometric structures develops in several directions, each one probing a different type of structure. Although these objects appear unrelated at first sight, they can be tackled by the same set of probabilistic and topological tools.

A first research topic is the study of *Order types*. Order types are combinatorial encodings of finite (planar) point sets, recording for each triple of points the orientation (clockwise or counterclockwise) of the triangle they form. This already determines properties such as convex hulls or half-space depths, and the behaviour of algorithms based on orientation predicates. These properties for all (infinitely many) n -point sets can be studied through the finitely many order types of size n . Yet, this finite space is poorly understood: its estimated size leaves an exponential margin of error, no method is known to sample it without concentrating on a vanishingly small corner, the effect of pattern exclusion or VC dimension-type restrictions are unknown. These are all directions we actively investigate.

A second research topic is the study of *Embedded graphs and simplicial complexes*. Many topological structures can be effectively discretized, for instance combinatorial maps record homotopy classes of embedded graphs and simplicial complexes represent a large class of topological spaces. This raises many structural and algorithmic questions on these discrete structures; for example, given a closed walk in an embedded graph, can we find a cycle of the graph homotopic to that walk? (The complexity status of that problem is unknown.) Going in the other direction, some purely discrete structures can be given an associated topological space that reveals some of their properties (*e.g.* the Nerve theorem for intersection patterns). An open problem is for instance to obtain fractional Helly theorems for set system of bounded topological complexity.

Another research topic is that of *Sparse inclusion-exclusion formulas*. For any family of sets A_1, A_2, \dots, A_n , by the principle of inclusion-exclusion we have

$$\mathbb{1}_{\bigcup_{i=1}^n A_i} = \sum_{I \subseteq \{1, 2, \dots, n\}} (-1)^{|I|+1} \mathbb{1}_{\bigcap_{i \in I} A_i} \quad (1)$$

where $\mathbb{1}_X$ is the indicator function of X . This formula is universal (it applies to any family of sets) but its number of summands grows exponentially with the number n of sets. When the sets are balls, the formula remains true if the summation is restricted to the regular triangulation; we proved that similar simplifications are possible whenever the Venn diagram of the A_i is sparse. There is much room for improvements, both for general set systems and for specific geometric settings. Another interesting problem (the subject of the PhD thesis of Galatée Hemery) is to combine these simplifications with the inclusion-exclusion algorithms developed, for instance, for graph coloring.

GRACE Project-Team

3. Research Program

3.1. Algorithmic Number Theory

Participants: Luca de Feo, François Morain, Benjamin Smith, Mathilde de La Morinerie, Antonin Leroux, Guénaél Renault.

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);
- 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

Participants: Luca de Feo, François Morain, Benjamin Smith, Mathilde de La Morinerie, Antonin Leroux.

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

Participants: Luca de Feo, François Morain, Benjamin Smith, Mathilde de La Morinerie, Antonin Leroux.

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

Participants: Daniel Augot, Alain Couvreur, Françoise Levy-Dit-Vehel, Maxime Roméas, Sarah Bordage, Adrien Hauteville, Isabella Panaccione.

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, Jared Guissmo Asuncion, Karim Belabas, Jean-Paul Cerri, Henri Cohen, Jean-Marc Couveignes, Andreas Enge, Fredrik Johansson, Aurel Page.

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 $n \geq 3$. 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 ζ , 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, ζ 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, ζ 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* 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 (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 [36] for a recent account.

The *analytic class number formula* links the invariants h_K and R_K (unfortunately, only their product) to the ζ -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 ζ - 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 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, Damien Robert, Jean Kieffer, Razvan Barbulescu.

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: Jared Guissmo Asuncion, Karim Belabas, Henri Cohen, Jean-Marc Couveignes, Andreas Enge, Fredrik Johansson, Chloe Martindale, Damien Robert.

Complex multiplication provides a link between number fields and algebraic curves; for a concise introduction in the elliptic curve case, see [38], for more background material, [37]. 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 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 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.

OURAGAN Project-Team

3. Research Program

3.1. Basic computable objects and algorithms

The development of basic computable objects is somehow *on demand* and depends on all the other directions. However, some critical computations are already known to be bottlenecks and are sources of constant efforts.

Computations with algebraic numbers appear in almost all our activities: when working with number fields in our work in algorithmic number theory as well as in all the computations that involve the use of solutions of zero-dimensional systems of polynomial equations. Among the identified problems: finding good representations for single number fields (optimizing the size and degree of the defining polynomials), finding good representations for towers or products of number fields (typically working with a tower or finding a unique good extension), efficiently computing in practice with number fields (using certified approximation vs working with the formal description based on polynomial arithmetics). Strong efforts are currently done in the understanding of the various strategies by means of tight theoretical complexity studies [70], [115], [50] and many other efforts will be required to find the right representation for the right problem in practice. For example, for isolating critical points of plane algebraic curves, it is still unclear (at least the theoretical complexity cannot help) that an intermediate formal parameterization is more efficient than a triangular decomposition of the system and it is still unclear that these intermediate computations could be dominated in time by the certified final approximation of the roots.

3.2. Algorithmic Number Theory

Concerning algorithmic number theory, the main problems we will be considering in the coming years are the following:

- *Number fields.* We will continue working on the problems of class groups and generators. In particular, the existence and accessibility of *good* defining polynomials for a fixed number field remain very largely open. The impact of better polynomials on the algorithmic performance is a very important parameter, which makes this problem essential.
- *Lattice reduction.* Despite a great amount of work in the past 35 years on the LLL algorithm and its successors, many open problems remain. We will continue the study of the use of interval arithmetic in this field and the analysis of variants of LLL along the lines of the *Potential-LLL* which provides improved reduction comparable to BKZ with a small block size but has better performance.
- *Elliptic curves and Drinfeld modules.* The study of elliptic curves is a very fruitful area of number theory with many applications in crypto and algorithms. Drinfeld modules are “cousins” of elliptic curves which have been less explored in the algorithm context. However, some recent advances [74] have used them to provide some fast sophisticated factoring algorithms. As a consequence, it is natural to include these objects in our research directions.

3.2.1. Rigorous numerical computations

Some studies in this area will be driven by some other directions, for example, the rigorous evaluation of non algebraic functions on algebraic varieties might become central for some of our work on topology in small dimension (volumes of varieties, drawing of amoeba) or control theory (approximations of discriminant varieties) are our two main current sources of interesting problems. In the same spirit, the work on L -functions computations (extending the computation range, algorithmic tools for computing algebraic data from the L function) will naturally follow.

On the other hand, another objective is to extend existing results on periods of algebraic curves to general curves and higher dimensional varieties is a general promising direction. This project aims at providing tools for integration on higher homology groups of algebraic curves, ie computing Gauss-Manin connections. It requires good understanding of their topology, and more algorithmic tools on differential equations.

3.3. Topology in small dimension

3.3.1. Character varieties

The brute force approach to computable objects from topology of small dimension will not allow any significant progress. As explained above, the systems that arise from these problems are simply outside the range of doable computations. We still continue the work in this direction by a four-fold approach, with all three directions deeply inter-related. First, we focus on a couple of especially meaningful (for the applications) cases, in particular the 3-dimensional manifold called Whitehead link complement. At this point, we are able to make steps in the computation and describe part of the solutions [79], [89]; we hope to be able to complete the computation using every piece of information to simplify the system. Second, we continue the theoretical work to understand more properties of these systems [77]. These properties may prove how useful for the mathematical understanding is the resolution of such systems - or at least the extraction of meaningful information. This approach is for example carried on by Falbel and his work on configuration of flags [80], [82]. Third, we position ourselves as experts in the know-how of this kind of computations and natural interlocutors for colleagues coming up with a question on such a computable object (see [87] and [89]). This also allows us to push forward the kind of computation we actually do and make progress in the direction of the second point. We are credible interlocutors because our team has the blend of theoretical knowledge and computational capabilities that grants effective resolutions of the problems we are presented. And last, we use the knowledge already acquired to pursue our theoretical study of the CR-spherical geometry [69], [81], [78].

Another direction of work is the help to the community in experimental mathematics on new objects. It involves downsizing the system we are looking at (for example by going back to systems coming from hyperbolic geometry and not CR-spherical geometry) and get the most out of what we can compute, by studying new objects. An example of this research direction is the work of Guilloux around the volume function on deformation varieties. This is a real-analytic function defined on the varieties we specialized in computing. Being able to do effective computations with this function led first to a conjecture [86]. Then, theoretical discussions around this conjecture led to a paper on a new approach to the Mahler measure of some 2-variables polynomials [88]. In turn, this last paper gave a formula for the Mahler measure in terms of a function akin to the volume function applied at points in an algebraic variety whose moduli of coordinates are 1. The OURAGAN team has the expertise to compute all the objects appearing in this formula, opening the way to another area of application. This area is deeply linked with number theory as well as topology of small dimension. It requires all the tools at disposition within OURAGAN.

3.3.2. Knot theory

We will carry on the exhaustive search for the lexicographic degrees for the rational knots. They correspond to trigonal space curves: computations in the braid group B_3 , explicit parametrization of trigonal curves corresponding to "dessins d'enfants", etc. The problem seems much more harder when looking for more general knots.

On the other hand, a natural direction would be: given an explicit polynomial space curve, determine the under/over nature of the crossings when projecting, draw it and determine the known knot⁰ it is isotopic to.

3.3.3. Vizualisation and Computational Geometry

As mentioned above, the drawing of algebraic curves and surfaces is a critical action in OURAGAN since it is a key ingredient in numerous developments. In some cases, one will need a fully certified study of the variety for deciding existence of solutions (for example a region in a robot's parameter's space with solutions

⁰for example the first rational knots are listed at <https://team.inria.fr/ouragan/knots>

to the DKP above or deciding if some variety crosses the unit polydisk for some stability problems in control-theory), in some other cases just a partial but certified approximation of a surface (path planning in robotics, evaluation of non algebraic functions over an algebraic variety for volumes of knot complements in the study of character varieties).

On the one hand, we will contribute to general tools like ISOTOP⁰ under the supervision of the GAMBLE project-team and, on the other hand, we will propose ad-hoc solutions by gluing some of our basic tools (problems of high degrees in robust control theory). The priority is to provide a first software that implements methods that fit as most as possible the very last complexity results we got on several (theoretical) algorithms for the computation of the topology of plane curves.

A particular effort will be devoted to the resolution of overconstraint bivariate systems which are useful for the studies of singular points and to polynomials systems in 3 variables in the same spirit : avoid the use of Gröbner basis and propose a new algorithm with a state-of-the-art complexity and with a good practical behavior.

In parallel, one will have to carefully study the drawing of graphs of non algebraic functions over algebraic complex surfaces for providing several tools which are useful for mathematicians working on topology in small dimension (a well known example is the drawing of amoebias, a way of representing a complex curve on a sheet of paper).

3.4. Algebraic analysis of functional systems

We want to further develop our expertise in the computational aspects of algebraic analysis by continuing to develop effective versions of results of module theory, homological algebra, category theory and sheaf theory [136] which play important roles in algebraic analysis [45], [103], [104] and in the algorithmic study of linear functional systems. In particular, we shall focus on linear systems of integro-differential-constant/varying/distributed delay equations [124], [126] which play an important role in mathematical systems theory, control theory, and signal processing [124], [131], [125], [128].

The rings of integro-differential operators are highly more complicated than the purely differential case (i.e. Weyl algebras) [12], due to the existence of zero-divisors, or the fact of having a coherent ring instead of a noetherian ring [42]. Therefore, we want to develop an algorithmic study of these rings. Following the direction initiated in [126] for the computation of zero divisors (based on the polynomial null spaces of certain operators), we first want to develop algorithms for the computation of left/right kernels and left/right/generalized inverses of matrices with entries in such rings, and to use these results in module theory (e.g. computation of syzygy modules, (shorter/shortest) free resolutions, split short/long exact sequences). Moreover, Stafford's results [137], algorithmically developed in [12] for rings of partial differential operators (i.e. the Weyl algebras), are known to still hold for rings of integro-differential operators. We shall study their algorithmic extensions. Our corresponding implementation will be extended accordingly.

Finally, within a computer algebra viewpoint, we shall continue to algorithmically study issues on rings of integro-differential-delay operators [124], [125] and their applications to the study of equivalences of differential constant/varying/distributed delay systems (e.g. Artstein's reduction, Fiagbedzi-Pearson's transformation) which play an important role in control theory.

⁰<https://isotop.gamble.loria.fr>

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 building blocks for higher level algorithms that 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: Jérémy Berthomieu, Jean-Charles Faugère, Mohab Safey El Din, Elias Tsigaridas, Dongming Wang, Matías Bender, Thi Xuan Vu.

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.

⁰J.-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, Elias Tsigaridas, Daniel Lazard, Thi Xuan Vu.

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, Mohab Safey El Din, Elias Tsigaridas, Olive Chakraborty, Jocelyn Ryckeghem.

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. The FGB library is an efficient one 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 LINBOX⁰ 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

⁰<http://www.linalg.org/>

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 domain⁰. 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 fields⁰ 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: Jérémy Berthomieu, Jean-Charles Faugère, Ludovic Perret, Olive Chakraborty, Nagardjun Chinthamani, Solane El Hirsch, Jocelyn Ryckeghem.

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*.

⁰ 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

⁰ e.g. point counting, discrete logarithm, isogeny.

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 cryptosystems. 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 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 ciphers⁰ or 57 new authenticated-encryption schemes⁰. 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 1994⁰ when Shor showed that a quantum computer could solve them efficiently. Though large enough quantum computers that would be able to threaten the

⁰35 are described on https://www.cryptolux.org/index.php/Lightweight_Block_Ciphers.

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

⁰P. 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 primitives⁰ has been launched by the NIST, 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 information-theoretic 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).

⁰<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 [18] 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 [70], 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 [70] 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 [71]. 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 [59], [67], [35], [33], [34], [54]. The past ÉPI Algorithms contributed several implementations (*gfun* [62], *Mgfun* [35]).

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* [58]) 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 [32]. All this is based on subtle algorithms for Hermite–Padé approximants [22]. Moreover, guessing can at times be complemented by proven quantitative results that turn the heuristics into an algorithm [30]. 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 [41], 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. [27].

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 [18], [20], [63] to websites and wikis⁰ 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 Site⁰ or DDMF⁰. 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 content⁰. As of today, there is no encyclopedia certified with formal proofs.

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

⁰<http://functions.wolfram.com/>

⁰<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 [66], [46], [55] 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 [38]. A discussion about the systematic separation of the search for a solution and the checking of the solution is already clearly outlined in [52].

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.

⁰<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 [56], which is based on set theory. In particular, the calculus of construction (CoC) [36] and its extension with inductive types (CIC) [37], have been studied for more than 20 years and been implemented by several independent tools (like Lego, Matita, and Agda). Its reference implementation, Coq [64], 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 [40] 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 [43] 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 [65], 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 [48], [49], [50], although some existing developments of interest [25], [57] 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 [45], [44], 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 [61]. For the Coq system, this is an active area of research.

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.

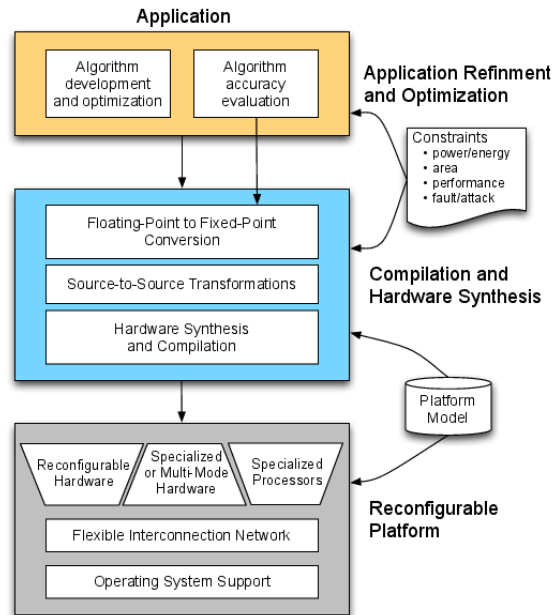


Figure 1. CAIRN's general design flow and related research themes

Figure 1 shows the global design flow we propose to develop. This flow is organized in levels corresponding 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 [69]. They can also be reconfigured dynamically. 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 related work has been 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 [53] or implementing an online filter to relocate the tasks [63]. 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 [70], [58], [68]. 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[9], Adres [66] or polymorphous computing fabrics[11]. These works have led to commercial products such as the PACT/XPP [52] 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 [56], [71], [67], [65], [64]. 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 [62]. Here again, existing tools are not well suited, as they fail at providing a 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) [5], ASIP optimizing compilers [12] 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 [6]. 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 [4].

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 [7] 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 [59]. 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 [60]. 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 [10].

CAMUS Project-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 [56]. Performance, correctness 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

The development of efficient and correct applications for multicore processors requires 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 performed, resulting in *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 actual 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, the complexity of current and future architectures avoids assuming an optimal behavior regarding a given program version. A monitoring process will make it possible to select on-the-fly the best parallelization.

These different parallelization steps are schematized in figure 1 .

Our project relies 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 correctness 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. This must be proved formally (issue 4).

In the following, those different issues are detailed while forming our global, 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, Bénénger Bramas, Harenome Ranaivoarivony-Razanajato.

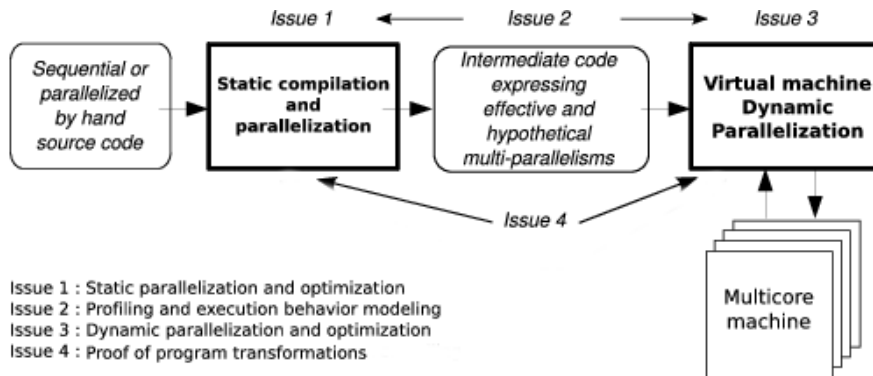


Figure 1. Steps for Automatic parallelization on multicore architectures.

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 [53]. Low-level optimizations, in the assembly code generated by the compiler, have also been extensively dealt with 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 architectures and expressing many potential parallelisms.

3.3. Profiling and Execution Behavior Modeling

Participants: Alain Ketterlin, Philippe Clauss, Salwa Kobeissi.

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: Philippe Clauss, Salwa Kobeissi, Jens Gustedt, Alain Ketterlin, Muthena Abdul-Wahab, Daniel Salas, Bérenger Bramas.

Dynamic parallelization and optimization has become essential with the advent of the new multicore architectures. When using a dynamic scheme, the 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 scheme 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 provides a significant optimization ability compared to a static compiler, while observing the evolution of the availability of live resources.

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 optimization to produce data-race free code. For the second stage of optimization we will first assume that the input code is data-race free. We will prove those transformations using Appel's concurrent separation logic [57]. Proving transformations involving programs which are not data-race free will constitute a longer term research goal.

CASH Project-Team

3. Research Program

3.1. Definition of dataflow representations of parallel programs

In the last decades, several frameworks have emerged to design efficient compiler algorithms. The efficiency of all the optimizations performed in compilers strongly relies on effective *static analyses* and *intermediate representations*. Dataflow models are a natural intermediate representation for hardware compilers (HLS) and more generally for parallelizing compilers. Indeed, dataflow models capture task-level parallelism and can be mapped naturally to parallel architectures. In a way, a dataflow model is a partition of the computation into processes and a partition of the flow dependences into channels. This partitioning prepares resource allocation (which processor/hardware to use) and medium-grain communications.

The main goal of the CASH team is to provide efficient analyses and the optimizing compilation frameworks for dataflow programming models. The results of the team relies on programming languages and representation of programs in which parallelism and dataflow play a crucial role. This first research direction aims at defining these dataflow languages and intermediate representations, both from a practical perspective (syntax or structure), and from a theoretical point of view (semantics). This first research direction thus defines the models on which the other directions will rely. It is important to note that we do not restrict ourselves to a strict definition of dataflow languages: more generally, we are interested in the parallel languages in which dataflow synchronization plays a significant role.

Intermediate dataflow model. The intermediate dataflow model is a representation of the program that is adapted for optimization and scheduling. It will be obtained from the analysis of a (parallel or sequential) program and should at some point be used for compilation. The dataflow model must specify precisely its semantics and parallelism granularity. It must also be analyzable with polyhedral techniques, where powerful concepts exist to design compiler analysis, e.g., scheduling or resource allocation. Polyhedral Process Networks [58] extended with a module system could be a good starting point. But then, how to fit non-polyhedral parts of the program? A solution is to hide non-polyhedral parts into processes with a proper polyhedral abstraction. This organization between polyhedral and non-polyhedral processes will be a key aspect of our medium-grain dataflow model. The design of our intermediate dataflow model and the precise definition of its semantics will constitute a reliable basis to formally define and ensure the correctness of algorithms proposed by CASH: compilation, optimizations and analyses.

Dataflow programming languages. Dataflow paradigm has also been explored quite intensively in programming languages. Indeed, there exists a large panel of dataflow languages, whose characteristics differ notably, the major point of variability being the scheduling of agents and their communications. There is indeed a continuum from the synchronous dataflow languages like Lustre [42] or Streamit [55], where the scheduling is fully static, and general communicating networks like KPNs [45] or RVC-Cal [25] where a dedicated runtime is responsible for scheduling tasks dynamically, when they *can* be executed. These languages share some similarities with actor languages that go even further in the decoupling of processes by considering them as independent reactive entities. Another objective of the CASH team is to study dataflow programming *languages*, their semantics, their expressiveness, and their compilation. The specificity of the CASH team is that these languages will be designed taking into consideration the compilation using polyhedral techniques. In particular, we will explore which dataflow constructs are better adapted for our static analysis, compilation, and scheduling techniques. In practice we want to propose high-level primitives to express data dependency, this way the programmer can express parallelism in a dataflow way instead of the classical communication-oriented dependencies. The higher-level more declarative point of view makes programming easier but also give more optimization opportunities. These primitives will be inspired by the existing works in the polyhedral model framework, as well as dataflow languages, but also in the actors and active object languages [32] that nowadays introduce more and more dataflow primitives to enable data-driven interactions between agents, particularly with *futures* [30], [37].

3.1.1. Expected Impact

Consequently, the impact of this research direction is both the usability of our representation for static analyses and optimizations performed in Sections 3.2 and 3.3, and the usability of its semantics to prove the correctness of these analyses.

3.1.2. Scientific Program

3.1.2.1. Short-term and ongoing activities.

We obtained preliminary experimental [22], [23], [38] and theoretical [43] results, exploring several aspects of dataflow models. The next step is to define accurately the intermediate dataflow model and to study existing programming and execution models:

- Define our medium-grain dataflow model. So far, a modular Polyhedral Process Networks appears as a natural candidate but it may need to be extended to be adapted to a wider range of applications. Precise semantics will have to be defined for this model to ensure the articulation with the activities discussed in Section 3.3.
- Study precisely existing dataflow languages, their semantics, their programmability, and their limitations.

3.1.2.2. Medium-term activities.

In a second step, we will extend the existing results to widen the expressiveness of our intermediate representation and design new parallelism constructs. We will also work on the semantics of dataflow languages:

- Propose new stream programming models and a clean semantics where all kinds of parallelisms are expressed explicitly, and where all activities from code design to compilation and scheduling can be clearly expressed.
- Identify a core language that is rich enough to be representative of the dataflow languages we are interested in, but abstract and small enough to enable formal reasoning and proofs of correctness for our analyses and optimizations.

3.1.2.3. Long-term activities.

In a longer-term vision, the work on semantics, while remaining driven by the applications, would lead to more mature results, for instance:

- Design more expressive dataflow languages and intermediate representations which would at the same time be expressive enough to capture all the features we want for aggressive HPC optimizations, and sufficiently restrictive to be (at least partially) statically analyzable at a reasonable cost.
- Define a module system for our medium-grain dataflow language. A program will then be divided into modules that can follow different compilation schemes and execution models but still communicate together. This will allow us to encapsulate a program that does not fit the polyhedral model into a polyhedral one and vice versa. Also, this will allow a compositional analysis and compilation, as opposed to global analysis which is limited in scalability.

3.2. Expressivity and Scalability of Static Analyses

The design and implementation of efficient compilers becomes more difficult each day, as they need to bridge the gap between *complex languages* and *complex architectures*. Application developers use languages that bring them close to the problem that they need to solve which explains the importance of high-level programming languages. However, high-level programming languages tend to become more distant from the hardware which they are meant to command.

In this research direction, we propose to design expressive and scalable static analyses for compilers. This topic is closely linked to Sections 3.1 and 3.3 since the design of an efficient intermediate representation is made while regarding the analyses it enables. The intermediate representation should be expressive enough to embed maximal information; however if the representation is too complex the design of scalable analyses will be harder.

The analyses we plan to design in this activity will of course be mainly driven by the HPC dataflow optimizations we mentioned in the preceding sections; however we will also target other kinds of analyses applicable to more general purpose programs. We will thus consider two main directions:

- Extend the applicability of the polyhedral model, in order to deal with HPC applications that do not fit totally in this category. More specifically, we plan to work on more complex control and also on complex data structures, like sparse matrices, which are heavily used in HPC.
- Design of specialized static analyses for memory diagnostic and optimization inside general purpose compilers.

For both activities, we plan to cross fertilize ideas coming from the abstract interpretation community as well as language design, dataflow semantics, and WCET estimation techniques.

Correct by construction analyses. The design of well-defined semantics for the chosen programming language and intermediate representation will allow us to show the correctness of our analyses. The precise study of the semantics of Section 3.1 will allow us to adapt the analysis to the characteristics of the language, and prove that such an adaptation is well founded. This approach will be applicable both on the source language and on the intermediate representation.

Such wellfoundedness criteria relatively to the language semantics will first be used to design our analyses, and then to study which extensions of the languages can be envisioned and analyzed safely, and which extensions (if any) are difficult to analyze and should be avoided. Here the correct identification of a core language for our formal studies (see Section 3.1) will play a crucial role as the core language should feature all the characteristics that might make the analysis difficult or incorrect.

Scalable abstract domains. We already have experience in designing low-cost semi relational abstract domains for pointers [50], [46], as well as tailoring static analyses for specialized applications in compilation [36], [54], Synchronous Dataflow scheduling [53], and extending the polyhedral model to irregular applications [21]. We also have experience in the design of various static verification techniques adapted to different programming paradigms.

3.2.1. Expected impact

The impact of this work is the significantly widened applicability of various tools/compilers related to parallelization: allow optimizations for a larger class of programs, and allow low-cost analysis that scale to very large programs.

We target both analysis for optimization and analysis to detect, or prove the absence of bugs.

3.2.2. Scientific Program

3.2.2.1. Short-term and ongoing activities.

Together with Paul Iannetta and Lionel Morel (INSA/CEA LETI), we are currently working on the *semantic rephrasing* of the polyhedral model [39]. The objective is to clearly redefine the key notions of the polyhedral model on general flowchart programs operating on arrays, lists and trees. We reformulate the algorithms that are performed to compute dependencies in a more semantic fashion, i.e. considering the program semantics instead of syntactical criteria. The next step is to express classical scheduling and code generation activities in this framework, in order to overcome the classical syntactic restrictions of the polyhedral model.

3.2.2.2. Medium-term activities.

In medium term, we want to extend the polyhedral model for more general data-structures like lists and sparse matrices. For that purpose, we need to find polyhedral (or other shapes) abstractions for non-array data-structures; the main difficulty is to deal with non-linearity and/or partial information (namely, over-approximations of the data layout, or over-approximation of the program behavior). This activity will rely on a formalization of the optimization activities (dependency computation, scheduling, compilation) in a more general Abstract-Interpretation based framework in order to make the approximations explicit.

At the same time, we plan to continue to work on scaling static analyses for general purpose programs, in the spirit of Maroua Maalej’s PhD [47], whose contribution is a sequence of memory analyses inside production compilers. We already began a collaboration with Sylvain Collange (PACAP team of IRISA Laboratory) on the design of static analyses to optimize copies from the global memory of a GPU to the block kernels (to increase locality). In particular, we have the objective to design specialized analyses but with an explicit notion of cost/precision compromise, in the spirit of the paper [41] that tries to formalize the cost/precision compromise of interprocedural analyses with respect to a “context sensitivity parameter”.

3.2.2.3. Long-term activities.

In a longer-term vision, the work on scalable static analyses, whether or not directed from the dataflow activities, will be pursued in the direction of large general-purpose programs.

An ambitious challenge is to find a generic way of adapting existing (relational) abstract domains within the Single Static Information [26] framework so as to improve their scalability. With this framework, we would be able to design static analyses, in the spirit of the seminal paper [31] which gave a theoretical scheme for classical abstract interpretation analyses.

We also plan to work on the interface between the analyses and their optimization clients inside production compilers.

3.3. Compiling and Scheduling Dataflow Programs

In this part, we propose to design the compiler analyses and optimizations for the *medium-grain* dataflow model defined in section 3.1 . We also propose to exploit these techniques to improve the compilation of dataflow languages based on actors. Hence our activity is split into the following parts:

- Translating a sequential program into a medium-grain dataflow model. The programmer cannot be expected to rewrite the legacy HPC code, which is usually relatively large. Hence, compiler techniques must be invented to do the translation.
- Transforming and scheduling our medium-grain dataflow model to meet some classic optimization criteria, such as throughput, local memory requirements, or I/O traffic.
- Combining agents and polyhedral kernels in dataflow languages. We propose to apply the techniques above to optimize the processes in actor-based dataflow languages and combine them with the parallelism existing in the languages.

We plan to rely extensively on the polyhedral model to define our compiler analysis. The polyhedral model was originally designed to analyze imperative programs. Analysis (such as scheduling or buffer allocation) must be redefined in light of dataflow semantics.

Translating a sequential program into a medium-grain dataflow model. The programs considered are compute-intensive parts from HPC applications, typically big HPC kernels of several hundreds of lines of C code. In particular, we expect to analyze the process code (actors) from the dataflow programs. On short ACL (Affine Control Loop) programs, direct solutions exist [56] and rely directly on array dataflow analysis [35]. On bigger ACL programs, this analysis no longer scales. We plan to address this issue by *modularizing* array dataflow analysis. Indeed, by splitting the program into processes, the complexity is mechanically reduced. This is a general observation, which was exploited in the past to compute schedules [34]. When the program is no longer ACL, a clear distinction must be made between polyhedral parts and non polyhedral parts. Hence, our medium-grain dataflow language must distinguish between polyhedral process networks, and non-polyhedral code fragments. This structure raises new challenges: How to abstract away non-polyhedral parts while keeping the polyhedrality of the dataflow program? Which trade-off(s) between precision and scalability are effective?

Medium-grain data transfers minimization. When the system consists of a single computing unit connected to a slow memory, the roofline model [59] defines the optimal ratio of computation per data transfer (*operational intensity*). The operational intensity is then translated to a partition of the computation (loop tiling) into *reuse units*: inside a reuse unit, data are transferred locally; between reuse units, data are transferred through the slow memory. On a *fine-grain* dataflow model, reuse units are exposed with loop tiling; this is the case for example

in Data-aware Process Network (DPN) [23]. The following questions are however still open: How does that translate on *medium-grain* dataflow models? And fundamentally what does it mean to *tile* a dataflow model?

Combining agents and polyhedral kernels in dataflow languages. In addition to the approach developed above, we propose to explore the compilation of dataflow programming languages. In fact, among the applications targeted by the project, some of them are already thought or specified as dataflow actors (video compression, machine-learning algorithms,...).

So far, parallelization techniques for such applications have focused on taking advantage of the decomposition into agents, potentially duplicating some agents to have several instances that work on different data items in parallel [40]. In the presence of big agents, the programmer is left with the splitting (or merging) of these agents by-hand if she wants to further parallelize her program (or at least give this opportunity to the runtime, which in general only sees agents as non-malleable entities). In the presence of arrays and loop-nests, or, more generally, some kind of regularity in the agent's code, however, we believe that the programmer would benefit from automatic parallelization techniques such as those proposed in the previous paragraphs. To achieve the goal of a totally integrated approach where programmers write the applications they have in mind (application flow in agents where the agents' code express potential parallelism), and then it is up to the system (compiler, runtime) to propose adequate optimizations, we propose to build on solid formal definition of the language semantics (thus the formal specification of parallelism occurring at the agent level) to provide hierarchical solutions to the problem of compilation and scheduling of such applications.

3.3.1. Expected impact

In general, splitting a program into simpler processes simplifies the problem. This observation leads to the following points:

- By abstracting away irregular parts in processes, we expect to structure the long-term problem of handling irregular applications in the polyhedral model. The long-term impact is to widen the applicability of the polyhedral model to irregular kernels.
- Splitting a program into processes reduces the problem size. Hence, it becomes possible to scale traditionally expensive polyhedral analysis such as scheduling or tiling to quote a few.

As for the third research direction, the short term impact is the possibility to combine efficiently classical dataflow programming with compiler polyhedral-based optimizations. We will first propose ad-hoc solutions coming from our HPC application expertise, but supported by strong theoretical results that prove their correctness and their applicability in practice. In the longer term, our work will allow specifying, designing, analyzing, and compiling HPC dataflow applications in a unified way. We target semi-automatic approaches where pertinent feedback is given to the developer during the development process.

3.3.2. Scientific Program

3.3.2.1. Short-term and ongoing activities.

We are currently working on the RTM (Reverse-Time Migration) kernel for oil and gas applications (≈ 500 lines of C code). This kernel is long enough to be a good starting point, and small enough to be handled by a polyhedral splitting algorithm. We figured out the possible splittings so the polyhedral analysis can scale and irregular parts can be hidden. In a first step, we plan to define splitting metrics and algorithms to optimize the usual criteria: communication volume, latency and throughput.

Together with Lionel Morel (INSA/CEA LETI), we currently work on the evaluation of the practical advantage of combining the dataflow paradigm with the polyhedral optimization framework. We empirically build a proof-of-concept tooling approach, using existing tools on existing languages [38]. We combine dataflow programming with polyhedral compilation in order to enhance program parallelization by leveraging both inter-agent parallelism and intra-agent parallelism (i.e., regarding loop nests inside agents). We evaluate the approach practically, on benchmarks coming from image transformation or neural networks, and the first results demonstrate that there is indeed a room for further improvement.

3.3.2.2. Medium-term activities.

The results of the preceding paragraph are partial and have been obtained with a simple experimental approach only using off-the-shelf tools. We are thus encouraged to pursue research on combining expertise from dataflow programming languages and polyhedral compilation. Our long term objective is to go towards a formal framework to express, compile, and run dataflow applications with intrinsic instruction or pipeline parallelism.

We plan to investigate in the following directions:

- Investigate how polyhedral analysis extends on modular dataflow programs. For instance, how to modularize polyhedral scheduling analysis on our dataflow programs?
- Develop a proof of concept and validate it on linear algebra kernels (SVD, Gram-Schmidt, etc.).
- Explore various areas of applications from classical dataflow examples, like radio and video processing, to more recent applications in deep learning algorithmic. This will enable us to identify some potential (intra and extra) agent optimization patterns that could be leveraged into new language idioms.

3.3.2.3. Long-term activities.

Current work focus on purely polyhedral applications. Irregular parts are not handled. Also, a notion of tiling is required so the communications of the dataflow program with the outside world can be tuned with respect to the local memory size. Hence, we plan to investigate the following points:

- Assess simple polyhedral/non polyhedral partitioning: How non-polyhedral parts can be hidden in processes/channels? How to abstract the dataflow dependencies between processes? What would be the impact on analyses? We target programs with irregular control (e.g., while loop, early exits) and regular data (arrays with affine accesses).
- Design tiling schemes for modular dataflow programs: What does it mean to tile a dataflow program? Which compiler algorithms to use?
- Implement a mature compiler infrastructure from the front-end to code generation for a reasonable subset of the representation.

3.4. HLS-specific Dataflow Optimizations

The compiler analyses proposed in section 3.3 do not target a specific platform. In this part, we propose to leverage these analysis to develop source-level optimizations for high-level synthesis (HLS).

High-level synthesis consists in compiling a kernel written in a high-level language (typically in C) into a circuit. As for any compiler, an HLS tool consists in a *front-end* which translates the input kernel into an *intermediate representation*. This intermediate representation captures the control/flow dependences between computation units, generally in a hierarchical fashion. Then, the *back-end* maps this intermediate representation to a circuit (e.g. FPGA configuration). We believe that HLS tools must be thought as fine-grain automatic parallelizers. In classic HLS tools, the parallelism is expressed and exploited at the back-end level during the scheduling and the resource allocation of arithmetic operations. We believe that it would be far more profitable to derive the parallelism at the front-end level.

Hence, CASH will focus on the *front-end* pass and the *intermediate representation*. Low-level *back-end* techniques are not in the scope of CASH. Specifically, CASH will leverage the dataflow representation developed in Section 3.1 and the compilation techniques developed in Section 3.3 to develop a relevant intermediate representation for HLS and the corresponding front-end compilation algorithms.

Our results will be evaluated by using existing HLS tools (e.g., Intel HLS compiler, Xilinx Vivado HLS). We will implement our compiler as a source-to-source transformation in front of HLS tools. With this approach, HLS tools are considered as a “back-end black box”. The CASH scheme is thus: (i) *front-end*: produce the CASH dataflow representation from the input C kernel. Then, (ii) turn this dataflow representation to a C program with pragmas for an HLS tool. This step must convey the characteristics of the dataflow representation found by step (i) (e.g. dataflow execution, fifo synchronisation, channel size). This source-to-source approach will allow us to get a full source-to-FPGA flow demonstrating the benefits of our tools while relying on existing tools for low-level optimizations. Step (i) will start from the DCC tool developed by Christophe Alias, which already produces a dataflow intermediate representation: the Data-aware Process Networks (DPN) [23]. Hence, the very first step is then to choose an HLS tool and to investigate which input should be fed to the HLS tool so it “respects” the parallelism and the resource allocation suggested by the DPN. From this basis, we plan to investigate the points described thereafter.

Roofline model and dataflow-level resource evaluation. Operational intensity must be tuned according to the roofline model. The roofline model [59] must be redefined in light of FPGA constraints. Indeed, the peak performance is no longer constant: it depends on the operational intensity itself. The more operational intensity we need, the more local memory we use, the less parallelization we get (since FPGA resources are limited), and finally the less performance we get! Hence, multiple iterations may be needed before reaching an efficient implementation. To accelerate the design process, we propose to iterate at the dataflow program level, which implies a fast resource evaluation at the dataflow level.

Reducing FPGA resources. Each parallel unit must use as little resources as possible to maximize parallel duplication, hence the final performance. This requires to factorize the control and the channels. Both can be achieved with source-to-source optimizations at dataflow level. The main issue with outputs from polyhedral optimization is large piecewise affine functions that require a wide silicon surface on the FPGA to be computed. Actually we do not need to compute a closed form (expression that can be evaluated in bounded time on the FPGA) *statically*. We believe that the circuit can be compacted if we allow control parts to be evaluated dynamically. Finally, though dataflow architectures are a natural candidate, adjustments are required to fit FPGA constraints (2D circuit, few memory blocks). Ideas from systolic arrays [52] can be borrowed to re-use the same piece of data multiple times, despite the limitation to regular kernels and the lack of I/O flexibility. A trade-off must be found between pure dataflow and systolic communications.

Improving circuit throughput. Since we target streaming applications, the throughput must be optimized. To achieve such an optimization, we need to address the following questions. How to derive an optimal upper bound on the throughput for polyhedral process network? Which dataflow transformations should be performed to reach it? The limiting factors are well known: I/O (decoding of burst data), communications through addressable channels, and latencies of the arithmetic operators. Finally, it is also necessary to find the right methodology to measure the throughput statically and/or dynamically.

3.4.1. Expected Impact

So far, the HLS front-end applies basic loop optimizations (unrolling, flattening, pipelining, etc.) and use a Hierarchical Control Flow Graph-like representation with data dependencies annotations (HCDFG). With this approach, we intend to demonstrate that polyhedral analysis combined with dataflow representations is an effective solution for HLS tools.

3.4.2. Scientific Program

3.4.2.1. Short-term and ongoing activities.

The HLS compiler designed in the CASH team currently extracts a fine-grain parallel intermediate representation (DPN [23], [22]) from a sequential program. We will not write a back-end that produces code for FPGA but we need to provide C programs that can be fed into existing C-to-FPGA compilers. However we obviously need an end-to-end compiler for our experiments. One of the first task of our HLS activity is to develop a DPN-to-C code generator suitable as input to an existing HLS tool like Vivado HLS. The generated code should exhibit the parallelism extracted by our compiler, and allow generating a final circuit more efficient than

the one that would be generated by our target HLS tool if ran directly on the input program. Source-to-source approaches have already been experimented successfully, e.g. in Alexandru Plesco's PhD [51].

3.4.2.2. *Medium-term activities.*

Our DPN-to-C code generation will need to be improved in many directions. The first point is the elimination of redundancies induced by the DPN model itself: buffers are duplicated to allow parallel reads, processes are produced from statements in the same loop, hence with the same control automaton. Also, multiplexing uses affine constraints which can be factorized [24]. We plan to study how these constructs can be factorized at C-level and to design the appropriate DPN-to-C translation algorithms.

Also, we plan to explore how on-the-fly evaluation can reduce the complexity of the control. A good starting point is the control required for the load process (which fetch data from the distant memory). If we want to avoid multiple load of the same data, the FSM (Finite State Machine) that describes it is usually very complex. We believe that dynamic construction of the load set (set of data to load from the main memory) will use less silicon than an FSM with large piecewise affine functions computed statically.

3.4.2.3. *Long-term activities.*

The DPN-to-C compiler opens new research perspectives. We will explore the roofline model accuracy for different applications by playing on DPN parameters (tile size). Unlike the classical roofline model, the peak performance is no longer assumed to be constant, but decreasing with operational intensity [60]. Hence, we expect a *unique* optimal set of parameters. Thus, we need to build a DPN-level cost model to derive an interval containing the optimal parameters.

Also, we want to develop DPN-level analysis and transformation to quantify the optimal reachable throughput and to reach it. We expect the parallelism to increase the throughput, but in turn it may require an operational intensity beyond the optimal point discussed in the first paragraph. We will assess the trade-offs, build the cost-models, and the relevant dataflow transformations.

3.5. Simulation of Hardware

Complex systems such as systems-on-a-chip or HPC computer with FPGA accelerator comprise both hardware and software parts, tightly coupled together. In particular, the software cannot be executed without the hardware, or at least a simulator of the hardware.

Because of the increasing complexity of both software and hardware, traditional simulation techniques (Register Transfer Level, RTL) are too slow to allow full system simulation in reasonable time. New techniques such as Transaction Level Modeling (TLM) [20] in SystemC [19] have been introduced and widely adopted in the industry. Internally, SystemC uses discrete-event simulation, with efficient context-switch using cooperative scheduling. TLM abstracts away communication details, and allows modules to communicate using function calls. We are particularly interested in the loosely timed coding style where the timing of the platform is not modeled precisely, and which allows the fastest simulations. This allowed gaining several orders of magnitude of simulation speed. However, SystemC/TLM is also reaching its limits in terms of performance, in particular due to its lack of parallelism.

Work on SystemC/TLM parallel execution is both an application of other work on parallelism in the team and a tool complementary to HLS presented in Sections 3.1 (dataflow models and programs) and 3.4 (application to FPGA). Indeed, some of the parallelization techniques we develop in CASH could apply to SystemC/TLM programs. Conversely, a complete design-flow based on HLS needs fast system-level simulation: the full-system usually contains both hardware parts designed using HLS, handwritten hardware components, and software.

We also work on simulation of the DPN intermediate representation. Simulation is a very important tool to help validate and debug a complete compiler chain. Without simulation, validating the front-end of the compiler requires running the full back-end and checking the generated circuit. Simulation can avoid the execution time of the backend and provide better debugging tools.

Automatic parallelization has shown to be hard, if at all possible, on loosely timed models [28]. We focus on semi-automatic approaches where the programmer only needs to make minor modifications of programs to get significant speedups.

3.5.1. *Expected Impact*

The short term impact is the possibility to improve simulation speed with a reasonable additional programming effort. The amount of additional programming effort will thus be evaluated in the short term.

In the longer term, our work will allow scaling up simulations both in terms of models and execution platforms. Models are needed not only for individual Systems on a Chip, but also for sets of systems communicating together (e.g., the full model for a car which comprises several systems communicating together), and/or heterogeneous models. In terms of execution platform, we are studying both parallel and distributed simulations.

3.5.2. *Scientific Program*

3.5.2.1. *Short-term and ongoing activities.*

We started the joint PhD (with Tanguy Sassolas) of Gabriel Busnot with CEA-LIST. The research targets parallelizing SystemC heterogeneous simulations. CEA-LIST already developed SScale [57], which is very efficient to simulate parallel homogeneous platforms such as multi-core chips. However, SScale cannot currently load-balance properly the computations when the platform contains different components modeled at various levels of abstraction. Also, SScale requires manual annotations to identify accesses to shared variables. These annotations are given as address ranges in the case of a shared memory. This annotation scheme does not work when the software does non-trivial memory management (virtual memory using a memory management unit, dynamic allocation), since the address ranges cannot be known statically. We started working on the “heterogeneous” aspect of simulations with an approach allowing changing the level of details in a simulation at runtime, and started tackling the virtual and dynamic memory management problem by porting Linux on our simulation platform.

We also started working on an improved support for simulation and debugging of the DPN internal representation of our parallelizing compiler (see Section 3.3). A previous quick experiment with simulation was to generate C code that simulates parallelism with POSIX-threads. While this simulator greatly helped debug the compiler, this is limited in several ways: simulations are not deterministic, and the simulator does not scale up since it would create a very large number of threads for a non-trivial design.

We are working in two directions. The first is to provide user-friendly tools to allow graphical inspection of traces. For example, we started working on the visualization of the sequence of steps leading to a deadlock when the situation occurs, and will give hints on how to fix the problem in the compiler. The second is to use an efficient simulator to speed up the simulation. We plan to generate SystemC/TLM code from the DPN representation to benefit from the ability of SystemC to simulate a large number of processes.

3.5.2.2. *Medium-term activities.*

Several research teams have proposed different approaches to deal with parallelism and heterogeneity. Each approach targets a specific abstraction level and coding style. While we do not hope for a universal solution, we believe that a better coordination of different actors of the domain could lead to a better integration of solutions. We could imagine, for example, a platform with one subsystem accelerated with SScale [57] from CEA-LIST, some compute-intensive parts delegated to sc-during [49] from Matthieu Moy, and a co-simulation with external physical solvers using SystemC-MDVP [27] from LIP6. We plan to work on the convergence of approaches, ideally both through point-to-point collaborations and with a collaborative project.

A common issue with heterogeneous simulation is the level of abstraction. Physical models only simulate one scenario and require concrete input values, while TLM models are usually abstract and not aware of precise physical values. One option we would like to investigate is a way to deal with loose information, e.g. manipulate intervals of possible values instead of individual, concrete values. This would allow a simulation to be symbolic with respect to the physical values.

Obviously, works on parallel execution of simulations would benefit to simulation of data-aware process networks (DPN). Since DPN are generated, we can even tweak the generator to guarantee some properties on the generated code, which gives us more freedom on the parallelization and partitioning techniques.

3.5.2.3. *Long-term activities.*

In the long term, our vision is a simulation framework that will allow combining several simulators (not necessarily all SystemC-based), and allow running them in a parallel way. The Functional Mockup Interface (FMI) standard is a good basis to build upon, but the standard does not allow expressing timing and functional constraints needed for a full co-simulation to run properly.

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 general purpose but also for domain specific 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 run-time 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, run-time 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); Run-time 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 run-time 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 run-time 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.

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.

Still, 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 members of the team), sequential tasks are still a major performance bottleneck. Addressing it is still on the agenda of the 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 thus confronted with a moving target, challenging portability and jeopardizing performance.

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 (non-Silicon, 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), quantum computing (impacts the entire software stack).

3.1.2. *Evolving community*

The PACAP project-team tackles performance-related issues, for conventional programming paradigms. In fact, programming complex environments is no longer the exclusive domain of 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 a 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.

PACAP works on fundamental technologies for computer science: processor architecture, performance-oriented compilation and guaranteed response time for real-time. The research results may have impact 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 members of the project teams for two decades, with undeniable contributions. They 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. One of our goals is to devise new static compilation techniques (cf. Section 3.2.1), but also build upon iterative [1] and split [40] 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 increased performance.

The end of Moore's law and Dennard's scaling⁰ 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 micro-architecture (Section 3.2.3). Reconciling CPU and GPU designs (Section 3.2.4) is one of our objectives.

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 as power efficiency (Section 3.2.6), and security (Section 3.2.7).

3.2.1. Static Compilation

Static compilation techniques continue to be relevant in addressing 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 studies new optimization opportunities and develops tailored compilation techniques for upcoming compute nodes. New technologies may also be coupled with traditional solutions to offer new

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

trade-offs. We 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 are 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 also leverage split-compilation [40]: 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 needs to adapt to its environment. In most cases, this environment remains 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 transparently [2]. 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 becomes 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 [46] 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 H2020 FET HPC project ANTAREX also addresses these challenges from the energy perspective. We further leverage our platform and initial results to address other adaptation opportunities. Efficient software adaptation requires expertise from all domains tackled by PACAP, and strong interaction between all team members is expected.

3.2.3. Research directions in uniprocessor micro-architecture

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, and value prediction. In particular, in recent years they have been recognized as world leaders in branch prediction [50] [44] and in cache prefetching [6] and they have revived the forgotten concept of value prediction [9][8]. This research was supported by the ERC Advanced grant DAL (2011-2016) and also by Intel. We pursue research on achieving ultimate uniprocessor 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 cores;
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) [6]. The first research objective 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 research objective 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 benefits 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 [11]. It offers new possibilities 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 processors. For the two past decades, implementing ever wider issue superscalar processors has been challenging. The objective of our research on the execution core is to explore (and revisit) directions that 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 are exploring is the use of clustered architectures [7]. Symmetric clustered organization allows to benefit from a simpler bypass network, but induce large complexity on the issue queue. One remarkable finding of our study [7] is that, when considering two large clusters (e.g. 8-wide), steering large groups of consecutive instructions (e.g. 64 μ ops) to the same cluster is quite efficient. This opens opportunities to limit the complexity of the issue queues (monitoring fewer buses) and register files (fewer ports and physical registers) in the clusters, since not all results have to be forwarded to the other cluster.

The second direction that we are exploring is associated with the approach that we developed with Sembrant et al. [47]. 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 micro-architecture with a cluster dedicated to the execution of urgent instructions and a second cluster executing the non-urgent instructions. The micro-architecture 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 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 [48].

In speculative execution, we have recently revisited value prediction (VP) which was a hot research topic between 1996 and 2002. However it was considered until 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 [9][8] [50] [44]. First, very high accuracy can be enforced at reasonable cost in coverage and minimal complexity [9]. 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 [8]. 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 [43] 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 micro-architectures 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 investigate the convergence of CPU and GPU at both architecture and compiler levels.

Architecture.

The architecture convergence between Single Instruction Multiple Threads (SIMT) GPUs and multicore processors that we have been pursuing [42] 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.

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 architecture⁰, or the Recore manycore hardware⁰) 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 analyze and/or control 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. Power efficiency

PACAP addresses power-efficiency at several levels. First, we 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 focus on high-performance low-power embedded compute nodes. Within the ANR project Continuum, in collaboration with architecture and technology experts from LIRMM and the SME Cortus, we research new static and dynamic compilation techniques that fully exploit emerging memory and NoC technologies. Finally, in collaboration with the CAIRN project-team, we 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 relies 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.

⁰<http://www.kalrayinc.com>

⁰<http://www.recoresystems.com/>

3.2.7. Security

Security is a mandatory concern of any modern computing system. Various threat models have led to a multitude of protection solutions. Members of PACAP already contributed in the past, thanks to the HAVEGE [49] random number generator, and code obfuscating techniques (the obfuscating just-in-time compiler [41], or thread-based control flow mangling [45]). Still, security is not core competence of PACAP members.

Our strategy consists in partnering 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 expertise in compilation and architecture helps design more efficient and less expensive protection mechanisms.

Examples of collaborations so far include the following:

Compilation: We partnered with experts in security and codes to prototype a platform that demonstrates resilient software. They designed and proposed advanced masking techniques to hide sensitive data in application memory. PACAP's expertise is 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.

Dynamic Binary Rewriting: Our expertise in dynamic binary rewriting combines well with the expertise of the CIDRE team in protecting application. Security has a high cost in terms of performance, and static insertion of counter measures cannot take into account the current threat level. In collaboration with CIDRE, we propose an adaptive insertion/removal of countermeasures in a running application based of dynamic assessment of the threat level.

WCET Analysis: Designing real-time systems requires computing an upper bound of the worst-case execution time. Knowledge of this timing information opens an opportunity to detect attacks on the control flow of programs. In collaboration with CIDRE, we are developing a technique to detect such attacks thanks to a hardware monitor that makes sure that statically computed time information is preserved (CAIRN is also involved in the definition of the hardware component).

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 reuse⁰. 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 Consortium⁰. A wider set of tools, both industrial and academic, now exists in this segment⁰. In the EDA sector, VHDL-AMS was developed as a standard [12] and also allows for differential algebraic equations. Several domain-specific languages and tools for mechanical systems or electronic circuits also support some restricted classes of differential algebraic equations. Spice is the historic and most striking instance of these domain-specific languages/tools⁰. The main difference is that equations are hidden and the fixed structure of the differential algebraic results from the physical domain covered by these languages.

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 [19], [15] and [16].

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 [15], [19], [17], [16]. 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 [2], a chapter of Simon Bliudze's PhD thesis [25], and a recent presentation of non-standard analysis, not axiomatic in style, due to the mathematician Lindström [49].

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

⁰<https://www.modelica.org/>

⁰SimScape by Mathworks, Amesim by LMS International, now Siemens PLM, and more.

⁰<http://bwrcs.eecs.berkeley.edu/Courses/IcBook/SPICE/MANUALS/spice3.html>

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 ∂ 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 [58], [41], [11]. 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. [45] first proposed using non-standard analysis to discuss the nature of time in hybrid systems. Bliudze and Krob [26], [25] 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.

3.3. Structural Analysis of DAE Systems

The Modelica language is based on Differential Algebraic Equations (DAE). The general form of a DAE is given by:

$$F(t, x, x', x'', \dots) \quad (2)$$

where F is a system of n_e equations $\{f_1, \dots, f_{n_e}\}$ and x is a finite list of n_v independent real-valued, smooth enough, functions $\{x_1, \dots, x_{n_v}\}$ of the independent variable t . We use x' as a shorthand for the list of first-order time derivatives of x_j , $j = 1, \dots, n_v$. High-order derivatives are recursively defined as usual, and $x^{(k)}$ denotes the list formed by the k -th derivatives of the functions x_j . Each f_i depends on the scalar t and some of the functions x_j as well as a finite number of their derivatives.

Let $\sigma_{i,j}$ denote the highest differentiation order of variable x_j effectively appearing in equation f_i , or $-\infty$ if x_j does not appear in f_i . The *leading variables* of F are the variables in the set

$$\left\{ x_j^{(\sigma_j)} \mid \sigma_j = \max_i \sigma_{i,j} \right\}$$

The *state variables* of F are the variables in the set

$$\left\{ x_j^{(\nu_j)} \mid 0 \leq \nu_j < \max_i \sigma_{i,j} \right\}$$

A leading variable $x_j^{(\sigma_j)}$ is said to be *algebraic* if $\sigma_j = 0$ (in which case, neither x_j nor any of its derivatives are state variables). In the sequel, v and u denote the leading and state variables of F , respectively.

DAE are a strict generalization of *ordinary differential equations (ODE)*, in the sense that it may not be immediate to rewrite a DAE as an explicit ODE of the form $v = G(u)$. The reason is that this transformation relies on the Implicit Function Theorem, requiring that the Jacobian matrix $\frac{\partial F}{\partial v}$ have full rank. This is, in general, not the case for a DAE. Simple examples, like the two-dimensional fixed-length pendulum in Cartesian coordinates [55], exhibit this behaviour.

For a square DAE of dimension n (i.e., we now assume $n_e = n_v = n$) to be solved in the neighborhood of some (v^*, u^*) , one needs to find a set of non-negative integers $C = \{c_1, \dots, c_n\}$ such that system

$$F^{(C)} = \{f_1^{(c_1)}, \dots, f_n^{(c_n)}\}$$

can locally be made explicit, i.e., the Jacobian matrix of $F^{(C)}$ with respect to its leading variables, evaluated at (v^*, u^*) , is nonsingular. The smallest possible value of $\max_i c_i$ for a set C that satisfies this property is the *differentiation index* [32] of F , that is, the minimal number of time differentiations of all or part of the equations f_i required to get an ODE.

In practice, the problem of automatically finding a "minimal" solution C to this problem quickly becomes intractable. Moreover, the differentiation index may depend on the value of (v^*, u^*) . This is why, in lieu of numerical nonsingularity, one is interested in the *structural nonsingularity* of the Jacobian matrix, i.e., its almost certain nonsingularity when its nonzero entries vary over some neighborhood. In this framework, the *structural analysis* (SA) of a DAE returns, when successful, values of the c_i that are independent from a given value of (v^*, u^*) .

A renowned method for the SA of DAE is the *Pantelides method*; however, Pryce's Σ -method is introduced also in what follows, as it is a crucial tool for our works.

3.3.1. Pantelides method

In 1988, Pantelides proposed what is probably the most well-known SA method for DAE [55]. The leading idea of his work is that the structural representation of a DAE can be condensed into a bipartite graph whose left nodes (resp. right nodes) represent the equations (resp. the variables), and in which an edge exists if and only if the variable occurs in the equation.

By detecting specific subsets of the nodes, called *Minimally Structurally Singular* (MSS) subsets, the Pantelides method iteratively differentiates part of the equations until a perfect matching between the equations and the leading variables is found. One can easily prove that this is a necessary and sufficient condition for the structural nonsingularity of the system.

The main reason why the Pantelides method is not used in our work is that it cannot efficiently be adapted to multimode DAE (mDAE). As a matter of fact, the adjacency graph of a mDAE has both its nodes and edges parametrized by the subset of modes in which they are active; this, in turn, requires that a parametrized Pantelides method must branch every time no mode-independent MSS is found, ultimately resulting, in the worst case, in the enumeration of modes.

3.3.2. Pryce's Σ -method

Albeit less renowned than the Pantelides method, Pryce's Σ -method [56] is an efficient SA method for DAE, whose equivalence to the Pantelides method has been proved by the author. This method consists in solving two successive problems, denoted by primal and dual, relying on the Σ -matrix, or *signature matrix*, of the DAE F .

This matrix is given by:

$$\Sigma = (\sigma_{ij})_{1 \leq i, j \leq n} \quad (3)$$

where σ_{ij} is equal to the greatest integer k such that $x_j^{(k)}$ appears in f_i , or $-\infty$ if variable x_j does not appear in f_i . It is the adjacency matrix of a weighted bipartite graph, with structure similar to the graph considered in the Pantelides method, but whose edges are weighted by the highest differentiation orders. The $-\infty$ entries denote non-existent edges.

The *primal problem* consists in finding a *maximum-weight perfect matching (MWPM)* in the weighted adjacency graph. This is actually an assignment problem, for the solving of which several standard algorithms exist, such as the push-relabel algorithm [44] or the Edmonds-Karp algorithm [43] to only give a few. However, none of these algorithms are easily parametrizable, even for applications to mDAE systems with a fixed number of variables.

The *dual problem* consists in finding the component-wise minimal solution $(C, D) = (\{c_1, \dots, c_n\}, \{d_1, \dots, d_n\})$ to a given linear programming problem, defined as the dual of the aforementioned assignment problem. This is performed by means of a *fixpoint iteration (FPI)* that makes use of the MWPM found as a solution to the primal problem, described by the set of tuples $\{(i, j_i)\}_{i \in \{1, \dots, n\}}$:

1. Initialize $\{c_1, \dots, c_n\}$ to the zero vector.

2. For every $j \in \{1, \dots, n\}$,

$$d_j \leftarrow \max_i (\sigma_{ij} + c_i)$$

3. For every $i \in \{1, \dots, n\}$,

$$c_i \leftarrow d_{j_i} - \sigma_{i, j_i}$$

4. Repeat Steps 2 and 3 until convergence is reached.

From the results proved by Pryce in [56], it is known that the above algorithm terminates if and only if it is provided a MWPM, and that the values it returns are independent of the choice of a MWPM whenever there exist several such matchings. In particular, a direct corollary is that the Σ -method succeeds as long as a perfect matching can be found between equations and variables.

Another important result is that, if the Pantelides method succeeds for a given DAE F , then the Σ -method also succeeds for F and the values it returns for C are exactly the differentiation indices for the equations that are returned by the Pantelides method. As for the values of the d_j , being given by $d_j = \max_i (\sigma_{ij} + c_i)$, they are the differentiation indices of the leading variables in $F^{(C)}$.

Working with this method is natural for our works, since the algorithm for solving the dual problem is easily parametrizable for dealing with multimode systems, as shown in our recent paper [31].

3.3.3. Block triangular decomposition

Once structural analysis has been performed, system $F^{(C)}$ can be regarded, for the needs of numerical solving, as an algebraic system with unknowns $x_j^{(d_j)}$, $j = 1 \dots n$. As such, (inter)dependencies between its equations must be taken into account in order to put it into block triangular form (BTF). Three steps are required:

1. the *dependency graph* of system $F^{(C)}$ is generated, by taking into account the perfect matching between equations $f_i^{(c_i)}$ and unknowns $x_j^{(d_j)}$;
2. the *strongly connected components (SCC)* in this graph are determined: these will be the *equation blocks* that have to be solved;
3. the *block dependency graph* is constructed as the condensation of the dependency graph, from the knowledge of the SCC; a BTF of system $F^{(C)}$ can be made explicit from this graph.

3.4. 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 [10]. 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 [53]. 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 [3]. 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 [46], [35], [52], [14], [37]. Contract theories were initially developed as specification formalisms able to refuse some inputs from the environment [42]. A/G-contracts were advocated in [18] and are still a very active research topic, with several contributions dealing with the timed [24] and probabilistic [29], [30] viewpoints in system design, and even mixed-analog circuit design [54].

Automata theoretic interfaces. Interfaces combine assumptions and guarantees in a single, automata theoretic specification. Most interface theories are based on Lynch Input/Output Automata [51], [50]. Interface Automata [61], [60], [62], [33] 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 [57] inherit from both Interface Automata and the originally unrelated notion of Modal Transition System [48], [13], [27], [47]. 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 [63], [21], [23], [39], [38], [22], probabilistic [29], [40] and energy-aware [34] 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 [59]. Most requirements engineering tools offer a poor structuring of the requirements and cannot be considered as formal modeling frameworks today. They are nothing less, but nothing more than an informal structured 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 were working on the development of the fly-by-wire and of the landing gear subsystems, leading to a long and chaotic convergence of the design process.

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.

Kairos Project-Team

3. Research Program

3.1. Cyber-Physical co-modeling

Cyber-Physical System modeling requires joint representation of digital/cyber controllers and natural physics environments. Heterogeneous modeling must then be articulated to support accurate (co-)simulation, (co-)analysis, and (co-)verification. The picture above sketches the overall design framework. It comprises functional requirements, to be met provided surrounding platform guarantees, in a contract approach. All relevant aspects are modeled with proper Domain Specific Languages (DSL), so that constraints can be gathered globally, then analyzed to build a mapping proposal with both a structural aspect (functions allocated to platform resources), but also a behavioral ones, scheduling activities. Mapping may be computed automatically or not, provably correct or not, obtained by static analytic methods or abstract execution. Physical phenomena (in a very broad acceptance of the term) are usually modeled using continuous-time models and differential equations. Then the “proper” discretization opportunities for numerical simulation form a large spectrum of mathematical engineering practices. This is not at all the domain of expertise of Kairos members, but it should not be a limitation as long as one can assume a number of properties from the discretized version. On the other hand, we do have a strong expertise on modeling of both embedded processing architectures and embedded software (i.e., the kind of usually concurrent, sometimes distributed software that reacts to and control the physical environment). This is important as, unlike in the “physical” areas where modeling is common-place, modeling of software and programs is far from mainstream in the Software Engineering community. These domains are also an area of computer science where modeling, and even formal modeling, of the real objects that are originally of discrete/cyber nature, takes some importance with formal Models of Computation and Communications. It seems therefore quite natural to combine physical and cyber modeling in a more global design approach (even multi-physic domains and systems of systems possibly, but always with software-intensive aspects involved). Our objective is certainly not to become experts in physical modeling and/or simulation process, but to retain from it only the essential and important aspects to include them into System-Level Engineering design, based on Model-Driven approaches allowing formal analysis.

This sets an original research agenda: Model-Based System Engineering environments exist, at various stages of maturity and specificity, in the academic and industrial worlds. Formal Methods and Verification/Certification techniques also exist, but generally in a point-wise fashion. Our approach aims at raising the level of formality describing relevant features of existing individual models, so that formal methods can have a greater general impact on usual, “industrial-level”, modeling practices. Meanwhile, the relevance of formal methods is enhanced as it now covers various aspects in a uniform setting (timeliness, energy budget, dependability, safety/security...).

New research directions on formal CPS design should focus on the introduction of uncertainty (stochastic models) in our particular framework, on relations between (logical) real-time and security, on relations between common programming languages paradigms and logical time, on extending logical frameworks with logical time, on the concern with resource discovery also in presence of mobility inherent to connected objects and Internet of Things.

3.2. Cyber-Physical co-simulation

The FMI standard (Functional Mock-Up Interface) has been proposed for “purely physical” (i.e., based on persistent signals) co-simulation, and then adopted in over 100 industrial tools including frameworks such as Matlab/Simulink and Ansys, to mention two famous model editors. With the recent use of co-simulation to cyber-physical systems, dealing with the discrete and transient nature of cyber systems became mandatory. Together with other people from the community, we shown that FMI and other frameworks for co-simulation

badly support co-simulation of cyber-physical systems; leading to bad accuracy and performances. More precisely, the way to interact with the different parts of the co-simulation require a specific knowledge about its internal semantics and the kind of data exposed (e.g., continuous, piecewise-constant). Towards a better co-simulation of cyber-physical systems, we are looking for conservative abstractions of the parts and formalisms that aim to describe the functional and temporal constraints that are required to bind several simulation models together.

3.3. Formal analysis and verification

Because the nature of our constraints is specific, we want to adjust verification methods to the goals and expressiveness of our modeling approach. Quantitative (interval) timing conditions on physical models combined with (discrete) cyber modes suggest the use of SMT (Satisfiability Modulo Theories) automatic solvers, but the natural expressiveness requested (as for instance in our CCSL constructs) shows this is not always feasible. Either interactive proofs, or suboptimal solutions (essentially resulting of abstract run-time simulations) should be considered. Complementarily to these approaches, we are experimenting with new variants of symbolic behavioural semantics, allowing to construct finite representations of the behaviour of CPS systems with explicit handling of data, time, or other non-functional aspects.

3.4. Relation between Model and Code

While models considered in Kairos can also be considered as executable specifications (through abstract simulation schemes), they can also lead to code synthesis and deployment. Conversely, code execution of smaller, elementary software components can lead to performance estimation enriching the models before global mapping optimization. CPS introduce new challenging problems for code performance stability. Indeed, two additional factors for performance variability appear, which were not present in classical embedded systems: 1) variable and continuous data input from the physical world and 2) variable underlying hardware platform. For the first factor, CPS software must be analysed in conjunction with its data input coming from the physics, so the variability of the performance may come from the various data. For the second factor, the underlying hardware of the CPS may change during the time (new computing actors appear or disappear, some actors can be reconfigured during execution). The new challenge is to understand how these factors influence performance variability exactly, and how to provide solutions to reduce it or to model it. The modeling of performance variability becomes a new input.

3.5. Code generation and optimization

A significant part CPS design happens at model level, through activities such as model construction, analysis, or verification. However, in most cases the objective of the design process is implementation. We mostly consider the implementation problem in the context of embedded, real-time, or edge computing applications, which are subject to stringent performance, embedding, and safety *non-functional requirements*.

The implementation of such systems usually involves a mix of synthesis—(real-time) scheduling, code generation, compilation—and performance (e.g. timing) analysis. One key difficulty here is that synthesis and performance analysis depend on each other. As enumerating the various solutions is not possible for complexity reasons, heuristic implementation methods are needed in all cases. One popular solution here is to build the system first using unsafe performance estimations for its components, and then check system *schedulability* through a global analysis. Another solution is to use safe, over-approximated performance estimations and perform their mapping in a way that ensures by construction the schedulability of the system.

In both cases, the specification of the design space—functional specification, execution platform model, non-functional requirements, implementation model—is a key problem. Another problem is the definition of scalable and efficient mapping methods based on both "exact" approaches (ILP/SMT/CP solving) and compilation-like heuristics.

3.6. Extending logical frameworks with logical time

The Curry-Howard isomorphism (*proposition-as-types and proofs-as-typed- λ -terms*) represent the logical and computational basis to interactive theorem provers: our challenge is to investigate and design time constraints within a Dependent Type Theory (e.g. if event A happened-before event B, then the timestamp/type of A is less (i.e. a subtype) than the timestamp/type of B). We are currently extending the Edinburgh Logical Framework (LF) of Harper-Honsell-Plotkin with relevant constructs expressing logical time and synchronization between processes. Also, union and intersection types with their subtyping constraints theories could capture some constraints expressions *à la* CCSL needed to formalize logical clocks (in particular CCSL expressions like subclock, clock union, intersection and concatenation) and provide opportunities for an *ad hoc* polymorphic timed Type Theory. Logical time constraints seen as property types can beneficially be handled by logical frameworks. The new challenge here is to demonstrate the relevance of Type Theory to work on logical and multiform timing constraint resolution.

3.7. Object-oriented programming and logical time

We formalize in the past object-oriented programming features and safe static type systems featuring delegation-based or trait inheritance: well typed program will never produce into the `message-not-found` infamous run-time error. We view the logical time as a means to enhance the description of timing constraints and properties on top of existing language semantics. When considering general purpose object-oriented languages, like Java, Type Theory is a natural way to provide such properties. Currently, few languages have special types to manage instants, time structures and instant relations like subclocking, precedence, causality, equality, coincidence, exclusion, independence, etc. CCSL provides ad-hoc constructors to specify clock constraints and logical time: enriching object-oriented type theories with CCSL expressions could constitute an interesting research perspective towards a wider usage of CCSL. The new challenge is to consider logical time constraints as behavioral type properties, and the design of programming language constructs and *ad-hoc* type systems. Advances of typed-calculi featuring those static time features will be applied to our extension [42] of the lambda-calculus of objects of Fisher-Honsell-Mitchell.

3.8. Extensions for spatio-temporal modeling and mobile systems

While Time is clearly a primary ingredient in the proper design of CPS systems, in some cases Space, and related notions of local proximity or conversely long distance, play also a key role for correct modeling, often in part because of the constraints this puts on interactions and time for communications. Once space is taken into account, one has to recognize also that many systems will request to consider mobility, originated as change of location through time. Mobile CPS (or mCPS) systems occur casually, e.g., in the case of Intelligent Transportation Systems, or in roaming connected objects of the IoT. Spatio-temporal and mobility modeling may each lead to dynamicity in the representation of constraints, with the creation/deletion/discovering of new components in the system. This opportunity for new expressiveness will certainly cause new needs in handling constraint systems and topological graph locations. The new challenge is to provide an algebraic support with a constraint description language that could be as simple and expressive as possible, and of use in the semantic annotations for mobile CPS design. We also aims to provide fully distributed routing protocols to manage Semantic Resource Discovery in IoT.

KOPERNIC Team

3. Research Program

3.1. Worst case execution time estimation of a program

Modern processors induce an increased variability of the execution time of programs, making difficult (or even impossible) a complete static analysis. Our objective is to propose a solution composing probabilistic and non-probabilistic approaches based both on static and on statistical analyses by answering the following **scientific challenges**:

1. **a classification of the variability of execution times** of a program with respect to the processor features. We will use as first measure our statistical estimator based on the Extreme Value Theory [18], [20]. An implementation of the estimator is available at <http://inria-rscript.serveftp.com>. The access to this later page requires a login (aoste) and a password (aoste). The difficulty of this challenge is related to the definition of an element belonging to the set of variability factors and its mapping to the execution time of the program.
2. **a compositional rule** of statistical models based on Bayesian approaches. The difficulty of this challenge comes from the fact that a global maximum cannot be obtained by upper bounding the corresponding local maxima. We will use as first rule of composition a Bayesian approach [22]. We consider as first statistical model those obtained by any static analysis of the program on a basic processor. Through the Bayesian approach we add iteratively the variability due to each processor feature as a new statistical model. The convergence of the global model is decided once no variability is detected at the level of the statistical estimator providing the bounds on the execution time of the program.

The problem of estimating the worst case execution time of a program is an excellent opportunity for the Extreme Values community to validate and to evolve as the context of obtaining measures is indefinitely reproducible.

3.2. Deciding the schedulability of all programs running within the same cyber component

In this context, the programs may have different time criticalities, but they share the same processor, possibly multicore⁰. Our objective is to propose a solution composing probabilistic and non-probabilistic approaches based on answers to the following **scientific challenges**:

1. **scheduling algorithms taking into account the interaction between different variability factors**. The proposed scheduling algorithms are the theoretical bases of a scheduler able to guarantee the time constraints of the cyber component. The existence of time parameters described by probability distributions imposes to answer to the challenge of revisiting scheduling algorithms that lose their optimality even in the case of an uncore processor [26]. Moreover, the multicore partitioning problem is, also, recognized difficult for the non-probabilistic case [29];
2. **schedulability analyses** based on the algorithms proposed previously. In the case of predictable processors, the schedulability analyses accounting for operating systems costs increase the dependability of CPSs [28]. Moreover, in presence of variability factors, the additivity property of non-probabilistic approaches is lost and new composition principles are required. We will propose new composition principles based on our preliminary results on the propagation of the probabilistic constraints [16]. The definition of these principles form the challenge related to this objective.

⁰This case is referred as a mixed criticality approach.

3.3. Deciding the schedulability of all programs communicating through predictable and non-predictable networks

In this case the programs of the same cyber component execute on the same processor and they may communicate with the programs of other cyber components through networks that may be predictable (network on chip) or non-predictable (internet, telecommunications). Our objective is to propose a solution to the challenge of analysing schedulability of programs, for which existing (worst case) probabilistic solutions exist [27], communicating through networks, for which probabilistic worst-case solutions [19] and average solutions exist [24]. Our solution is based on the results obtained for the two first objectives, making this third objective a longer-term one.

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,⁰ 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,⁰ with MODELICA⁰ 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.

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

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

⁰<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.

SPADES Project-Team

3. Research Program

3.1. Introduction

The SPADES research program is organized around three main themes, *Design and Programming Models*, *Certified real-time programming*, and *Fault management and causal analysis*, 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. Design and Programming Models

Work on this theme aims to develop models, languages and tools to support a “correct-by-construction” approach to the development of embedded systems.

On the programming side, we focus on the definition of domain specific programming models and languages supporting static analyses for the computation of precise resource bounds for program executions. We propose dataflow models supporting dynamicity while enjoying effective analyses. In particular, we study parametric extensions where properties such as liveness and boundedness remain statically analyzable.

On the design side, we focus on the definition of component-based models for software architectures combining distribution, dynamicity, real-time and fault-tolerant aspects. Component-based construction has long been advocated as a key approach to the “correct-by-construction” design of complex embedded systems [55]. Witness component-based toolsets such as PTOLEMY [47], BIP [38], or the modular architecture frameworks used, for instance, in the automotive industry (AUTOSAR) [30]. 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.

Formal models for component-based design are an active area of research. 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).

We plan to develop our component theory by progressing on two fronts: a semantical framework and domain-specific programming models. 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 first main objective for this axis.

3.3. Certified Real-Time 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 [40]), 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 [36], [45], 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. [36]. For our part, we intend to focus on devising synchronous programming languages for distributed systems and precision-timed architectures.

3.4. Fault Management and Causal Analysis

Managing faults is a clear and present necessity in networked embedded systems. At the hardware level, modern multicore architectures are manufactured using inherently unreliable technologies [41], [51]. 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.

In this axis we intend to address the question of *how to cope with faults and failures in embedded systems?*. We will tackle this question by exploiting reversible programming models and by developing techniques for fault ascription and explanation in component-based systems.

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 [61], natural sciences, law [62], and statistics [63], 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 [58], to allow the diagnosis of faults in a complex concurrent system [54], or to enforce accountability [57], 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 [53]), or is broken (*e.g.*, by limiting fault propagation [65]).

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.

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-timeTM-synchronized clouds, of tomorrow’s grids.

By contrast with a synchronous hypothesis, yet from the same era, the polychronous MoCC is inherently capable of describing multi-clock abstractions of GALS systems. Polychrony is implemented in the data-flow specification language Signal, available in the Eclipse project POP⁰ and in the CCSL standard⁰ available from the TimeSquare project. Both provide tooled 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. Timed Modeling

To formalize timed semantics for system design, we shall rely on algebraic representations of time as clocks found in previous works and introduce a paradigm of “time system” (types that 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 timed concurrency.

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

⁰Clock 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 systems⁰, 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 logic⁰. 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 requires the use of assume-guarantee reasoning to allow modularity and to facilitate composition by behavioral sub-typing, in the spirit of the (static) contract-based formalism proposed by Passerone et al.⁰. Verification problems encompassing heterogeneously timed specifications are common and of great variety: checking correctness between abstract (e.g. the synchronous hypothesis) and concrete time models (e.g. real-time architectures) relates to desynchronisation (from synchrony to asynchrony) and scheduling analysis (from synchronous data-flow 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 to use abstraction and refinement mechanisms for time models (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 tooling validation of common cross-domain properties (e.g. cost v.s. power v.s. performance v.s. software mapping) and tackle problems such as these integrating constraints of battery capacity, on-board CPU performance, available memory resources, software schedulability, to logical software correctness and plant controllability.

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 AADL⁰, SysML⁰, UML MARTE⁰.

In system design, an architectural specification serves several important purposes. First, it breaks down a system model into components of manageable size and complexity, 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 interact with 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.

Related works in this aim, and closer in spirit to our approach (to focus on modeling time) are domain-specific languages such as Prelude⁰ to model the real-time characteristics of embedded software architectures.

⁰Abstract Refinement Types. N. Vazou, P. Rondon, and R. Jhala. European Symposium on Programming. Springer, 2013.

⁰LTL types FRP. A. Jeffrey. Programming Languages meets Program Verification.

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

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

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

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

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

Conversely, standard architecture description languages could be based on algebraic modeling tools, such as interface theories with the ECDAR tool⁰.

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 scheduling⁰ 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 techniques⁰. It consists, first, of approximating task communication patterns (e.g. between 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 to concretize them as real-time schedules in the program implementations⁰⁰.

Abstract scheduling, or the use of abstraction and refinement techniques in scheduling borrowed to the theory of abstract interpretation⁰ is a promising development toward toolled 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 focus.

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. Verified programming for system design

The IoT is a network of devices that sense, actuate and change our immediate environment. Against this fundamental role of sensing and actuation, design of edge devices often considers actions and event timings to be primarily software implementation issues: programming models for IoT abstract even the most rudimentary information regarding timing, sensing and the effects of actuation. As a result, applications programming interfaces (API) for IoT allow wiring systems fast without any meaningful assertions about correctness, reliability or resilience.

We make the case that the "API glue" must give way to a logical interface expressed using contracts or refinement types. Interfaces can be governed by a calculus – a refinement type calculus – to enable reasoning on time, sensing and actuation, in a way that provides both deep specification refinement, for mechanized verification of requirements, and multi-layered abstraction, to support compositionality and scalability, from one end of the system to the other.

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

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

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

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

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

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

Our project seeks to elevate the “function as type” paradigm to that of “system as type”: to define a refinement type calculus based on concepts of contracts for reasoning on networked devices and integrate them as cyber-physical systems ⁰. An invited paper ⁰ outlines our progress with respect to this aim and plans towards building a verified programming environment for networked IoT devices: we propose a type-driven approach to verifying and building safe and secure IoT applications.

Accounting for such constraints in a more principled fashion demands reasoning about the composition of all the software and hardware components of the application. Our proposed framework takes a step in this direction by (1) using refinement types to make physical constraints explicit and (2) imposing an event-driven programming discipline to simplify the reasoning of system-wide properties to that of an event queue. In taking this approach, our approach would make it possible for a developer to build a verified IoT application by ensuring that a well-typed program cannot violate the physical constraints of its architecture and environment.

⁰Refinement types for system design. Jean-Pierre Talpin. FDL’18 keynote.

⁰Steps toward verified programming of embedded computing systems. Jean-Pierre Talpin, Jean-Joseph Marty, Deian Stefan, Shravan Nagarayan, Rajesh Gupta, DATE’18.

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 one to express 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* [22]. 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* [22], which over-approximate 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.

The same principles can be applied successfully to other goals. In particular, the abstract interpretation framework should be viewed as 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 [21], 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. From properties to explanations

In many application domains, we can go beyond the proof that a program satisfies its specification. Abstractions can also offer new perspectives to understand how complex behaviors of programs emerge from simpler computation steps. Abstractions can be used to find compact and readable representations of sets of traces, causal relations, and even proofs. For instance, abstractions may decipher how the collective behaviors of agents emerge from the orchestration of their individual ones in distributed systems (such as consensus protocols, models of signaling pathways). Another application is the assistance for the diagnostic of alarms of a static analyzer.

Complex systems and software have often times intricate behaviors, leading to executions that are hard to understand for programmers and also difficult to reason about with static analyzers. Shared memory and distributed systems are notorious for being hard to reason about due to the interleaving of actions performed by different processes and the non-determinism of the network that might lose, corrupt, or duplicate messages. Reduction theorems, e.g., Lipton's theorem, have been proposed to facilitate reasoning about concurrency, typically transforming a system into one with a coarse-grained semantics that usually increases the atomic sections. We investigate reduction theorems for distributed systems and ways to compute the coarse-grained counter part of a system automatically. Compared with shared memory concurrency, automated methods to reason about distributed systems have been less investigated in the literature. We take a programming language approach based on high-level programming abstractions. We focus on partially-synchronous communication closed round-based models, introduced in the distributed algorithms community for its simpler proof arguments. The high-level language is compiled into a low-level (asynchronous) programming language. Conversely, systems defined under asynchronous programming paradigms are decompiled into the high-level programming abstractions. The correctness of the compilation/decompilation process is based on reduction theorems (in the spirit of Lipton and Elrad-Francez) that preserve safety and liveness properties.

In models of signaling pathways, collective behavior emerges from competition for common resources, separation of scales (time/concentration), non linear feedback loops, which are all consequences of mechanistic interactions between individual bio-molecules (e.g., proteins). While more and more details about mechanistic interactions are available in the literature, understanding the behavior of these models at the system level is far from easy. Causal analysis helps explaining how specific events of interest may occur. Model reduction techniques combine methods from different domains such as the analysis of information flow used in communication protocols, and tropicalization methods that comes from physics. The result is lower dimension systems that preserve the behavior of the initial system while focusing of the elements from which emerges the collective behavior of the system.

The abstraction of causal traces offer nice representation of scenarios that lead to expected or unexpected events. This is useful to understand the necessary steps in potential scenarios in signaling pathways; this is useful as well to understand the different steps of an intrusion in a protocol. Lastly, traces of computation of

a static analyzer can themselves be abstracted, which provides assistance to classify true and false alarms. Abstracted traces are symbolic and compact representations of sets of counter-examples to the specification of a system which help one to either understand the origin of bugs, or to find that some information has been lost in the abstraction leading to false alarms.

CAMBIUM Project-Team

3. Research Program

3.1. Research Directions

Our research proposal is organized along three main axes, namely **programming language design and implementation**, **concurrency**, and **program verification**. These three areas have strong connections. For instance, the definition and implementation of Multicore OCaml intersects the first two axes, whereas creating verification technology for Multicore OCaml programs intersects the last two.

In short, the “programming language design and implementation” axis includes:

- The search for richer type disciplines, in an effort to make our programming languages safer and more expressive. Two domains, namely modules and effects, appear of particular interest. In addition, we view type inference as an important cross-cutting concern.
- The continued evolution of OCaml. The major evolutions that we envision in the medium term are the integration of Multicore OCaml, the addition of modular implicits, and a redesign of the type-checker.
- Research on refactoring and program transformations.

The “concurrency” axis includes:

- Research on weak memory models, including axiomatic models, operational models, and event-structure models.
- Research on the Multicore OCaml memory model. This might include proving that the axiomatic and operational presentations of the model agree; testing the Multicore OCaml implementation to ensure that it conforms to the model; and extending the model with new features, should the need arise.

The “program verification” axis includes:

- The continued evolution of CompCert.
- Building new verified tools, such as verified compilers for domain-specific languages, verified components for the Coq type-checker, and so on.
- Verifying algorithms and data structures implemented in OCaml and in Multicore OCaml and enriching Separation Logic with new features, if needed, to better support this activity.
- The continued development of tools for TLA+.

CELTIQUE Project-Team (section vide)

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 [6] 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 μ -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*) [47], which extends the modal μ -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.

DEDUCTEAM Project-Team

3. Research Program

3.1. Logical Frameworks

A thesis, which is at the root of our research effort, is that logical systems should be expressed as theories in a logical framework. As a consequence, proof-checking systems should not be focused on one theory, such as Simple type theory, Martin-Löf's type theory, or the Calculus of constructions, but should be theory independent. On the more theoretical side, the proof search algorithms, or the algorithmic interpretation of proofs should not depend on the theory in which proofs are expressed, but this theory should just be a parameter. This is for instance expressed in the title of our invited talk at ICALP 2012: *A theory independent Curry-De Bruijn-Howard correspondence* [25].

Various limits of Predicate logic have led to the development of various families of logical frameworks: λ -prolog and Isabelle have allowed terms containing free variables, the Edinburgh logical framework has allowed proofs to be expressed as λ -terms, Pure type systems have allowed propositions to be considered as terms, and Deduction modulo theory has allowed theories to be defined not only with axioms, but also with computation rules.

The $\lambda\Pi$ -calculus modulo theory, that is implemented in the system DEDUKTI and that is a synthesis of the Edinburgh logical framework and of Deduction modulo theory, subsumes them all. Part of our research effort is focused on improving the $\lambda\Pi$ -calculus modulo theory, for instance allowing to define congruences with associative and commutative rewriting. Another part of our research effort is focused on the automatic analysis of theories to prove their confluence, termination, and consistency either by pencil and paper proofs or automatically [4].

3.2. Interoperability and proof encyclopediae

Using a single prover to check proofs coming from different systems naturally leads to investigate how these proofs can be translated from one theory to another and used in a system different from the system in which they have been developed. 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.

For each proof, independently of the system in which it has been developed, we should be able to identify the systems in which it can be expressed. For instance, we have shown that many proofs developed in the MATITA prover did not use the full strength of the logic of MATITA and could be exported, for instance, to the systems of the HOL family, that are based on a weaker logic.

Rather than importing proofs from one system, transforming them, and exporting them to another system, we can use the same tools to develop system-independent proof encyclopedia called Logipedia. In such a library, each proof is labeled with the theories in which it can be expressed and so with the systems in which it can be used.

3.3. Interactive theorem proving

If our main goal with DEDUKTI is to import, transform, and export proofs developed in other systems, we also want to investigate how DEDUKTI can be used as the basis of an interactive theorem prover. This leads to two new scientific questions: first, how much can a tactic system be theory independent, and then how does rewriting extends the possibility to write tactics.

This has led to the development of a new version of DEDUKTI, which supports metavariables. Several tactics have been developed for this system, which are intended to help a human user to write proofs in our system instead of writing proof terms by hand. This work is a continuation of the previous work the team did on DEMON, which was an extension of DEDUKTI, whereas the support for interactive theorem proving is now native in DEDUKTI.

GALLINETTE Project-Team

3. Research Program

3.1. Scientific Context

Software quality is a requirement that is becoming more and more prevalent, by now far exceeding the traditional scope of embedded systems. The development of tools to construct software that respects a given specification is a major challenge facing computer science. *Proof assistants* such as Coq [49] provide a formal method whose central innovation is to produce *certified programs* by transforming the very activity of programming. Programming and proving are merged into a single development activity, informed by an elegant but rigid mathematical theory inspired by the correspondence between programming, logic and algebra: the *Curry-Howard correspondence*. For the certification of programs, this approach has shown its efficiency in the development of important pieces of certified software such as the C compiler of the CompCert project [78]. The extracted CompCert compiler is reliable and efficient, running only 15% slower than GCC 4 at optimisation level 2 (`gcc -O2`), a level of optimisation that was considered before to be highly unreliable.

Proof assistants can also be used to *formalise mathematical theories*: they not only provide a means of representing mathematical theories in a form amenable to computer processing, but their internal logic provides a language for reasoning about such theories. In the last decade, proof assistants have been used to verify extremely large and complicated proofs of recent mathematical results, sometimes requiring either intensive computations [60], [64] or intricate combinations of a multitude of mathematical theories [59]. But formalised mathematics is more than just proof checking and proof assistants can help with the organisation mathematical knowledge or even with the discovery of new constructions and proofs.

Unfortunately, the rigidity of the theory behind proof assistants impedes their expressiveness both as programming languages and as logical systems. For instance, a program extracted from Coq only uses a purely functional subset of OCaml, leaving behind important means of expression such as side-effects and objects. Limitations also appears in the formalisation of advanced mathematics: proof assistants do not cope well with classical axioms such as excluded middle and choice which are sometimes used crucially. The fact of the matter is that the development of proof assistants cannot be dissociated from a reflection on the nature of programs and proofs coming from the Curry-Howard correspondence. In the EPC Gallinette, we propose to address several drawbacks of proof assistants by pushing the boundaries of this correspondence.

In the 1970's, the Curry-Howard correspondence was seen as a perfect match between functional programs, intuitionistic logic, and Cartesian closed categories. It received several generalisations over the decades, and now it is more widely understood as a fertile correspondence between computation, logic, and algebra. Nowadays, the view of the Curry-Howard correspondence has evolved from a perfect match to a collection of theories meant to explain similar structures at work in logic and computation, underpinned by mathematical abstractions. By relaxing the requirement of a perfect match between programs and proofs, and instead emphasising the common foundations of both, the insights of the Curry-Howard correspondence may be extended to domains for which the requirements of programming and mathematics may in fact be quite different.

Consider the following two major theories of the past decades, which were until recently thought to be irreconcilable:

- **(Martin-Löf) Type theory:** introduced by Martin-Löf in 1971, this formalism [85] is both a programming language and a logical system. The central ingredient is the use of *dependent types* to allow fine-grained invariants to be expressed in program types. In 1985, Coquand and Huet developed a similar system called the *calculus of constructions*, which served as logical foundation of the first implementation of Coq. This kind of systems is still under active development, especially with the recent advent of homotopy type theory (HoTT) [107] which gives a new point of view on types and the notion of equality in type theory.

- **The theory of effects:** starting in the 1980's, Moggi [90] and Girard [57] put forward monads and co-monads as describing various compositional notions of computation. In this theory, programs can have side-effects (state, exceptions, input-output), logics can be non-intuitionistic (linear, classical), and different computational universes can interact (modal logics). Recently, the safe and automatic management of resources has also seen a coming of age (Rust, Modern C++) confirming the importance of linear logic for various programming concepts. It is now understood that the characteristic feature of the theory of effects is sensitivity to *evaluation order*, in contrast with type theory which is built around the assumption that evaluation order is irrelevant.

We now outline a series of scientific challenges aimed at understanding of type theory, effects, and their combination.

More precisely, three key axes of improvement have been identified:

1. Making the notion of equality closer to what is usually assumed when doing proofs on black board, with a balance between irrelevant equality for simple structures and equality up-to equivalences for more complex ones (Section 3.2). Such a notion of equality should allow one to implement traditional model transformations that enhance the logical power of the proof assistant using distinct compilation phases.
2. Advancing the foundations of effects within the Curry-Howard approach. The objective is to pave the way for the integration of effects in proof assistants and to prototype the corresponding implementation. This integration should allow for not only certified programming with effects, but also the expression of more powerful logics (Section 3.3).
3. Making more programming features (notably, object polymorphism) available in proof assistants, in order to scale to practical-sized developments. The objective is to enable programming styles closer to common practices. One of the key challenges here is to leverage gradual typing to dependent programming (Section 3.4).

To validate the new paradigms, we propose in Section 3.5 three particular application fields in which members of the team already have a strong expertise: code refactoring, constraint programming and symbolic computation.

3.2. Enhance the computational and logical power of proof assistants

The democratisation of proof assistants based on type theory has likely been impeded one central problem: the mismatch between the conception of equality in mathematics and its formalisation in type theory. Indeed, some basic principles that are used implicitly in mathematics—such as Church's principle of propositional extensionality, which says that two propositions are equal when they are logically equivalent—are not derivable in type theory. Even more problematically, from a computer science point of view, the basic concept of two functions being equal when they are equal at every “point” of their domain is also not derivable: rather, it must be added as an additional axiom. Of course, these principles are consistent with type theory so that working under the corresponding additional assumptions is safe. But the use of these assumptions in a definition potentially clutters its computational behaviour: since axioms are computational black boxes, computation gets stuck at the points of the code where they have been used.

We propose to investigate how expressive logical transformations such as forcing [70] and sheaf construction might be used to enhance the computational and logical power of proof assistants—with a particular emphasis on their implementation in the Coq proof assistant by the means of effective translations (or compilation phases). One of the main topics of this task, in connection to the ERC project CoqHoTT, is the integration in Coq of new concepts inspired by homotopy type theory [107] such as the univalence principle, and higher inductive types.

3.2.1. A definitional proof-irrelevant version of Coq.

In the Coq proof assistant, the sort **Prop** stands for the universe of types which are propositions. That is, when a term P has type **Prop**, the only relevant fact is whether P is inhabited (that is true) or not (that is false). This property, known as *proof irrelevance*, can be expressed formally as: $\forall x y : P, x = y$. Originally, the *raison d'être* of the sort **Prop** was to characterise types with no computational meaning with the intention that terms of such types could be erased upon extraction. However, the assumption that every element of **Prop** should be proof irrelevant has never been integrated to the system. Indeed, in Coq, proof irrelevance for the sort **Prop** is not incorporated into the theory: it is only compatible with it, in the sense that its assumption does not give rise to an inconsistent theory. In fact, the exact status of the sort **Prop** in Coq has never been entirely clarified, which explains in part this lack of integration. Homotopy type theory brings fresh thinking on this issue and suggests turning **Prop** into the collection of terms that a certain static inference procedure tags as proof irrelevant. The goal of this task is to integrate this insight in the Coq system and to implement a definitional proof-irrelevant version of the sort **Prop**.

3.2.2. Extend the Coq proof assistant with a computational version of univalence

The univalence principle is becoming widely accepted as a very promising avenue to provide new foundations for mathematics and type theory. However, this principle has not yet been incorporated into a proof assistant. Indeed, the very mathematical structures (known as ∞ -groupoids) motivating the theory remain to this day an active area of research. Moreover, a correct and decidable type checking procedure for the whole theory raises both computational complexity and logical coherence issues. Observational type theory [32], as implemented in Epigram, provides a first-stage approximation to homotopy type theory, but only deals with functional extensionality and does not capture univalence. Coquand and his collaborators have obtained significant results on the computational meaning of univalence using cubical sets [39], [45]. Bickford has initiated a promising formalisation work⁰ in the NuPRL system. However, a complete formalisation in intensional type theory remains an open problem.

Hence a major objective is to achieve a complete internalisation of univalence in intensional type theory, including an integration to a new version of Coq. We will strive to keep compatibility with previous versions, in particular from a performance point of view. Indeed, the additional complexity of homotopy type theory should not induce an overhead in the type checking procedure used by the software if we want our new framework to become rapidly adopted by the community. Concretely, we will make sure that the compilation time of Coq's Standard Library will be of the same order of magnitude.

3.2.3. Extend the logical power of type theory without axioms in a modular way

Extending the power of a logic using model transformations (*e.g.*, forcing transformation [71], [70] or the sheaf construction [100]) is a classic topic of mathematical logic [46], [76]. However, these ideas have not been much investigated in the setting of type theory, even though they may provide a useful framework for extending the logical power of proof assistant in a modular way. There is a good reason for this: with a syntactic notion of equality, the underlying structure of type theory does not conform to the structure of topos used in mathematical logic. A direct incorporation of the standard techniques is therefore not possible. However, a univalent notion of equality brings type theory closer to the required algebraic structure, as it corresponds to the notion of ∞ -topos recently studied by Lurie [83]. The goal of this task is to revisit model transformations in the light of the univalence principle, and to obtain in this way new internal transformations in type theory which can in turn be seen as compilation phases. The general notion of an internal syntactical translation has already been investigated in the team [40].

3.2.4. Methodology: Extending type theory with different compilation phases

The Gallinette project advocates the use of distinct compilation phases as a methodology for the design of a new generation of proof assistants featuring modular extensions of a core logic. The essence of a compiler is the separation of the complexity of a translation process into modular stages, and the organization of their

⁰<http://www.nuprl.org/wip/Mathematics/cubical!type!theory/index.html>

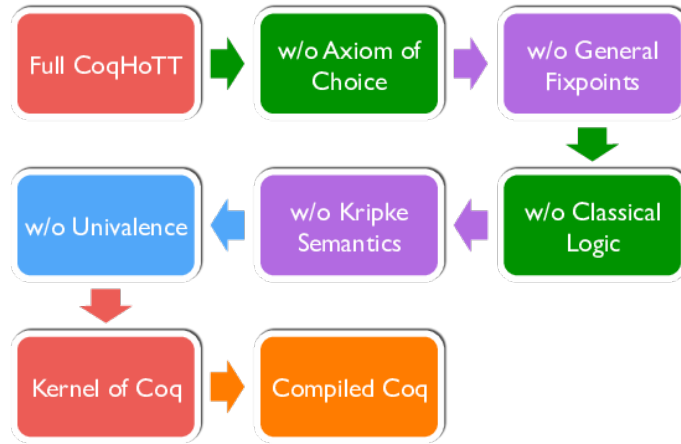


Figure 1. Multiple compilation phases to increase the logical and computational power of Coq.

re-composition. This idea finds a natural application in the design of complex proof assistants (Figure 1). For instance, the definition of type classes in Coq follows this pattern, and is morally given by the means of a translation into a type-class free kernel. More recently, a similar approach by compilation stages, using the forcing transformation, was used to relax the strict positivity condition guarding inductive types [71], [70]. We believe that this flavour of compilation-based strategies offers a promising direction of investigation for the propose of defining a decidable type checking algorithm for HoTT.

3.3. Semantic and logical foundations for effects in proof assistants based on type theory

We propose the incorporation of effects in the theory of proof assistants at a foundational level. Not only would this allow for certified programming with effects, but it would moreover have implications for both semantics and logic.

We mean *effects* in a broad sense that encompasses both Moggi’s monads [90] and Girard’s linear logic [57]. These two seminal works have given rise to respective theories of effects (monads) and resources (co-monads). Recent advances, however have unified these two lines of thought: it is now clear that the defining feature of effects, in the broad sense, is sensitivity to evaluation order [79], [50].

In contrast, the type theory that forms the foundations of proof assistants is based on pure λ calculus and is built on the assumption that evaluation order is irrelevant. Evaluation order is therefore the blind spot of type theory. In Moggi [91], integrating the dependent types of type theory with monads is “*the next difficult step [...] currently under investigation*”.

Any realistic program contains effects: state, exceptions, input-output. More generally, evaluation order may simply be important for complexity reasons. With this in mind, many works have focused on certified programming with effects: notably Ynot [95], and more recently F^{\star} [105] and Idris [41], which propose various ways for encapsulating effects and restricting the dependency of types on effectful terms. Effects are either specialised, such as the monads with Hoare-style pre- and post-conditions found in Ynot or F^{\star} , or more general, such as the algebraic effects implemented in Idris. But whereas there are several experiments and projects pursuing the certification of programs with effects, each making its own choices on how effects and dependency should be merged, there is on the other hand a deficit of logical and semantic investigations.

We propose to develop the foundations of a type theory with effects taking into account the logical and semantic aspects, and to study their practical and theoretical consequences. A type theory that integrates effects would have logical, algebraic and computational implications when viewed through the Curry-Howard correspondence. For instance, effects such as control operators establish a link with classical proof theory [62]. Indeed, control operators provide computational interpretations of type isomorphisms such as $A \cong \neg\neg A$ and $\neg\forall x.A \cong \exists x.\neg A$ (e.g. [92]), whereas the conventional wisdom of type theory holds that such axioms are non-constructive (this is for instance the point of view that has been advocated so far in homotopy type theory [107]). Another example of an effect with logical content is state (more precisely memoization) which is used to provide constructive content to the classical dependent axiom of choice [38], [74], [66]. In the long term, a whole body of literature on the constructive content of classical proofs is to be explored and integrated, providing rich sources of inspiration: Kohlenbach’s proof mining [73] and Simpson’s reverse mathematics [103], for instance, are certainly interesting to investigate from the Curry-Howard perspective.

The goal is to develop a type theory with effects that accounts both for practical experiments in certified programming, and for clues from denotational semantics and logical phenomena, in a unified setting.

3.3.1. Models for integrating effects with dependent types

A crucial step is the integration of dependent types with effects, a topic which has remained “*currently under investigation*” [91] ever since the beginning. The difficulty resides in expressing the dependency of types on terms that can perform side-effects during the computation. On the side of denotational semantics, several extensions of categorical models for effects with dependent types have been proposed [29], [108] using axioms that should correspond to restrictions in terms of expressivity but whose practical implications, however, are not immediately transparent. On the side of logical approaches [66], [67], [77], [89], one first considers a drastic restriction to terms that do not compute, which is then relaxed by semantic means. On the side of systems for certified programming such as F^{\star} , the type system ensures that types only depend on pure and terminating terms.

Thus, the recurring idea is to introduce restrictions on the dependency in order to establish an encapsulation of effects. In our approach, we seek a principled description of this idea by developing the concept of *semantic value* (thinkables, linears) which arose from foundational considerations [56], [102], [93] and whose relevance was highlighted in recent works [80], [99]. The novel aspect of our approach is to seek a proper extension of type theory which would provide foundations for a classical type theory with axiom of choice in the style of Herbelin [66], but which moreover could be generalised to effects other than just control by exploiting an abstract and adaptable notion of semantic value.

3.3.2. Intuitionistic depolarisation

In our view, the common idea that evaluation order does not matter for pure and termination computations should serve as a bridge between our proposals for dependent types in the presence of effects and traditional type theory. Building on the previous goal, we aim to study the relationship between semantic values, purity, and parametricity theorems [101], [58]. Our goal is to characterise parametricity as a form of intuitionistic *depolarisation* following the method by which the first game model of full linear logic was given (Melliès [86], [87]). We have two expected outcomes in mind: enriching type theory with intensional content without losing its properties, and giving an explanation of the dependent types in the style of Idris and F^{\star} where purity- and termination-checking play a role.

3.3.3. Developing the rewriting theory of calculi with effects

An integrated type theory with effects requires an understanding of evaluation order from the point of view of rewriting. For instance, rewriting properties can entail the decidability of some conversions, allowing the automation of equational reasoning in types [27]. They can also provide proofs of computational consistency (that terms are not all equivalent) by showing that extending calculi with new constructs is conservative [104]. In our approach, the λ -calculus is replaced by a calculus modelling the evaluation in an abstract machine [51]. We have shown how this approach generalises the previous semantic and proof-theoretic approaches [33], [79], [81], and overcomes their shortcomings [94].

One goal is to prove computational consistency or decidability of conversions purely using advanced rewriting techniques following a technique introduced in [104]. Another goal is the characterisation of weak reductions: extensions of the operational semantics to terms with free variables that preserve termination, whose iteration is equivalent to strong reduction [28], [54]. We aim to show that such properties derive from generic theorems of higher-order rewriting [110], so that weak reduction can easily be generalised to richer systems with effects.

3.3.4. Direct models and categorical coherence

Proof theory and rewriting are a source of *coherence theorems* in category theory, which show how calculations in a category can be simplified with an embedding into a structure with stronger properties [84], [75]. We aim to explore such results for categorical models of effects [79], [50]. Our key insight is to consider the reflection between *indirect and direct models* [56], [93] as a coherence theorem: it allows us to embed the traditional models of effects into structures for which the rewriting and proof-theoretic techniques from the previous section are effective.

Building on this, we are further interested in connecting operational semantics to 2-category theory, in which a second dimension is traditionally considered for modelling conversions of programs rather than equivalences. This idea has been successfully applied for the λ -calculus [72], [68] but does not scale yet to more realistic models of computation. In our approach, it has already been noticed that the expected symmetries coming from categorical dualities are better represented, motivating a new investigation into this long-standing question.

3.3.5. Models of effects and resources

The unified theory of effects and resources [50] prompts an investigation into the semantics of safe and automatic resource management, in the style of Modern C++ and Rust. Our goal is to show how advanced semantics of effects, resources, and their combination arise by assembling elementary blocks, pursuing the methodology applied by Melliès and Tabareau in the context of continuations [88]. For instance, by combining control flow (exceptions, return) with linearity allows us to describe in a precise way the “Resource Acquisition Is Initialisation” idiom in which the resource safety is ensured with scope-based destructors. A further step would be to reconstruct uniqueness types and borrowing using similar ideas.

3.4. Language extensions for the scaling of proof assistants

The development of tools to construct software systems that respect a given specification is a major challenge of current and future research in computer science. Certified programming with dependent types has recently attracted a lot of interest, and Coq is the *de facto* standard for such endeavours, with an increasing number of users, pedagogical resources, and large-scale projects. Nevertheless, significant work remains to be done to make Coq more usable from a software engineering point of view. The Gallinette team proposes to make progress on three lines of work: (i) the development of gradual certified programming, (ii) the integration of imperative features and object polymorphism in Coq, and (iii) the development of robust tactics for proof engineering for the scaling of formalised libraries.

3.4.1. Gradual Certified Programming

One of the main issues faced by a programmer starting to internalise in a proof assistant code written in a more permissive world is that type theory is constrained by a strict type discipline which lacks flexibility. Concretely, as soon that you start giving more a precise type/specification to a function, the rest of the code interacting with this functions needs to be more precise too. To address this issue, the Gallinette team will put strong efforts into the development of gradual typing in type theory to allow progressive integration of code that comes from a more permissive world.

Indeed, on the way to full verification, programmers can take advantage of a gradual approach in which some properties are simply asserted instead of proven, subject to dynamic verification. Tabareau and Tanter have made preliminary progress in this direction [106]. This work, however, suffers from a number of limitations, the most important being the lack of a mechanism for handling the possibility of runtime errors within Coq. Instead of relying on axioms, this project will explore the application of Section 3.3 to embed effects in Coq.

This way, instead of postulating axioms for parts of the development that are too hard/marginal to be dealt with, the system adds dynamic checks. Then, after extraction, we get a program that corresponds to the initial program but with dynamic check for parts that have not been proven, ensuring that the program will raise an error instead of going outside its specification.

This will yield new foundations of gradual certified programming, both more expressive and practical. We will also study how to integrate previous techniques with the extraction mechanism of Coq programs to OCaml, in order to exploit the exception mechanism of OCaml.

3.4.2. Imperative features and object polymorphism in the Coq proof assistant

3.4.2.1. Imperative features.

Abstract data types (ADTs) become useful as the size of programs grows since they provide for a modular approach, allowing abstractions about data to be expressed and then instantiated. Moreover, ADTs are natural concepts in the calculus of inductive constructions. But while it is easy to declare an ADT, it is often difficult to implement an efficient one. Compare this situation with, for example, Okasaki's purely functional data structures [96] which implement ADTs like queues in languages with imperative features. Of course, Okasaki's queues enforce some additional properties for free, such as persistence, but the programmer may prefer to use and to study a simpler implementation without those additional properties. Also in certified symbolic computation (see 3.5.3), an efficient functional implementation of ADTs is often not available, and efficiency is a major challenge in this area. Relying on the theoretical work done in 3.3, we will equip Coq with imperative features and we will demonstrate how they can be used to provide efficient implementations of ADTs. However, it is also often the case that imperative implementation are hard-to-reason-on, requiring for instance the use of separation logic. But in that case, we could take benefice of recent works on integration of separation logic in the Coq proof assistant and in particular the Iris project <http://iris-project.org/>.

3.4.2.2. Object polymorphism.

Object-oriented programming has evolved since its foundation based on the representation of computations as an exchange of messages between objects. In modern programming languages like Scala, which aims at a synthesis between object-oriented and functional programming, object-orientation concretely results in the use of hierarchies of interfaces ordered by the subtyping relation and the definition of interface implementations that can interoperate. As observed by Cook and Aldrich [48], [31], interoperability can be considered as the essential feature of objects and is a requirement for many modern frameworks and ecosystems: it means that two different implementations of the same interface can interoperate.

Our objective is to provide a representation of object-oriented programs, by focusing on subtyping and interoperability.

For subtyping, the natural solution in type theory is coercive subtyping [82], as implemented in Coq, with an explicit operator for coercions. This should lead to a shallow embedding, but has limitations: indeed, while it allows subtyping to be faithfully represented, it does not provide a direct means to represent union and intersection types, which are often associated with subtyping (for instance intersection types are present in Scala). A more ambitious solution would be to resort to subsumptive subtyping (or semantic subtyping [55]): in its more general form, a type algebra is extended with boolean operations (union, intersection, complementing) to get a boolean algebra with operators (the original type constructors). Subtyping is then interpreted as the natural partial order of the boolean algebra.

We propose to use the type class machinery of Coq to implement semantic subtyping for dependent type theory. Using type class resolution, we can emulate inference rules of subsumptive subtyping without modifying Coq internally. This has also another advantage. As subsumptive subtyping for dependent types should be undecidable in general, using type class resolution allows for an incomplete yet extensible decision procedure.

3.4.3. Robust tactics for proof engineering for the scaling of formalised libraries

When developing certified software, a major part of the effort is spent not only on writing proof scripts, but on *rewriting* them, either for the purpose of code maintenance or because of more significant changes in the base

definitions. Regrettably, proof scripts suffer more often than not from a bad programming style, and too many proof developers casually neglect the most elementary principles of well-behaved programmers. As a result, many proof scripts are very brittle, user-defined tactics are often difficult to extend, and sometimes even lack a clear specification. Formal libraries are thus generally very fragile pieces of software. One reason for this unfortunate situation is that proof engineering is very badly served by the tools currently available to the users of the Coq proof assistant, starting with its tactic language. One objective of the Gallinette team is to develop better tools to write proof scripts.

Completing and maintaining a large corpus of formalised mathematics requires a well-designed tactic language. This language should both accommodate the possible specific needs of the theories at stake, and help with diagnostics at refactoring time. Coq's tactic language is in fact two-leveled. First, it includes a basic tactic language, to organise the deductive steps in a proof script and to perform the elementary bureaucracy. Its second layer is a meta-programming language, which allows user to defined their own new tactics at toplevel. Our first direction of work consists in the investigation of the appropriate features of the *basic tactic language*. For instance, the design of the Ssreflect tactic language, and its support for the small scale reflection methodology [61], has been a key ingredient in at least two large scale formalisation endeavours: the Four Colour Theorem [60] and of the Odd Order Theorem [59]. Building on our experience with the Ssreflect tactic language, we will contribute to the ongoing work on the basic tactic language for Coq. The second objective of this task is to contribute to the design of a *typed tactic language*. In particular, we will build on the work of Ziliani and his collaborators [109], extending it with reasoning about the effects that tactics have on the "state of a proof" (e.g. number of sub-goals, metavariables in context). We will also develop a novel approach for incremental type checking of proof scripts, so that programmers gain access to a richer discovery- engineering interaction with the proof assistant.

3.5. Practical experiments

The first three axes of the EPC Gallinette aim at developing a new generation of proof assistants. But we strongly believe that foundational investigations must go hand in hand with practical experiments. Therefore, we expect to benefit from existing expertise and collaborations in the team to experiment our extensions of Coq on real world developments. It should be noticed that those practical experiments are strongly guided by the deep history of research on software engineering of team members.

3.5.1. Certified Code Refactoring

In the context of refactoring of C programs, we intend to formalise program transformations that are written in an imperative style to test the usability of our addition of effects in the proof assistant. This subject has been chosen based on the competence of members of the team.

We are currently working on the formalisation of refactoring tools in Coq [44]. Automatic refactoring of programs in industrial languages is difficult because of the large number of potential interactions between language features that are difficult to predict and to test. Indeed, all available refactoring tools suffer from bugs : they fail to ensure that the generated program has the same behaviour as the input program. To cope with that difficulty, we have chosen to build a refactoring tool with Coq : a program transformation is written in the Coq programming language, then proven correct on all possible inputs, and then an OCaml executable program is generated by the platform. We rely on the CompCert C formalisation of the C language. CompCert is currently the most complete formalisation of an industrial language, which justifies that choice. We have three goals in that project :

- Build a refactoring tool that programmers can rely on and make it available in a popular platform (such as Eclipse, IntelliJ or Frama-C).
- Explore large, drastic program transformations such as replacing a design architecture for an other one, by applying a sequence of small refactoring operations (as we have done for Java and Haskell programs before [47], [43], [30]), while ensuring behaviour preservation.
- Explore the use of enhancements of proof systems on large developments. For instance, refactoring tools are usually developed in the imperative/object paradigm, so the extension of Coq with side effects or with object features proposed in the team can find a direct use-case here.

3.5.2. *Certified Constraint Programming*

We plan to make use of the internalisation of the object-oriented paradigm in the context of constraint programming. Indeed, this domain is made of very complex algorithms that are often developed using object-oriented programming (as it is the case for instance for CHOCO, which is developed in the Tasc Group at IMT Atlantique, Nantes). We will in particular focus on filtering algorithms in constraint solvers, for which research publications currently propose new algorithms with manual proofs. Their formalisation in Coq is challenging. Another interesting part of constraint solving to formalise is the part that deals with program generation (as opposed to extraction). However, when there are numerous generated pieces of code, it is not realistic to prove their correctness manually, and it can be too difficult to prove the correctness of a generator. So we intend to explore a middle path that consists in generating a piece of code along with its corresponding proof (script or proof term). A target application could be interval constraints (for instance Allen interval algebra or region connection calculus) that can generate thousands of specialised filtering algorithms for a small number of variables [36].

Finally, Rémi Douence has already worked (articles publishing [63], [97], [53], PhD Thesis advising [98]) with different members of the Tasc team. Currently, he supervises with Nicolas Beldiceanu the PhD Thesis of Ekaterina Arafailova in the Tasc team. She studies finite transducers to model time-series constraints [37], [35], [34]. This work requires proofs, manually done for now, we would like to explore when these proofs could be mechanised.

3.5.3. *Certified Symbolic Computation*

We will investigate how the addition of effects in the Coq proof assistant can facilitate the marriage of computer algebra with formal proofs. Computer algebra systems on one hand, and proof assistants on the other hand, are both designed for doing mathematics with the help of a computer, by the means of symbolic computations. These two families of systems are however very different in nature: computer algebra systems allow for implementations faithful to the theoretical complexity of the algorithms, whereas proof assistants have the expressiveness to specify exactly the semantic of the data-structures and computations.

Experiments have been run that link computer algebra systems with Coq [52], [42]. These bridges rely on the implementation of formal proof-producing core algorithms like normalisation procedures. Incidentally, they require non trivial maintenance work to survive the evolution of both systems. Other proof assistants like the Isabelle/HOL system make use of so-called reflection schemes: the proof assistant can produce code in an external programming language like SML, but also allows to import the values output by these extracted programs back inside the formal proofs. This feature extends the trusted base of code quite significantly but it has been used for major achievements like a certified symbolic/numeric ODE solver [69].

We would like to bring Coq closer to the efficiency and user-friendliness of computer algebra systems: for now it is difficult to use the Coq programming language so that certified implementations of computer algebra algorithms have the right, observable, complexity when they are executed inside Coq. We see the addition of effects to the proof assistant as an opportunity to ease these implementations, for instance by making use of caching mechanisms or of profiling facilities. Such enhancements should enable the verification of computation-intensive mathematical proofs that are currently beyond reach, like the validation of Helfgott's proof of the weak Goldbach conjecture [65].

MEXICO Project-Team

3. Research Program

3.1. Concurrency

Participants: Thomas Chatain, Philippe Dague, 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: Stefan Haar, Serge Haddad, Stefan Schwoon, Philippe Dague, Lina Ye.

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 [45], [46], [47], 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 [34]

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*. In 2019, a new property, manifestability, weaker than diagnosability (dual in some sense to opacity) has been studied in the context of automata and timed automata.

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 [46], [36], 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 [43], [49]). 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 [33], [39]). 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 [51], [52]. 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. Hybrid Systems

Participants: Philippe Dague, Lina Ye, Serge Haddad.

Hybrid systems constitute a model for cyber-physical systems which integrates continuous-time dynamics (modes) governed by differential equations, and discrete transitions which switch instantaneously from one mode to another. Thanks to their ease of programming, hybrid systems have been integrated to power electronics systems, and more generally in cyber-physical systems. In order to guarantee that such systems meet their specifications, classical methods consist in finitely abstracting the systems by discretization of the (infinite) state space, and deriving automatically the appropriate mode control from the specification using standard graph techniques.

Diagnosability of hybrid systems has also been studied through an abstraction / refinement process in terms of timed automata.

3.1.4. 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 [44].

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 [35]. 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 [53] 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.

3.2. Management of Quantitative Behavior

Participants: Thomas Chatain, Stefan Haar, Serge Haddad.

3.2.1. 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.

3.2.2. Probabilistic distributed Systems

Participants: Stefan Haar, Serge Haddad.

3.2.2.1. 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. [38], [31], 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 [48], 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 [40] 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*.

3.2.2.2. 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 [32]; 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.

3.2.3. 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 [37].
- 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 [42].
- 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 [54].

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 [6]); (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.

3.2.4. 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 [50] [41]. 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.

3.2.4.1. 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.

MOCQUA Team

3. Research Program

3.1. Quantum Computing

While it can be argued that the quantum revolution has already happened in cryptography [39] or in optics [38], quantum computers are far from becoming a common commodity, with only a few teams around the world working on a practical implementation. In fact, one of the most commonly known examples of a quantum computer, the D-Wave 2X System, defies the usual definition of a computer: it is not general-purpose, and can only solve (approximately) a very specific hardwired problem.

Most current prototypes of a quantum computer differ fundamentally on the hardware substrate, and it is quite hard to predict which solution will finally be adopted. The landscape of quantum programming languages is also constantly evolving. Comparably to compiler design, the foundation of quantum software therefore relies on an intermediate representation that is suitable for manipulation, easy to produce from software and easily encodable into hardware. The language of choice for this is the ZX-calculus.

Regardless of the actual model that will be accepted by the industry, it is becoming clear that some of the hurdles into scaling up quantum computers from a few qubits to very large arrays will remain. As an example, current implementations of quantum computers working on hundreds of qubits indeed are not able to form and maintain all possible forms of entanglement between qubits. This raises two questions. First, does this restrict the computational power, and the supposed advantage of the quantum computer over the classical computer? Second, how to ensure that a quantum program that was designed for a theoretical quantum computer will work on the practical implementations? This will be investigated, in particular by providing static analysis methods for evaluating a priori how much entanglement a quantum program needs.

3.2. Higher-Order Computing

While programs often operate on natural numbers or finite structures such as graphs or finite strings, they can also take functions as input. In that case, the program is said to perform higher-order computations, or to compute a higher-order functional. Functional programming or object-oriented programming are important paradigms allowing higher-order computations.

While the theory of computation is well developed for first-order programs, difficulties arise when dealing with higher-order programs. There are many non-equivalent ways of presenting inputs to such programs: an input function can be presented as a black-box, encoded in an infinite binary sequence, or sometimes by a finite description. Comparing those representations is an important problem. A particularly useful application of higher-order computations is to compute with infinite objects that can be represented by functions or symbolic sequences. The theory works well in many cases (to be precise, when these objects live in a topological space with a countable basis [42]), but is not well understood in other interesting cases. For instance, when the inputs are the second-order functionals (of type $(\mathbb{N} \rightarrow \mathbb{N}) \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$), the classical theory does not apply and many problems are still open.

3.3. Dynamical Systems

The most natural example of a computation with infinite precision is the simulation of a dynamical system. The underlying space might be \mathbb{R}^n in the case of the simulation of physical systems, or the Cantor space $\{0, 1\}^{\mathbb{Z}}$ in the case of discrete dynamical systems.

From the point of view of computation, the main point of interest is the link between the long-term behavior of a system and its initial configuration. There are two questions here: (a) predict the behavior, (b) design dynamical systems with some prescribed behavior. The first will be mainly examined through the angle of reachability and more generally control theory for hybrid systems.

The model of cellular automata will be of particular interest. This computational model is relevant for simulating complex global phenomena which emerge from simple interactions between simple components. It is widely used in various natural sciences (physics, biology, etc.) and in computer science, as it is an appropriate model to reason about errors that occur in systems with a great number of components.

The simulation of a physical dynamical system on a computer is made difficult by various aspects. First, the parameters of the dynamical systems are seldom exactly known. Secondly, the simulation is usually non exact: real numbers are usually represented by floating-point numbers, and simulations of cellular automata only simulate the behavior of finite or periodic configurations. For some chaotic systems, this means that the simulation can be completely irrelevant.

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 β -reduction or cut-elimination are merely particular forms of proof normalization. Functional programming is based on normalization [51], and normalization in different logics can justify the design of new and different functional programming languages [32].
- *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 [55], and different proof search strategies can be used to justify the design of new and different logic programming languages [54].

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 [34] 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 [60] 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 [57] 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 [58], 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 [53] [52] [56] [61], and on more recent work by Gacek, Miller, and Nadathur [41] [40]. The Parsifal team, along with our colleagues in Minneapolis, Canberra, Singapore, and Cachan, 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 λ -calculus and π -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* [35] [37] [38] [39] [43] [49] [50] 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 [44], [46] 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 category-theoretic 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, atomic flows, and combinatorial proofs

Proof nets graph-like presentations of sequent calculus 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 [42] for linear logic but also in Robinson's proof nets [59] 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., [48][2] leading to the notion of *atomic flow* and [45]). 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.

Combinatorial proofs [47] are another form syntax-independent proof presentation which separates the multiplicative from the additive behaviour of classical connectives.

The following research questions investigated by members of the Parsifal team:

- Finding (for classical and intuitionistic logic) a notion of canonical proof presentation that is deductive, i.e., can effectively be used for doing proof search.
- Studying the normalization of proofs using atomic flows and combinatorial proofs, as they simplify the normalization procedure for proofs in deep inference, and additionally allow to get new insights in the complexity of the normalization.
- Studying the size of proofs in the combinatorial proof formalism.

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 λ -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 λ -calculus instead relies on a single non-atomic operation, β -reduction, for which costs in terms of time and space are far from evident.

Nonetheless, it turns out that the number of β -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 β -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 [36] (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 [33].

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 λ -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.

PI.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 Gentsen [77] 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 [72], then by Howard and de Bruijn at the end of the 60’s [89], [109], 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 λ -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 [69], [70] stemmed out – a formalism that is both a logic and a programming language and that is at the source of the Coq system [107].

3.1.2. *Towards the calculus of constructions*

The λ -calculus, defined by Church [67], 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 λ -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 λ -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 [60].

To explain the Curry-Howard correspondence, it is important to distinguish between intuitionistic and classical logic: following Brouwer at the beginning of the 20th 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) λ -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 [98].

In 1985, Coquand and Huet [69], [70] in the Formel team of Inria-Rocquencourt explored an alternative approach based on Girard-Reynolds’ system F [78], [102]. 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 [71] 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

During 1984-2012 period, 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 was six years ago. 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 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 mainly by employees of Inria, the CNAM, and Paris Diderot.

In the last seven years, Hugo Herbelin and Matthieu Sozeau coordinated the development of the system, the official coordinator hat passed from Hugo to Matthieu in August 2016. The ecosystem and development model changed greatly during this period, with a move towards an entirely distributed development model, integrating contributions from all over the world. While the system had always been open-source, its development team was relatively small, well-knit and gathered regularly at Coq working groups, and many developments on Coq were still discussed only by the few interested experts.

The last years saw a big increase in opening the development to external scrutiny and contributions. This was supported by the “core” team which started moving development to the open GitHub platform (including since 2017 its bug-tracker [43] and wiki), made its development process public, starting to use public pull requests to track the work of developers, organising yearly hackatons/coding-sprints for the dissemination of expertise and developers & users meetings like the Coq Workshop and CoqPL, and, perhaps more anecdotally, retransmitting Coq working groups on a public YouTube channel.

This move was also supported by the hiring of Maxime Dénès in 2016 as an Inria research engineer (in Sophia-Antipolis), and the work of Matej Košík (2-year research engineer). Their work involved making the development process more predictable and streamlined and to provide a higher level of quality to the whole system. In se 2018, a second engineer, Vincent Laporte, was hired. Yves Bertot, Maxime Dénès and

Vincent Laporte are developing the Coq consortium, which aims to become the incarnation of the global Coq community and to offer support for our users.

Today, the development of Coq involves participants from the Inria project-teams pi.r2 (Paris), Marelle (Sophia-Antipolis), Toccata (Saclay), Gallinette (Nantes), Gallium (Paris), and Camus (Strasbourg), the LIX at École Polytechnique and the CRI Mines-ParisTech. Apart from those, active collaborators include members from MPI-Saarbrücken (D. Dreyer's group), KU Leuven (B. Jacobs group), MIT CSAIL (A. Chlipala's group, which hosted an Inria/MIT engineer, and N. Zeldovich's group), the Institute for Advanced Study in Princeton (from S. Awodey, T. Coquand and V. Voevodsky's Univalent Foundations program) and Intel (M. Soegtrop). The latest released version Coq 8.8.0 had 40 contributors (counted from the start of 8.8 development) and the upcoming Coq 8.9 has 54.

On top of the developer community, there is a much wider user community, as Coq is being used in many different fields. The [Software Foundations series](#), authored by academics from the USA, along with the reference Coq'Art book by Bertot and Castéran [61], the more advanced Certified Programming with Dependent Types book by Chlipala [66] and the recent [book](#) on the Mathematical Components library by Mahboubi, Tassi et al. provide resources for gradually learning the tool.

In the programming languages community, Coq is being taught in two summer schools, [OPLSS](#) and the [DeepSpec](#) summer school. For more mathematically inclined users, there are regular [Winter Schools](#) in Nice and in 2017 there was a [school](#) on the use of the Univalent Foundations library in Birmingham.

Since 2016, Coq also provides a central repository for Coq packages, the Coq opam archive, relying on the OCaml opam package manager and including around 250 packages contributed by users. It would be too long to make a detailed list of the uses of Coq in the wild. We only highlight four research projects relying heavily on Coq. The [Mathematical Components library](#) has its origins in the formal proof of the Four Colour Theorem and has grown to cover many areas of mathematics in Coq using the now integrated (since Coq 8.7) SSREFLECT proof language. The [DeepSpec](#) project is an NSF Expedition project led by A. Appel whose aim is full-stack verification of a software system, from machine-checked proofs of circuits to an operating system to a web-browser, entirely written in Coq and integrating many large projects into one. The ERC [CoqHoTT](#) project led by N. Tabareau aims to use logical tools to extend the expressive power of Coq, dealing with the univalence axiom and effects. The ERC [RustBelt](#) project led by D. Dreyer concerns the development of rigorous formal foundations for the Rust programming language, using the Iris Higher-Order Concurrent Separation Logic Framework in Coq.

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 language.

3.2.6. Documentation

Coq is a feature-rich system and requires extensive training in order to be used proficiently; current documentation includes the reference manual, the reference for the standard library, as well as tutorials, and related tooling [sphinx plugins, coqdoc]. The jsCoq tool allows writing interactive web pages where Coq programs can be embedded and executed.

3.2.7. Proof development infrastructure

Coq is used in large-scale proof developments, and provides users miscellaneous tooling to help with them: the `coq_makefile` and Dune build systems help with incremental proof-checking; the Coq OPAM repository contains a package index for most Coq developments; the CoqIDE, ProofGeneral, and VSCoq user interfaces are environments for proof writing; and the Coq's API does allow users to extend the system in many important ways. Among the current extensions we have QuickChik, a tool for property-based testing; STMCoq and CoqHammer integrating Coq with automated solvers; ParamCoq, providing automatic derivation of parametricity principles; MetaCoq for metaprogramming; Equations for dependently-typed programming; SerAPI, for data-centric applications; etc... This also includes the main open Coq repository living at Github.

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 (Ω 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 (Concoction at Rice and Colorado, Ynot at Harvard, Why in the ProVal team at Inria, Iris at MPI-Saarbrücken). 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 [79] 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 [96] and Reynolds [101] and started to be studied in an abstract way in the 80's by Felleisen *et al* [75], leading to Parigot's $\lambda\mu$ -calculus [99], 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 λ -calculus is Landin's SECD machine [95] and Krivine's abstract machine for call-by-name evaluation [92], [91]. 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 [76]: any side-effect expressible in monadic style (and this covers references, exceptions, states, dynamic bindings, ...) can be simulated in λ -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, ∞ -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 ∞ -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 [59]) 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 [74]).

The categorification process has also reached logic, with the introduction of homotopy type theory. After a preliminary result that had identified categorical structures in type theory [88], it has been observed recently that the so-called "identity types" are naturally equipped with a structure of ∞ -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 ∞ -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 [106], and independently by Burroni [64], 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 [93], [97], [80], [81], and beyond: Petri nets [83] and formal proofs of classical and linear logic have been expressed in this framework [82]. 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 [62].

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 works of Anick [57] and Squier [105], [104]: 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 [14], [15], [85], [86], [87].

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 [14] and algebras [32]. 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, corresponding each 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 [94], [58]). 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 [11], and in homological algebra [84][32].

STAMP Project-Team

3. Research Program

3.1. Theoretical background

The proof assistants that we consider provide both a programming language, where users can describe algorithms performing tasks in their domain of interest, and a logical language to reason about the programs, thus making it possible to ensure that the algorithms do solve the problems for which they were designed. trustability is gained because algorithms and logical statements provide multiple views of the same topic, thus making it possible to detect errors coming from mismatch between expected and established properties. The verification process is itself a logical process, where the computer can bring rigor in aligning expectations and guarantees.

The foundations of proof assistants rest on the very foundations of mathematics. As a consequence, all aspects of reasoning must be made completely explicit in the process of formally verifying an algorithm. All aspects of the formal verification of an algorithm are expressed in a discourse whose consistency is verified by the computer, so that unclear or intuitive arguments need to be replaced by precise logical inferences.

One of the foundational features on which we rely extensively is *Type Theory*. In this approach a very simple programming language is equipped with a powerful discipline to check the consistency of usage: types represent sets of data with similar behavior, functions represent algorithms mapping types to other types, and the consistency can be verified by a simple computer program, a *type-checker*. Although they can be verified by a simple program, types can express arbitrary complex objects or properties, so that the verification work lives in an interesting realm, where verifying proofs is decidable, but finding the proofs is undecidable.

This process for producing new algorithms and theorems is a novelty in the development of mathematical knowledge or algorithms, and new working methods must be devised for it to become a productive approach to high quality software development. Questions that arise are numerous. How do we avoid requiring human assistance to work on mundane aspects of proofs? How do we take advantage of all the progress made in automatic theorem proving? How do we organize the maintenance of ambitious corpora of formally verified knowledge in the long term?

To acquire hands-on expertise, we concentrate our activity on three aspects. The first one is foundational: we develop and maintain a library of mathematical facts that covers many aspects of algebra. In the past, we applied this library to proofs in group theory, but it is increasingly used for many different areas of mathematics and by other teams around the world, from combinatorics to elliptic cryptography, for instance. The second aspect is applicative: we develop a specific tool for proofs in cryptography, where we need to reason on the probability that opponents manage to access information we wish to protect. For this activity, we develop a specific proof system, relying on a wider set of automatic tools, with the objective of finding the tools that are well adapted to this domain and to attract users that are initially specialists in cryptography but not in formal verification. The third domain is robotics, as we believe that the current trend towards more and more autonomous robots and vehicles will raise questions of safety and trustability where formal verification can bring significant added value.

SUMO Project-Team

3. Research Program

3.1. Introduction

Since its creation in 2015, SUMO has successfully developed formal methods for large quantitative systems, in particular addressing verification, synthesis and control problems. Our current motivation is to expand this by putting emphasis on new concerns, such as algorithm efficiency, imprecision handling, and the more challenging objective of addressing incomplete or missing models. In the following we list a selection of detailed research goals, structured into four axes according to model classes: quantitative models, large systems, population models, and data-driven models. Some correspond to the pursuit of previously obtained results, others are more prospective.

3.2. Axis 1: Quantitative models

The analysis and control of quantitative models will remain at the heart of a large part of our research activities. In particular, we have two starting collaborative projects focusing on **timed models**, namely our ANR project TickTac and our collaboration with MERCE. The main expected outcome of TickTac is an open-source tool implementing the latest algorithms and allowing for quick prototyping of new algorithms. Several other topics will be explored in these collaborations, including robustness issues, game-theoretic problems, as well as the development of efficient algorithms, *e.g.* based on CEGAR approach or specifically designed for subclasses of automata (*e.g.* automata with few clocks and/or having a specific structure, as in [38]). Inspired by our collaboration with Alstom, we also aim at developing symbolic techniques for analysing non-linear timed models.

Stochastic models are another important focus for our research. On the one hand, we want to pursue our work on the optimization of non-standard properties for Markov decision processes, beyond the traditional verification questions, and explore *e.g.* long-run probabilities, and quantiles. Also, we aim at lifting our work on decisiveness from purely stochastic [36], [37] to non-deterministic and stochastic models in order to provide approximation schemes for the probability of (repeated) reachability properties in infinite-state Markov decision processes. On the other hand, in order to effectively handle large stochastic systems, we will pursue our work on approximation techniques. We aim at deriving simpler models, enjoying or preserving specific properties, and at determining the appropriate level of abstraction for a given system. One needs of course to quantify the approximation degrees (distances), and to preserve essential features of the original systems (explainability). This is a connection point between formal methods and the booming learning methods.

Regarding **diagnosis/opacity** issues, we will explore further the quantitative aspects. For diagnosis, the theory needs extensions to the case of incomplete or erroneous models, and to reconfigurable systems, in order to develop its applicability (see Sec. 3.6). There is also a need for non-binary causality analysis (*e.g.* performance degradations in complex systems). For opacity, we aim at quantifying the effort attackers must produce *vs* how much of a secret they can guess. We also plan to synthesize robust controllers resisting to sensor failures/attacks.

3.3. Axis 2: Large systems

Part of the background of SUMO is on the analysis and management of concurrent and modular/distributed systems, that we view as two main approaches to address state explosion problems. We will pursue the study of these models (including their quantitative features): verification of timed concurrent systems, robust distributed control of modular systems, resilient control to coalitions of attackers, distributed diagnosis, modular opacity analysis, distributed optimal planning, etc. Nevertheless, we have identified two new lines of effort, inspired by our application domains.

Reconfigurable systems. This is mostly motivated by applications at the convergence of virtualization techs with networking (Orange and Nokia PhDs). Software defined networks, either in the core (SDN/NFV) or at the edge (IoT) involve distributed systems that change structure constantly, to adapt to traffic, failures, maintenance, upgrades, etc. Traditional verification, control, diagnosis approaches (to mention only those) assume static and known models that can be handled as a whole. This is clearly insufficient here: one needs to adapt existing results to models that (sometimes automatically) change structure, incorporate new components/users or lose some, etc. At the same time, the programming paradigms for such systems (chaos monkey) incorporate resilience mechanisms, that should be considered by our models.

Hierarchical systems. Our experience with the regulation of subway lines (Alstom) revealed that large scale complex systems are usually described at a single level of granularity. Determining the appropriate granularity is a problem in itself. The control of such systems, with humans in the loop, can not be expressed at this single level, as tasks become too complex and require extremely skilled staff. It is rather desirable to describe models simultaneously at different levels of granularity, and to perform control at the appropriate level: humans in charge of managing the system by high level objectives, and computers in charge of implementing the appropriate micro-control sequences to achieve these tasks.

3.4. Axis 3: Population models

We want to step up our effort in parameterized verification of systems consisting of many identical components, so-called population models. In a nutshell our objectives summarize as "from Boolean to quantitative".

Inspired by our experience on the analysis of populations of yeasts, we aim at developing the quantitative analysis and control of population models, *e.g.* using Markov decision processes together with quantitative properties, and focusing on generating strategies with fast convergence.

As for broadcast networks, the challenge is to model the mobility of nodes (representing mobile ad hoc networks) in a faithful way. The obtained model should reflect on the one hand, the placement of nodes at a given time instant, and on the other hand, the physical movement of nodes over time. In this context, we will also use game theory techniques which allows one to study cooperative and conflictual behaviors of the nodes in the network, and to synthesize correct-by-design systems in adversarial environments.

As a new application area, we target randomized distributed algorithms. Our goal is to provide probabilistic variants of threshold automata [39] to represent fault-tolerant randomized distributed algorithms, designed for instance to solve the consensus problem. Most importantly, we then aim at developing new parameterized verification techniques, that will enable the automated verification of the correctness of such algorithms, as well as the assessment of their performances (in particular the expected time to termination).

In this axis, we will investigate whether fluid model checking and mean-field approximation techniques apply to our problems. More generally, we aim at a fruitful cross-fertilizing of these approaches with parameterized model-checking algorithms.

3.5. Axis 4: Data-driven models

In this axis, we will consider data-centric models, and in particular their application to crowd-sourcing. Many data-centric models such as Business Artifacts [40] orchestrate simple calls and answers to tasks performed by a single user. In a crowd-sourcing context, tasks are realized by pools of users, which may result in imprecise, uncertain and (partially) incompatible information. We thus need mechanisms to reconcile and fuse the various contributions in order to produce reliable information. Another aspect to consider concerns answers of higher-order: how to allow users to return intentional answers, under the form of a sub-workflow (coordinated set of tasks) which execution will provide the intended value. In the framework of the ANR Headwork we will build on formalisms such as GAG (guarded attribute grammars) or variants of business artifacts to propose formalisms adapted to crowd-sourcing applications, and tools to analyze them. To address imprecision, we will study techniques to handle fuzziness in user answers, will explore means to set incentives (rewards) dynamically, and to set competence requirements to guide the execution of a complex workflow, in order to achieve an objective with a desired level of quality.

In collaboration with Open Agora, CESPAs and University of Yaoundé (Cameroun) we intend to implement in the GAG formalism some elements of argumentation theory (argumentation schemes, speech acts and dialogic games) in order to build a tool for the conduct of a critical discussion and the collaborative construction of expertise. The tool would incorporate point of view extraction (using clustering mechanisms), amendment management and consensus building mechanisms.

3.6. Transversal concern: missing models

We are concerned with one important lesson derived from our involvement in several application domains. Most of our background gets in force as soon as a perfect model of the system under study is available. Then verification, control, diagnosis, test, etc. can mobilize a solid background, or suggest new algorithmic problems to address. In numerous situations, however, assuming that a model is available is simply unrealistic. This is a major bottleneck for the impact of our research. We therefore intend to address this difficulty, in particular for the following domains.

- Model building for diagnosis. As a matter of fact, diagnosis theory hardly touches the ground to the extent that complete models of normal behavior are rarely available, and the identification of the appropriate abstraction level is unclear. Knowledge of faults and their effects is even less accessible. Also, the actual implemented systems may differ significantly from behaviors described in the norms. One therefore needs a theory for incomplete and erroneous models. Besides, one is often less bothered by partial observations than drowned by avalanches of alerts when malfunctions occur. Learning may come to the rescue, all the more that software systems may be deployed in sandpits and damaged for experimentation, thus allowing the collection of masses of labeled data. Competition on that theme clearly comes from Machine Learning techniques.
- Verification of large scale software. For some verification problems like the one we address in the IPL HAC-Specis, one does not have access to a formal model of the distributed program under study, but only to executions in a simulator. Formal verification poses new problems due to the difficulties to capture global states, to master state space explosion by gathering and exploiting concurrency information.
- Learning of stochastic models. Applications in bioinformatics often lead to large scale models, involving numerous chains of interactions between chemical species and/or cells. Fine grain models can be very precise, but very inefficient for inference or verification. Defining the appropriate levels of description/abstraction, given the available data and the verification goals, remains an open problem. This cannot be considered as a simple data fitting problem, as elements of biological knowledge must be combined with the data in order to preserve explainability of the phenomena.
- Testing and learning timed models: during conformance testing of a black-box implementation against its formal specification, one wants to detect non-conformances but may also want to learn the implementation model. Even though mixing testing and learning is not new, this is more recent and challenging for continuous-time models.
- Process mining. We intend to extend our work on process discovery using Petri net synthesis [35] by using negative information (*e.g.* execution traces identified as outliers) and quantitative information (probabilistic or fuzzy sets of execution traces) in order to infer more robust and precise models.

TOCCATA Project-Team

3. Research Program

3.1. Research Program

3.1.1. Panorama of Deductive Verification

There are two main families of approaches for deductive verification. Methods in the first family build on top of mathematical proof assistants (e.g., Coq, Isabelle) in which both the model and the program are encoded; the proof that the program meets its specification is typically conducted in an interactive way using the underlying proof construction engine. Methods from the second family proceed by the design of standalone tools taking as input a program in a particular programming language (e.g., C, Java) specified with a dedicated annotation language (e.g., ACSL [49], JML [56]) and automatically producing a set of mathematical formulas (the *verification conditions*) which are typically proved using automatic provers (e.g., Z3 [70], Alt-Ergo [58], CVC4 [48]).

The first family of approaches usually offers a higher level of assurance than the second, but also demands more work to perform the proofs (because of their interactive nature) and makes them less easy to adopt by industry. Moreover, they generally do not allow to directly analyze a program written in a mainstream programming language like Java or C. The second kind of approaches has benefited in the past years from the tremendous progress made in SAT and SMT solving techniques, allowing more impact on industrial practices, but suffers from a lower level of trust: in all parts of the proof chain (the model of the input programming language, the VC generator, the back-end automatic prover), potential errors may appear, compromising the guarantee offered. Moreover, while these approaches are applied to mainstream languages, they usually support only a subset of their features.

3.1.2. Overall Goals of the Toccata Project

One of our original skills is 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. Indeed, an axis of research of Toccata is the development of languages, methods and tools that are themselves formally proved correct.

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 [47], various examples of applications of that approach [54], the use of the Gappa solver for proving numerical algorithms [68], an approach to take architectures and compilers into account when dealing with floating-point programs [55], [66]. We also contributed to the Handbook of Floating-Point Arithmetic [65]. 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 [52] [51]. Our experience led us to a conclusion that verification of numerical programs can benefit a lot from combining automatic and interactive theorem proving [53], [54], [59]. Verification of numerical programs is another main axis of Toccata.

Our scientific programme detailed below is structured into four axes:

1. Foundations and spreading of deductive program verification;
2. Reasoning on mutable memory in program verification;
3. Verification of Computer Arithmetic;
4. Spreading Formal Proofs.

Let us conclude with more general considerations about our agenda of the next four years: we want to keep on

- with general audience actions;
- industrial transfer, in particular through an extension of the perimeter of the ProofInUse joint lab.

3.2. Foundations and spreading of deductive program verification

Permanent researchers: S. Conchon, J.-C. Filliâtre, C. Marché, G. Melquiond, A. Paskevich

This axis covers the central theme of the team: deductive verification, from the point of view of its foundations but also our will to spread its use in software development. The general motto we want to defend is “deductive verification for the masses”. A non-exhaustive list of subjects we want to address is as follows.

- The verification of general-purpose algorithms and data structures: the challenge is to discover adequate invariants to obtain a proof, in the most automatic way as possible, in the continuation of the current VOCaL project and the various case studies presented in Axis 4 below.
- Uniform approaches to obtain correct-by-construction programs and libraries, in particular by automatic extraction of executable code (in OCaml, C, CakeML, etc.) from verified programs, and including innovative general methods like advanced ghost code, ghost monitoring, etc.
- Automated reasoning dedicated to deductive verification, so as to improve proof automation; improved combination of interactive provers and fully automated ones, proof by reflection.
- Improved feedback in case of proof failures: based on generation of counterexamples, or symbolic execution, or possibly randomized techniques à la quickcheck.
- Reduction of the trusted computing base in our toolchains: production of certificates from automatic proofs, for goal transformations (like those done by Why3), and from the generation of VCs

A significant part of the work achieved in this axis is related to the Why3 toolbox and its ecosystem, displayed on Figure 1. The boxes in red background correspond to the tools we develop in the Toccata team.

3.3. Reasoning on mutable memory in program verification

Permanent researchers: J.-C. Filliâtre, C. Marché, G. Melquiond, A. Paskevich

This axis concerns specifically the techniques for reasoning on programs where aliasing is the central issue. It covers the methods based on type-based alias analysis and related memory models, on specific program logics such as separation logics, and extended model-checking. It concerns the application on analysis of C or C++ codes, on Ada codes involving pointers, but also concurrent programs in general. The main topics planned are:

- The study of advanced type systems dedicated to verification, for controlling aliasing, and their use for obtaining easier-to-prove verification conditions. Modern typing system in the style of Rust, involving ownership and borrowing, will be considered.
- The design of front-ends of Why3 for the proofs of programs where aliasing cannot be fully controlled statically, via adequate memory models, aiming in particular at extraction to C; and also for concurrent programs.
- The continuation of fruitful work on concurrent parameterized systems, and its corresponding specific SMT-based model-checking.
- Concurrent programming on weak memory models, on one hand as an extension of parameterized systems above, but also in the specific context of OCaml multicore (<http://ocamlabs.io/doc/multicore.html>).
- In particular in the context of the ProofInUse joint lab, design methods for Ada, C, C++ or Java using memory models involving fine-grain analysis of pointers. Rust programs could be considered as well.

3.4. Verification of Computer Arithmetic

Permanent researchers: S. Boldo, C. Marché, G. Melquiond

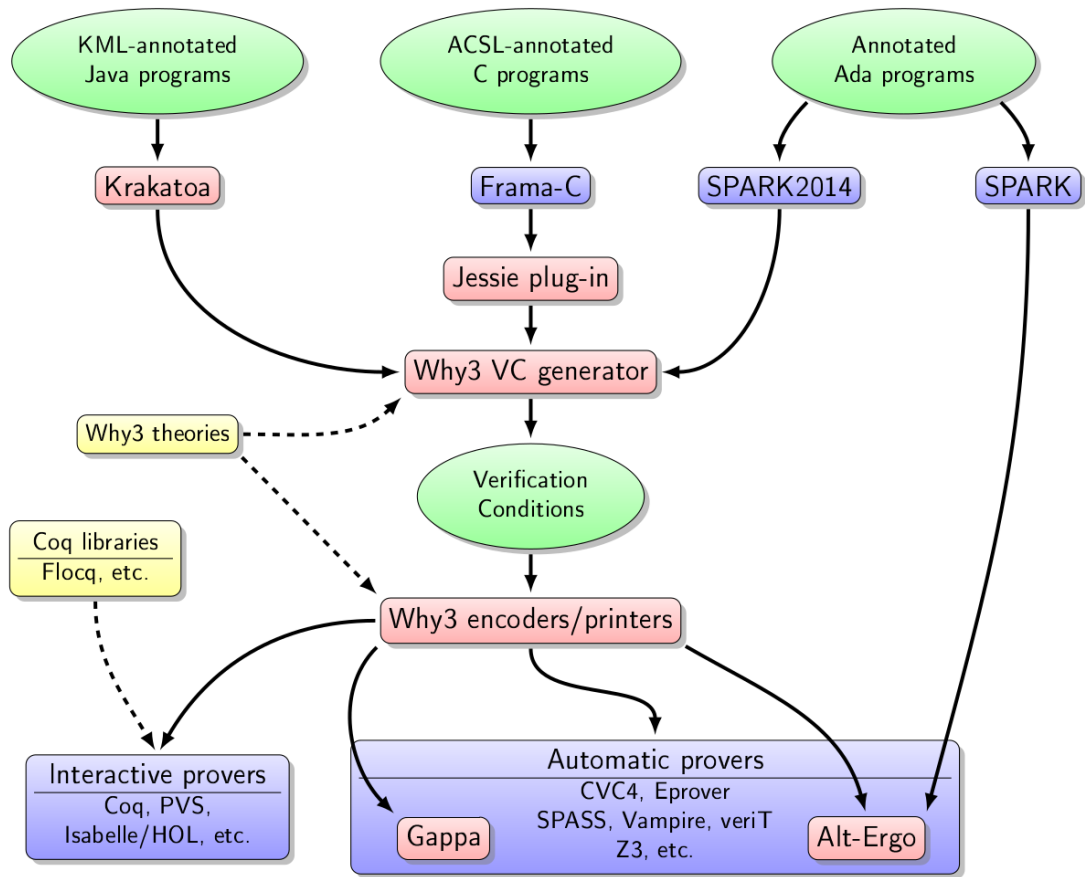


Figure 1. The Why3 ecosystem

We of course want to keep this axis which is a major originality of Toccata. The main topics of the next 4 years will be:

- Fundamental studies concerning formalization of floating-point computations, algorithms, and error analysis. Related to numerical integration, we will develop the relationships between mathematical stability and floating-point stability of numerical schemes.
- A significant effort dedicated to verification of numerical programs written in C, Ada, C++. This involves combining specifications in real numbers and computation in floating-point, and underlying automated reasoning techniques with floating-point numbers and real numbers. A new approach we have in mind concerns some variant of symbolic execution of both code and specifications involving real numbers.
- We have not yet studied embedded systems. Our approach is first to tackle numerical filters. This requires more results on fixed-point arithmetic and a careful study of overflows.
- Also a specific focus on arbitrary precision integer arithmetic, in the continuation of the ongoing PhD thesis of R. Rieu-Helft.

3.5. Spreading Formal Proofs

Permanent researchers: S. Boldo, S. Conchon, J.-C. Filliâtre, C. Marché, G. Melquiond, A. Paskevich

This axis covers applications in general. The applications we currently have in mind are:

- Hybrid Systems, i.e., systems mixing discrete and continuous transitions. This theme covers many aspects such as general techniques for formally reasoning of differential equations, and extending SMT-based reasoning. The challenge is to support both abstract mathematical reasoning and concrete program execution (e.g., using floating-point representation). Hybrid systems will be a common effort with other members of the future laboratory joint with LSV of ENS Cachan.
- Applied mathematics, in the continuation of the current efforts towards verification of Finite Element Method. It has only been studied in the mathematical point of view during this period. We plan to also consider their floating-point behavior and a demanding application is that of molecular simulation exhibited in the new EMC2 project. The challenge here is both in the mathematics to be formalized, in the numerical errors that have never been studied (and that may be huge in specific cases), and in the size of the programs, which requires that our tools scale.
- Continuation of our work on analysis of shell scripts. The challenge is to be able to analyze a large number of scripts (more than 30,000 in the corpus of Debian packages installation scripts) in an automatic manner. An approach that will be considered is some form of symbolic execution.
- Explore proof tools for mathematics, in particular automated reasoning for real analysis (application: formalization of the weak Goldbach conjecture), and in number theory.
- Obtain and distribute verified OCaml libraries, as expected outcome of the VOCaL project.
- Formalization of abstract interpretation and WP calculi: in the continuation of the former project Verasco, and an ongoing project proposal joint with CEA List. The difficulty of achieving full verification of such tools will be mitigated by use of certificate techniques.

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 the SPASS [10] **workbench**. It currently consists of one of the leading automated theorem provers for first-order logic based on the superposition calculus [56] and a theory solver for linear arithmetic.

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 SMT⁰ solver that combines decision procedures for different fragments of 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 difficult or impossible to express 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, i.e. by embedding decision procedures 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 on the development of methods and tools for the formal proof of TLA⁺ [66] 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.

⁰Satisfiability Modulo Theories [58]

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* [55], [57], [70] 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.

CIDRE Project-Team

3. Research Program

3.1. Our perspective

For many aspects of our daily lives, we rely heavily on computer systems, many of which are based on massively interconnected devices that support a population of interacting and cooperating entities. As these systems become more open and complex, accidental and intentional failures become much more frequent and serious. We believe that the purpose of attacks against these systems is expressed at a high level (compromise of sensitive data, unavailability of services). However, these attacks are often carried out at a very low level (exploitation of vulnerabilities by malicious code, hardware attacks).

The CIDRE team is specialized in the defense of computer systems. We argue that to properly protect these systems we must have a complete understanding of the attacker's concrete capabilities. In other words, **to defend properly we must understand the attack.**

The CIDRE team therefore strives to have a global expertise in information systems: from hardware to distributed architectures. Our objective is to highlight security issues and propose preventive or reactive countermeasures in widely used and privacy-friendly systems.

3.2. Attack Comprehension

An attack on a computer system begins with the exploitation of one or more vulnerabilities of that system. Generally speaking, a vulnerability can be a software bug or a misconfiguration that can be exploited by the attacker to perform unauthorized actions. Exploiting a vulnerability leads to a use of the system according to a case not foreseen in its specification, implementation or configuration. This puts the system in an inconsistent state allowing the attacker to divert the use of the system in his or her own interest.

The systems we use are large, interconnected, constantly evolving and, therefore, are likely to retain many vulnerabilities; their security depends on our ability to update them quickly when new threats are discovered. It is thus necessary to understand how the attacker has compromised the system: what vulnerabilities he has exploited, what actions he has conducted, where he is located in the system. It is essential to study statically the malicious code used by the attacker. It is also important to be able to study it dynamically to be able to replay attacks on demand.

Ideally, we should be ahead of the attacker and therefore imagine new ways to attack. In addition, we believe it is necessary to improve the feedback to the expert by allowing him to quickly understand the progress of an attack. The first step before being able to offer secure systems is to understand and measure the real capabilities of the attacker.

Our first research axis therefore aims at highlighting both the effective attacker's means and the way an attack unfolds and spreads.

In this context, we are particularly interested in

- **highlighting attacks** on the micro-architecture that affect software security
- **providing expert support**
 - to analyze malicious code
 - to quickly investigate an intrusion on a system monitored by an intrusion detection system

3.3. Attack Detection

An attack is generally composed of several steps. During a first approach step the attacker enters the system, locates the target and makes itself persistent. Then, in a second step, the payload of the attack is effectively launched, leading to a violation of the security policy (attacks against confidentiality, integrity, or availability of OS, applications, services, or data).

The objective of intrusion detection is to be able to detect the attacker, ideally during the first step of the attack. To do this, intrusion detection systems (IDS) are based on probes that continuously monitor the system. These probes report events to a core engine that decide whether or not to alert the expert.

Intrusion detection systems are important for all systems handling sensitive data that may be accessible to a malicious agent. They are especially crucial for low-level systems that provide essential support services to other systems. They are essential in inter-connected systems that are designed to last a long time and are difficult to update.

3.4. Attack Resistance

The first two axes of the team allowed us to measure the concrete technical means of the attacker. We claim that the attacker can always avoid the measures put in place to secure a system. We believe that another way to offer more secure systems is to take into account from the design phase that these systems will operate in the presence of an omnipotent attacker. The last research axis of the CIDRE team is focused on offering systems that are resistant to attackers, *i.e.* they can provide the expected services even in the presence of an attacker.

To achieve this goal, we explore two approaches:

- deceptive security
- malicious behavior tolerance

In the notion of *deceptive security* we group together all the approaches that aim to mislead the active attacker in a system in order to deceive him on the exact nature of his target. These approaches can slow down the attacker or lead him to abandon his attack.

Finally, we contribute to the design of architectures or services relying on the collaboration of entities that is not affected by the minority presence of malicious entities. These architectures or services are based on the collaboration of a set of nodes that are not affected by the presence in minority of malicious nodes.

COMETE Project-Team

3. Research Program

3.1. Probability and information theory

Participants: Konstantinos Chatzikokolakis, Catuscia Palamidessi, Marco Romanelli, Anna Pazzi.

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: Frank Valencia, Santiago Quintero.

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,

DATASPHERE Team

3. Research Program

3.1. Dynamics of digital transformations

The research program of the Datasphere team aims at understanding the transformations induced by digital systems on socio-economic and socio-ecological organizations. These transformations are very broad and impact a large part of society. Understanding these changes is very ambitious and would require much more resources than those of the team. Interactions with other teams in other disciplines is thus of strategic importance. The research directions we have worked in and will continue to in the coming years are the following.

- The legal and strategic implications of the development of networks, the growing global interdependencies, and the increase of digital flows beyond control.
- The geopolitics of digital systems, data flows and cyber control, the raise of new strategic imbalances, and digital powers (US, China, Russia, etc.)
- The structural consequences of the translation of governance to digital actors, their inclusion into diplomatic forums, and the weakening of sovereignty over territories.

3.2. Foundations of digital economy

- The economy of intermediation and the progressive control of all two-sided and multi-sided markets by remote digital platforms.
- The methodologies for assessing the strategic value of data and evaluating its leverage for the political economy.
- The analysis of Online Advertisement/tracking ecosystems.

3.3. Ecosystems and Anthropocene

- The interdependencies of natural ecosystems and socio-economic systems, and the role of digital systems on measuring and controlling the global natural/social system.
- The role of digital actors in the adaptation and mitigation of climate change.
- The information economy of planetary challenges related to global warming, biodiversity, health monitoring.

3.4. Large scale graph analysis

- Community analysis and extraction, spectral methods.
- Manifold based approaches to large scale graph analysis, optimal transport.
- Information/rumor/fake news propagation in social networks.

3.5. Cyberstrategy

- Geopolitics of BGP
- Cyberstrategy of infrastructures
- Internet content control

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 must 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 [53].

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. [52]. 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 out-sourced, as well as sanitisation of data in social networks. We expect that many privacy and anonymity properties may be modelled as particular observational equivalences in process calculi [48], or indistinguishability between cryptographic games [3]; 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 state explosion problems. For instance, the use of Horn clause resolution techniques requires dedicated resolution methods [41] [44]. Another example is unification modulo equational theory, which is a key technique in several tools, e.g. [51]. 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 [46], which is used in several tools, e.g., *Akiss* [44], *Maude-NPA* [51] and *Tamarin* [54]. Another example is the notion of asymmetric unification [50] 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 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 a notion of *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 finding 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 include *composition results* that allow one to design protocols in a modular way [47], [45]. 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 the 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 work on the automated synthesis of the orchestrator or monitors for enforcing the security goals. These problems require the study of new classes of automata that communicate with structured messages.

3.3.2. New protocol design

We also design new protocols. Application areas that seem of particular importance are:

- External hardware devices such as security APIs that allow for flexible key management, including key revocation, and their integration in security protocols. The security *fiasco* of the PKCS#11 standard [43], [49] witnesses the need for new protocols in this area.
- Election systems that provide strong security guarantees. We have been working (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 PROSECCO) 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, 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
- F*: a new language that enables the verification of cryptographic applications

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 [53]; it has motivated a serious research effort on the formal analysis of cryptographic protocols, starting with [49] 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* [44]. ProVerif can handle a wide variety of cryptographic primitives, defined by rewrite rules or by some equations, and prove a wide variety of security properties: secrecy [42], [30], correspondences (including authentication) [43], and observational equivalences [41]. 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 has long been the only tool that proves equivalences for an unbounded number of sessions. (Maude-NPA in 2014 and Tamarin in 2015 adopted ProVerif's approach to proving equivalences.)

Using ProVerif, it is now possible to verify large parts of industrial-strength protocols, such as TLS [36], Signal [51], JFK [31], and Web Services Security [40], 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 120 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 [46], results which were extended by formal analysis work using a Dolev-Yao style model of the standard [47]. 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 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 [55], that models typically ignore. So even a protocol has been proved secure in theory, 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# [39]. 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. Our current focus is on designing and implementing the programming language F* [57], [34], [52], in collaboration with Microsoft Research. F* (pronounced F star) is an ML-like functional programming language aimed at program verification. Its type system includes polymorphism, dependent types, monadic effects, refinement types, and a weakest precondition calculus. Together, these features allow expressing precise and compact specifications for programs, including functional correctness and security properties. The F* type-checker aims to prove that programs meet their specifications using a combination of SMT solving and interactive proofs[23]. Programs written in F* can be translated to efficient OCaml, F#, or C for execution [54]. The main ongoing use case of F* is building a verified, drop-in replacement for the whole HTTPS stack in Project Everest [37] (a larger collaboration with Microsoft Research). This includes a verified implementation of TLS 1.2 and 1.3 [38] and of the underlying cryptographic primitives [58].

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 designed the automatic tool CryptoVerif, which generates proofs by sequences of games. We already applied it to important protocols such as TLS [36] and Signal [51] but more work is still needed in order to develop this approach, so that it is easier to apply to more protocols. We also design and implement techniques for proving implementations

of protocols secure in the computational model. In particular, CryptoVerif can generate implementations from CryptoVerif specifications that have been proved secure [45]. We plan to continue working on this approach.

A different approach is to directly verify cryptographic applications in the computational model by typing. A recent work [50] 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 for Program Verification

F* [57], [34] is a verification system for effectful programs developed collaboratively by Inria and Microsoft Research. It puts together the automation of an SMT-backed deductive verification tool with the expressive power of a proof assistant based on dependent types. After verification, F* programs can be extracted to efficient OCaml, F#, or C code [54]. This enables verifying the functional correctness and security of realistic applications. F*'s type system includes dependent types, monadic effects, refinement types, and a weakest precondition calculus. Together, these features allow expressing precise and compact specifications for programs, including functional correctness and security properties. The F* type-checker aims to prove that programs meet their specifications using a combination of SMT solving and interactive proofs. The main ongoing use case of F* is building a verified, drop-in replacement for the whole HTTPS stack in Project Everest. This includes verified implementations of TLS 1.2 and 1.3 [38] and of the underlying cryptographic primitives [58].

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 compilation chains for realistic low-level programming languages (the C language, and Low* a safe subset of C embedded in F* for verification [54]). These compilation chains 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 compilation chains target a tagged architecture [35], which associates a metadata tag to each word and efficiently propagates and checks tags according to software-defined rules. We hope to experimentally evaluate and carefully optimize the efficiency of our secure compilation chains on realistic workloads and standard benchmark suites. We are also using property-based testing and formal verification to provide high confidence that our compilation chains are indeed secure. Formally, we are constructing machine-checked proofs of a new security criterion we call robustly safe compilation, which is defined as the preservation of safety properties even against an adversarial context [32], [33]. This strong criterion complements compiler correctness and ensures that no machine-code attacker can do more harm to securely compiled components than a component already could with respect to a secure source-level semantics.

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 [48] and TS* [56], 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. We have seen considerable progress in identity with the UnlimitID design and with messaging via the IETF MLS effort, with new work on blockchain technology underway.

TAMIS Project-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.