Activity Report 2011

Section Software

Edition: 2012-03-22
ALGORITHMS, CERTIFICATION, AND CRYPTOGRAPHY

1. ALGORITHMS Project-Team ................................................................. 5
2. ARENAIRE Project-Team ................................................................. 6
3. CARAMEL Project-Team ................................................................. 10
4. CASCADE Project-Team ................................................................. 13
5. GALAAD Project-Team ................................................................. 15
6. GEOMETRICA Project-Team ......................................................... 17
7. LFANT Project-Team ................................................................. 19
8. SALSA Project-Team ................................................................. 21
9. SECRET Project-Team (section vide) ............................................... 22
10. TANC Project-Team ................................................................. 23
11. VEGAS Project-Team ................................................................. 25

ARCHITECTURE AND COMPILING

12. ALF Project-Team ........................................................................ 27
13. CAIRN Project-Team .................................................................... 30
14. CAMUS Team ............................................................................... 35
15. COMPSYS Project-Team .............................................................. 38

EMBEDDED AND REAL TIME SYSTEMS

16. AOSTE Project-Team .................................................................... 43
17. DART Project-Team ..................................................................... 46
18. ESPRESSO Project-Team ............................................................... 48
19. LICIT Exploratory Action (section vide) ......................................... 52
20. PARKAS Team ............................................................................... 53
21. POP ART Project-Team ................................................................. 56
22. S4 Project-Team ........................................................................... 61
23. TRIO Project-Team ....................................................................... 62
24. VASY Project-Team ....................................................................... 65
25. VERTECS Project-Team ................................................................. 68

PROGRAMS, VERIFICATION AND PROOFS

26. ABSTRACTION Project-Team ......................................................... 70
27. ATEAMS Project-Team ................................................................. 74
28. CARTE Project-Team .................................................................... 78
29. CASSIS Project-Team .................................................................... 79
30. CELTIQUE Project-Team ............................................................... 82
31. COMETE Project-Team ................................................................. 83
32. CONTRAINTES Project-Team ....................................................... 85
33. FORMES Team ............................................................................... 88
34. GALLIUM Project-Team ............................................................... 91
35. MARELLE Project-Team ............................................................... 92
36. MEXICO Project-Team ................................................................. 93
37. MOSCOVA Project-Team .............................................................. 94
<table>
<thead>
<tr>
<th>Project-Team</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>38. PAREO Project-Team</td>
<td>97</td>
</tr>
<tr>
<td>39. PARSIFAL Project-Team</td>
<td>99</td>
</tr>
<tr>
<td>40. PLR2 Project-Team</td>
<td>100</td>
</tr>
<tr>
<td>41. PROVAL Project-Team</td>
<td>102</td>
</tr>
<tr>
<td>42. SECSI Project-Team</td>
<td>106</td>
</tr>
<tr>
<td>43. TASC Project-Team</td>
<td>108</td>
</tr>
<tr>
<td>44. TYPICAL Project-Team</td>
<td>110</td>
</tr>
<tr>
<td>45. VERIDIS Team</td>
<td>111</td>
</tr>
</tbody>
</table>
4. Software

4.1. Algolib

The Algolib library is a set of Maple routines that have been developed in the project for more than 15 years. Several parts of it have been incorporated into the standard library of Maple, but the most up-to-date version is always available for free from our web pages http://algo.inria.fr/libraries/. This library provides: tools for combinatorial structures (the combstruct package), including enumeration, random or exhaustive generation, generating functions for a large class of attribute grammars; tools for linear difference and differential equations (the gfun package), which have received a very positive review in Computing Reviews and have been incorporated in N. Sloane's superseeker at Bell Labs; tools for systems of multivariate linear operators, definite sums and integrals (the Mgfun package), including Gröbner bases in Ore algebras, that also treat commutative polynomials and have been the standard way to solve polynomial systems in Maple for a long period (although the user would not notice it); Mgfun has also been chosen at Risc (Linz) as the basis for their package Desing, tools for expansions in general asymptotic scales, which make it possible to handle in a transparent and automatic way the problems of finding the proper scale for an expansion and of dealing with the indefinite cancellation problem (the MultiSeries package).

4.2. Mathematics on the Web

We also provide access to our work to scientists who are not using Maple or any other computer algebra system in the form of automatically generated encyclopedias available on the web. The Encyclopedia of Combinatorial Structures at http://algo.inria.fr/ecs/ thus contains more than 1000 combinatorial structures for which generating functions, enumeration sequences, recurrences, and asymptotic approximations have been computed automatically. It gets more than 16,000 hits per month. The Dynamic Dictionary of Mathematical Functions (DDMF) at http://ddmf.msr-inria.inria.fr/ gathers several dozens of special functions for which identities, guaranteed high-precision numerical evaluations, power-series and asymptotic expansions, graphs, ...are generated automatically and on the user’s request, starting from a linear differential equation and its initial conditions. The underlying symbolic algorithms and implementations are those of gfun and Mgfun. All the production process being automated, the difficult and expensive step of checking each formula individually is suppressed. A nice specificity of this encyclopedia is its interactivity: the approximations values (numbers, series) are not bound to a statically set precision, rather, the user can fill in the precision he wants in a form, before clicking to ask for a refined identity to be generated, then displayed. This interactivity is based on a tool DynaMoW at http://ddmf.msr-inria.inria.fr/DynaMoW/ (for Dynamic Mathematics on the Web) that we develop as well. This is an Ocaml library that simultaneously controls external symbolic calculations and webpage generation at the same time. Being available on the web, the DDMF also plays the role of a showcase for part of the packages developed in our project. It is a successor of our former Encyclopedia of Special Functions at http://algo.inria.fr/esf/.
ARENAIRE Project-Team

5. Software

5.1. Introduction

Arénaire proposes various software and hardware realizations that are accessible from the web page http://www.ens-lyon.fr/LIP/Arenaire/Ware/. We describe below only those which progressed in 2011.

![Diagram of relationships between some Arénaire developments.](image)

**Figure 1. Relationships between some Arénaire developments.**

5.2. FloPoCo

**Participants:** Florent Dinechin [correspondant], Bogdan Pasca, Laurent-Stéphane Didier.

The purpose of the FloPoCo project is to explore the many ways in which the flexibility of the FPGA target can be exploited in the arithmetic realm. FloPoCo is a generator of operators written in C++ and outputting synthesizable VHDL automatically pipelined to an arbitrary frequency.

In 2011, FloPoCo was turned into a library which can be used as a back-end to high-level synthesis tools. An expression parser that generates a complete pipeline was also added for this context. The integer multiplier and floating-point adder were rewritten, and several new operators were added, including a floating-point power operator, and novel operators for integer and floating-point division by a constant.

Versions 2.2.0, 2.2.1, and 2.3.0 were released in 2011.

5.3. GNU MPFR

Participants: Vincent Lefèvre [correspondant], Paul Zimmermann.

GNU MPFR is an efficient multiple-precision floating-point library with well-defined semantics (copying the good ideas from the IEEE-754 standard), in particular correct rounding in 5 rounding modes. GNU MPFR provides about 80 mathematical functions, in addition to utility functions (assignments, conversions...). Special data (Not a Number, infinities, signed zeros) are handled like in the IEEE-754 standard.

MPFR was one of the main pieces of software developed by the old SPACES team at Loria. Since late 2006, with the departure of Vincent Lefèvre to Lyon, it has become a joint project between the Caramel (formerly SPACES then CACAO) and the Arénaire project-teams. MPFR has been a GNU package since 26 January 2009. GNU MPFR 3.0.1 was released on 4 April 2011 and GNU MPFR 3.1.0 was released on 3 October 2011.

The main improvements are the generic tests in a reduced exponent range, the possibility to include the mpfr.h header file several times while still supporting optional functions, and, for the developers, the choice of the native type for the exponent (and various corrections related to these features).

URL: http://www.mpfr.org/

- ACM: D.2.2 (Software libraries), G.1.0 (Multiple precision arithmetic), G.4 (Mathematical software).
- AMS: 26-04 Real Numbers, Explicit machine computation and programs.
- APP: no longer applicable (copyright transferred to the Free Software Foundation).
- License: LGPL version 3 or later.
- Type of human computer interaction: C library, callable from C or other languages via third-party interfaces.
- OS/Middleware: any OS, as long as a C compiler is available.
- Required library or software: GMP.
- Programming language: C.
- Documentation: API in texinfo format (and other formats via conversion); algorithms are also described in a separate document.

5.4. Exhaustive Tests for the Correct Rounding of Mathematical Functions

Participant: Vincent Lefèvre.

The search for the worst cases for the correct rounding (hardest-to-round cases) of mathematical functions (exp, log, sin, cos, etc.) in a fixed precision (mainly double precision) using Lefèvre’s algorithm is implemented by a set of utilities written in Perl, with calls to Maple/intpakX for computations on intervals and with C code generation for fast computations. It also includes a client-server system for the distribution of intervals to be tested and for tracking the status of intervals (fully tested, being tested, aborted).
These programs are run on the LIP network via Grid Engine (SGE). In June 2011, the SGE configuration was changed by the system administrator so that SIGSTOP/SIGCONT signals are sent to the jobs, allowing several users to use SGE at the same time. These signals make Maple crash (segmentation fault), and the Perl scripts needed to be improved to handle these crashes gracefully (by restarting the computations when need be, etc.). This SGE change made other problems appear, such as when the client is first stopped by SGE and is then killed by SGE (without being woke up by SGE), it cannot do its usual clean-up; workarounds were tried, but without success.

The above problems also made an inconsistency in the client-server protocol appear. The validity of the results was not affected, but the protocol had to be redesigned.

5.5. CGPE: Code Generation for Polynomial Evaluation

Participants: Christophe Mouilleron, Claude-Pierre Jeannerod.

The CGPE project, developed with Guillaume Revy (DALI research team, Université de Perpignan and LIRMM laboratory), aims at generating C codes for fast and certified polynomial evaluation, given various accuracy and architectural constraints. Several improvements for this tool, based on the addition of constraints in the first step of the generation process, were proposed in the PhD thesis of Ch. Mouilleron [12]. These improvements have been implemented, thus allowing us to reduce the whole generation time by about 50% on average.

- ACM: D.2.2 (Software libraries), G.4 (Mathematical software).
- Recommended library or software: MPFI or Gappa.
- License: CeCILL
- Type of human computer interaction: command-line interface
- OS/Middleware: Unix
- Required library or software: Xerces-C++ XML Parser library and MPFR
- Programming Language: C++
- Status: beta
- Documentation: available in html format on URL: http://cgpe.gforge.inria.fr/

5.6. FLIP: Floating-point Library for Integer Processors

Participants: Claude-Pierre Jeannerod, Jingyan Jourdan-Lu.

FLIP is a C library for the efficient software support of binary32 IEEE 754-2008 floating-point arithmetic on processors without floating-point hardware units, such as VLIW or DSP processors for embedded applications. The current target architecture is the VLIW ST200 family from STMicroelectronics (especially the ST231 cores). This year, we have mostly worked on improving the design and implementation of the following operators with correct rounding “to nearest even”: DP2 (fused dot product in dimension two) and sum of two squares. The impact of the DP2 operator has been evaluated on the UTDSP benchmark, and on some kernels speed-ups of 1.46 have been observed. On the other hand, specializing DP2 to a sum of squares brings a speed-up of 2.

URL: http://flip.gforge.inria.fr/

- ACM: D.2.2 (Software libraries), G.4 (Mathematical software)
- AMS: 26-04 Real Numbers, Explicit machine computation and programs.
- APP: IDDN.FR.001.230018.S.A.2010.000.10000
- License: CeCILL v2
- Type of human computer interaction: C library callable, from any C program.
- OS/Middleware: any, as long as a C compiler is available.
- Required library or software: none.
- Programming language: C
5.7. SIPE: Small Integer Plus Exponent

Participant: Vincent Lefèvre.

SIPE (Small Integer Plus Exponent) is a C header file providing a fast floating-point arithmetic with correct rounding to the nearest in very small precision. Implemented operations are the addition, subtraction, multiplication, FMA, and minimum/maximum/comparison functions (of the signed numbers or in magnitude). SIPE has been written for exhaustive tests of simple algorithms in small precision in order to prove results or find conjectures (which could then be proved). In 2011, a research report was written about SIPE [62], including documentation and proof of the implementation; some bugs were fixed at the same time.

- ACM: D.2.2 (Software libraries), G.4 (Mathematical software).
- AMS: 26-04 Real Numbers, Explicit machine computation and programs.
- License: LGPL version 2.1 or later.
- Type of human computer interaction: C header file.
- OS/Middleware: any OS.
- Required library or software: GCC compiler.
- Programming language: C.
- Documentation: Research report RR-7832. [62]
- URL: http://www.vinc17.net/software/sipe.h
CARAMEL Project-Team

5. Software

5.1. Introduction

A major part of the research done in the CARAMEL team is published within software. On the one hand, this enables everyone to check that the algorithms we develop are really efficient in practice; on the other hand, this gives other researchers — and us of course — basic software components on which they — and we — can build other applications.

5.2. GNU MPFR

Participant: Paul Zimmermann [contact].

GNU MPFR is one of the main pieces of software developed by the CARAMEL team. Since end 2006, with the departure of Vincent Lefèvre to ENS Lyon, it has become a joint project between CARAMEL and the ARENAIRE project-team (INRIA Grenoble - Rhône-Alpes). GNU MPFR is a library for computing with arbitrary precision floating-point numbers, together with well-defined semantics, and is distributed under the LGPL license. All arithmetic operations are performed according to a rounding mode provided by the user, and all results are guaranteed correct to the last bit, according to the given rounding mode.

Several software systems use GNU MPFR, for example: the GCC and GFortran compilers; the SAGE computer algebra system; the KDE calculator Abakus by Michael Pyne; CGAL (Computational Geometry Algorithms Library) developed by the Geometrica project-team (INRIA Sophia Antipolis - Méditerranée); Gappa, by Guillaume Melquiond; Sollya, by Sylvain Chevillard, Mioara Joldeș and Christoph Lauter; Genius Math Tool and the GEL language, by Jiri Lebl; Giac/Xcas, a free computer algebra system, by Bernard Parisse; the iRRAM exact arithmetic implementation from Norbert Müller (University of Trier, Germany); the Magma computational algebra system; and the Wcalc calculator by Kyle Wheeler.

The main developments in 2011 are the release of version 3.0.1 in April, and the release of version 3.1.0 (the “canard à l’orange” release) in October. The main changes in GNU MPFR 3.1.0 are the following: thread local storage (TLS) support is now detected automatically, the squaring and division routines got a major speed up thanks to Mulders’ algorithm [20], and a new divide-by-zero exception was introduced. Note that the automatic TLS support did exhibit several compiler bugs (http://www.loria.fr/~zimmerma/software/compilerbugs.html). We had a developers meeting in January 13-14, and in August GNU MPFR was presented at the GNU Hackers Meeting in Paris.

5.3. MPC

Participant: Paul Zimmermann [contact].

MPC is a floating-point library for complex numbers, which is developed on top of the GNU MPFR library, and distributed under the LGPL license. It is co-written with Andreas Enge (LFANT project-team, INRIA Bordeaux - Sud-Ouest). A complex floating-point number is represented by $x + iy$, where $x$ and $y$ are real floating-point numbers, represented using the GNU MPFR library. The MPC library provides correct rounding on both the real part $x$ and the imaginary part $y$ of any result. MPC is used in particular in the TRIP celestial mechanics system developed at IMCCE (Institut de Mécanique Céleste et de Calcul des Éphémérides), and by the Magma computational number theory system.

A new version, MPC 0.9 (Epilobium montanum), was released in February 2011, with new functions, some speed-ups, a few bug fixes, and a logging feature for debugging. Since version 4.5 of GCC, released in May 2010, GCC requires MPC to compute constant complex expressions at compile-time (constant folding), like it requires GNU MPFR since GCC 4.3.
5.4. GMP-ECM

Participants: Cyril Bouvier, Paul Zimmermann [contact].

GMP-ECM is a program to factor integers using the Elliptic Curve Method. Its efficiency comes both from the use of the GMP library, and from the implementation of state-of-the-art algorithms. GMP-ECM contains a library (LIBECM) in addition to the binary program (ECM). The binary program is distributed under GPL, while the library is distributed under LGPL, to allow its integration into other non-GPL software. The Magma computational number theory software and the SAGE computer algebra system both use LIBECM.

During his internship of 4 months in 2011, Cyril Bouvier developed a version of ECM for GPUs. The code was written for NVIDIA GPUs using CUDA. First, the code was written for all NVIDIA cards, and later, it was optimized for the newer Fermi cards. As there is no modular arithmetic library (like GMP) available for GPU, it was necessary to implement a modular arithmetic using array of unsigned integers from scratch, while taking into account constraints of GPU programming. The code was optimized for factoring 1024 bits integers. For now, the code has a throughput roughly four times bigger than GMP-ECM on one core. This code is not yet fully integrated in GMP-ECM but is available in the GMP-ECM svn repository.

The implementation of ECM on GPU uses a different algorithm for scalar multiplication (the binary ladder instead of PRAC) and a different parametrization. This new approach was implemented for CPU in GMP-ECM. It results in a speedup in the execution time of GMP-ECM for finding big factors (more than 20 digits). It will be integrated in the next release of GMP-ECM.

5.5. Finite fields

Participants: Pierrick Gaudry, Emmanuel Thomé [contact].

mpF_q is (yet another) library for computing in finite fields. The purpose of mpF_q is not to provide a software layer for accessing finite fields determined at runtime within a computer algebra system like Magma, but rather to give a very efficient, optimized code for computing in finite fields precisely known at compile time. mpF_q is not restricted to a finite field in particular, and can adapt to finite fields of any characteristic and any extension degree. However, one of the targets being the use in cryptology, mpF_q somehow focuses on prime fields and on fields of characteristic two.

mpF_q’s ability to generate specialized code for desired finite fields differentiates this library from its competitors. The performance achieved is far superior. For example, mpF_q can be readily used to assess the throughput of an efficient software implementation of a given cryptosystem. Such an evaluation is the purpose of the “EBats” benchmarking tool. mpF_q entered this trend in 2007, establishing reference marks for fast elliptic curve cryptography: the authors improved over the fastest examples of key-sharing software in genus 1 and 2, both over binary fields and prime fields. These timings are now comparison references for other implementations.

The library’s purpose being the generation of code rather than its execution, the working core of mpF_q consists of roughly 18,000 lines of Perl code, which generate most of the C code. mpF_q is distributed at http://mpfq.gforge.inria.fr/.

The mpF_q library has undergone no change in 2011.

5.6. gf2x

Participants: Pierrick Gaudry, Emmanuel Thomé [contact], Paul Zimmermann.

GF2X is a software library for polynomial multiplication over the binary field, developed together with Richard Brent (Australian National University, Canberra, Australia). It holds state-of-the-art implementation of fast algorithms for this task, employing different algorithms in order to achieve efficiency from small to large operand sizes (Karatsuba and Toom-Cook variants, and eventually Schönhage’s or Cantor’s FFT-like algorithms). GF2X takes advantage of specific processor instructions (SSE, PCLMULQDQ).

1 http://www.ecrypt.eu.org/ebats/
The current version of GF2X is 1.0, released in 2010 under the GNU GPL. Since 2009, GF2X can be use as an auxiliary package for the widespread software library NTL, as of version 5.5. There has been no update of GF2X in 2011, but the software is still maintained. An LGPL-licensed portion of GF2X is also part of the CADO-NFS software package.

### 5.7. CADO-NFS

**Participants:** Jérémie Detrey, Pierrick Gaudry, Lionel Muller, Emmanuel Thomé [contact], Paul Zimmermann.

CADO-NFS is a program to factor integers using the Number Field Sieve algorithm (NFS), developed in the context of the ANR-CADO project (November 2006 to January 2010).

NFS is a complex algorithm which contains a large number of sub-algorithms. The implementation of all of them is now complete, but still leaves some places to be improved. Compared to existing implementations, the CADO-NFS implementation is already a reasonable player. Several factorizations have been completed using our implementations.

Since 2009, the source repository of CADO-NFS is publicly available for download. On October 28, 2011, the 1.1 version of CADO-NFS has been released. Several improvements to the program have been obtained, in practically all areas of the program. In particular, the polynomial selection code described by Thorsten Kleinjung at the CADO workshop in 2008 is now used within CADO-NFS, together with some efficient root-sieve code written by Shi Bai (Australian National University). Overall, CADO-NFS keeps improving its competitiveness over alternative code bases. The lattice siever now supports a sieving region of $2^{31} (I = 16)$; its code has been deeply reorganized to allow future improvements that we have in mind but were difficult to implement (proper sieving of powers, sieve according to the parities of the coordinates of the location). The executables in the linear algebra step have been reorganized (now using shared libraries), and now use a code generation mechanism built on top of the MPFQ library for the arithmetic parts. This is in particular meant to ease future accommodation of other base fields that GF(2), which is a requirement for adapting CADO-NFS to discrete logarithm computation. The MPI performance of the linear algebra code has been optimized. Some experimental scripts have been added to execute the sieve on a cluster; these scripts rely on the OAR job scheduler being used, and exploit its "besteffort" mode.

The largest factorizations performed by CADO-NFS in 2011 are a 170-digit integer from aliquot sequence 660 and a 171-digit integer from aliquot sequence 966.

### 5.8. AVIsogenies

**Participants:** Gaëtan Bisson [contact], Romain Cosset.

AVISOGENIES (Abelian Varieties and Isogenies) is a Magma package for working with abelian varieties, with a particular emphasis on explicit isogeny computation; it has been publicly released under the LGPLv2+ license in 2010.

Its prominent feature is the computation of $(\ell, \ell)$-isogenies between Jacobian varieties of genus-2 hyperelliptic curves over finite fields; practical runs have involved values of $\ell$ in the hundreds. It also provides procedures for exploring and drawing isogeny graphs, and for computing various complex-multiplication-related structures, such as Shimura’s gothic $C$ group.

In 2011, two incremental versions have been released. They provide the following new features: the characteristic 2 is now supported, and the complete addition laws of [23] have been implemented.

The package can be obtained at [http://avisogenies.gforge.inria.fr/](http://avisogenies.gforge.inria.fr/).
5. Software

5.1. MitMTool

Participants: Charles Bouillaguet, Patrick Derbez, Pierre-Alain Fouque.

The purpose of MitMTool is to look for guess-and-determine and meet-in-the-middle attacks on AES and AES-based constructions. This tool allows us to improve known attacks on round-reduced versions of AES, on the LEX stream-cipher and the PELICAN Message Authentication Code and on fault attack on AES. Basically, it solves the problem to find all the solutions of a linear system of equations on the variables $x$ and $S(x)$ where $S$ is an inert function. The tool allows to compute the complexity of some good attack as well as the C code of the attack. We verify that the complexity estimates are accurate using experiments. We also use it to find one solution of the system for chosen-key differential attacks. There are mainly two tools: the first one only looks for guess-and-determine attack and tries to propagate some knowledge and guesses value when it cannot find automatically the value of some variable. The second tool uses the technique of the first tool and more advanced technique to take into account attacks with memory that use the meet-in-the-middle attack.

5.2. ProVerif

Participants: Bruno Blanchet, Vincent Cheval.

ProVerif (www.proverif.ens.fr) is an automatic security protocol verifier, in the formal model (so called Dolev–Yao model). In this model, cryptographic primitives are considered as black boxes. This protocol verifier is based on an abstract representation of the protocol by Horn clauses. Its main features are:

- It can handle many different cryptographic primitives, including shared- and public-key cryptography (encryption and signatures), hash functions, and Diffie–Hellman key agreements, specified both as rewrite rules or as equations.
- It can handle an unbounded number of sessions of the protocol (even in parallel) and an unbounded message space. This result has been obtained thanks to some well-chosen approximations. This means the verifier can give false attacks, but if it claims that the protocol satisfies some property, then the property is actually satisfied. ProVerif also provides attack reconstruction: when it cannot prove a property, it tries to reconstruct an attack, that is, an execution trace of the protocol that falsifies the desired property.

The ProVerif verifier can prove the following properties:

- secrecy (the adversary cannot obtain the secret);
- authentication and more generally correspondence properties, of the form "if an event has been executed, then other events have been executed as well";
- strong secrecy (the adversary does not see the difference when the value of the secret changes);
- equivalences between processes that differ only by terms;

ProVerif has been used by researchers for studying various kinds of protocols, including electronic voting protocols, certified email protocols, and zero-knowledge protocols. It has been used as a back-end for the tool Tulafale implemented at Microsoft Research Cambridge, which verifies web services protocols. It has also been used as a back-end for verifying implementations of protocols in F# (a dialect of ML included in .NET), by Microsoft Research Cambridge and the joint INRIA-Microsoft research center.

ProVerif is freely available on the web, at www.proverif.ens.fr, under the GPL license.
5.3. CryptoVerif

Participants: Bruno Blanchet, David Cadé.

CryptoVerif (www.cryptoverif.ens.fr) is an automatic protocol prover sound in the computational model. In this model, messages are bitstrings and the adversary is a polynomial-time probabilistic Turing machine. CryptoVerif can prove:

- secrecy;
- correspondences, which include in particular authentication.

CryptoVerif provides a generic mechanism for specifying the security assumptions on cryptographic primitives, which can handle in particular symmetric encryption, message authentication codes, public-key encryption, signatures, hash functions, Diffie-Hellman key agreement.

The generated proofs are proofs by sequences of games, as used by cryptographers. These proofs are valid for a number of sessions polynomial in the security parameter, in the presence of an active adversary. CryptoVerif can also evaluate the probability of success of an attack against the protocol as a function of the probability of breaking each cryptographic primitive and of the number of sessions (exact security).

CryptoVerif is still at a rather early stage of development, but it has already been used for a study of Kerberos in the computational model. It is also used as a back-end for verifying implementations of protocols in F# at Microsoft Research Cambridge and at the joint INRIA-Microsoft research center.

CryptoVerif is freely available on the web, at www.cryptoverif.ens.fr, under the CeCILL-B license.
5. Software

5.1. Mathemagix, a free computer algebra environment

Participants: Bernard Mourrain, Angelos Mantzaflaris.

http://www.mathemagix.org/

MATHEMAGIX is a free computer algebra system which consists of a general purpose interpreter, which can be used for non-mathematical tasks as well, and efficient modules on algebraic objects. It includes the development of standard libraries for basic arithmetic on dense and sparse objects (numbers, univariate and multivariate polynomials, power series, matrices, etc., based on FFT and other fast algorithms). These developments are based on C++, offer generic programming without losing effectiveness, via the parameterization of the code (template) and the control of their instantiations.

The language of the interpreter is imperative, strongly typed and high level. A compiler of this language is available. A special effort has been put on the embedding of existing libraries written in other languages like C or C++. An interesting feature is that this extension mechanism supports template types, which automatically induce generic types inside Mathemagix. Connections with GMP, MPFR for extended arithmetic, LAPACK for numerical linear algebra are currently available in this framework.

The project aims at building a bridge between symbolic computation and numerical analysis. It is structured by collaborative software developments of different groups in the domain of algebraic and symbolic-numeric computation.

In this framework, we are working more specifically on the following components:

- REALROOT: a set of solvers using subdivision methods to isolate the roots of polynomial equations in one or several variables; continued fraction expansion of roots of univariate polynomials; Bernstein basis representation of univariate and multivariate polynomials and related algorithms; exact computation with real algebraic numbers, sign evaluation, comparison, certified numerical approximation.

- SHAPE: tools to manipulate curves and surfaces of different types including parameterised, implicit with different type of coefficients; algorithms to compute their topology, intersection points or curves, self-intersection locus, singularities, ...

These packages are integrated from the former library SYNAPS (SYmbolic Numeric APplicationS) dedicated to symbolic and numerical computations. There are also used in the algebraic-geometric modeler AXEL.

Collaborators: Grégoire Lecerf, Joris van der Hoeven and Philippe Trébuchet.

5.2. Axel, a geometric modeler for algebraic objects

Participants: Angelos Mantzaflaris, Bernard Mourrain, Meriadeg Perrinel.

http://axel.inria.fr

We are developing a software called AXEL (Algebraic Software-Components for gEometric modeLing) dedicated to algebraic methods for curves and surfaces. Many algorithms in geometric modeling require a combination of geometric and algebraic tools. Aiming at the development of reliable and efficient implementations, AXEL provides a framework for such combination of tools, involving symbolic and numeric computations.
The software contains data structures and functionalities related to algebraic models used in geometric modeling, such as polynomial parameterisation, B-Spline, implicit curves and surfaces. It provides algorithms for the treatment of such geometric objects, such as tools for computing intersection points of curves or surfaces, detecting and computing self-intersection points of parameterized surfaces, implicitization, for computing the topology of implicit curves, for meshing implicit (singular) surfaces, etc.

The developments related to isogeometric analysis in Exciting have been integrated as dedicated plugins. Optimisation techniques and solvers for partial differential equations developed by R. Duvigneau (OPALE) have been connected.

A new version of the algebraic-geometric modelers is developed by Meriadeg Perinnel to connect it to the platform Dtk in order to provide a better modularity and a better interface to existing computation facilities and geometric rendering interface.

The package is distributed as binary packages for Linux as well as for MacOSX. It is hosted at the INRIA’s gforge (http://gforge.inria.fr) and referenced by many leading software websites such as http://apple.com. The first version of the software has been downloaded more than 15000 times, since it is available. Collaboration with Gang Xu (Hangzhou Dianzi University, China), Julien Wintz (Dream).

### 5.3. Maple packages for differential algebra and algebraic invariants

**Participant:** Evelyne Hubert.

- The Maple package **diffalg** is a collection of routines to handle systems of polynomial differential equations and inequations. The functionalities include differential elimination, expansion of the solutions into formal power series and analysis of singular solutions. The underlying theory and terminology belongs to differential algebra.

  Collaborators: François Boulier and François Lemaire from University of Lille.

- The Maple **AIDA** package is a collection of routines to explore algebra of differential invariants: computation of generating sets of invariants, rewritings, syzygies, and their differential analogues. The package builds on the Maple libraries Groebner, Vessiot and diffalg.

5. Software

5.1. CGAL, the Computational Geometry Algorithms Library

Participants: Pierre Alliez, Jean-Daniel Boissonnat, Olivier Devillers, Monique Teillaud, Mariette Yvinec.

With the collaboration of Hervé Brönnimann, Manuel Caroli, Pedro Machado Manhães de Castro, Frédéric Cazals, Frank Da, Christophe Delage, Andreas Fabri, Julia Flötotto, Philippe Guigue, Michael Hemmer, Samuel Hornus, Menelaos Karavelas, Sébastien Loriot, Abdelkrim Mebarki, Naceur Meskini, Andreas Meyer, Sylvain Pion, Marc Pouget, François Rebﬁat, Laurent Rineau, Laurent Saboret, Stéphane Tayeb, Radu Ursu, and Camille Wormser. http://www.cgal.org

CGAL is a C++ library of geometric algorithms and data structures. Its development has been initially funded and further supported by several European projects (CGAL, GALIA, ECG, ACS, AIM@SHAPE) since 1996. The long term partners of the project are research teams from the following institutes: INRIA Sophia Antipolis - Méditerranée, Max-Planck Institut Saarbrücken, ETH Zürich, Tel Aviv University, together with several others. In 2003, CGAL became an Open Source project (under the LGPL and QPL licenses), and it also became commercialized by GEOMETRY FACTORY, a company Born of INRIA founded by Andreas Fabri.

The aim of the CGAL project is to create a platform for geometric computing supporting usage in both industry and academia. The main design goals are genericity, numerical robustness, efficiency and ease of use. These goals are enforced by a review of all submissions managed by an editorial board. As the focus is on fundamental geometric algorithms and data structures, the target application domains are numerous: from geological modeling to medical images, from antenna placement to geographic information systems, etc.

The CGAL library consists of a kernel, a list of algorithmic packages, and a support library. The kernel is made of classes that represent elementary geometric objects (points, vectors, lines, segments, planes, simplices, isotetic boxes, circles, spheres, circular arcs...), as well as affine transformations and a number of predicates and geometric constructions over these objects. These classes exist in dimensions 2 and 3 (static dimension) and d (dynamic dimension). Using the template mechanism, each class can be instantiated following several representation modes: one can choose between Cartesian or homogeneous coordinates, use different types to store the coordinates, and use reference counting or not. The kernel also provides some robustness features using some specifically-devised arithmetic (interval arithmetic, multi-precision arithmetic, static filters...).

A number of packages provide geometric data structures as well as algorithms. The data structures are polygons, polyhedra, triangulations, planar maps, arrangements and various search structures (segment trees, d-dimensional trees...). Algorithms are provided to compute convex hulls, Voronoi diagrams, Boolean operations on polygons, solve certain optimization problems (linear, quadratic, generalized of linear type). Through class and function templates, these algorithms can be used either with the kernel objects or with user-defined geometric classes provided they match a documented interface.

Finally, the support library provides random generators, and interfacing code with other libraries, tools, or file formats (ASCII files, QT or LEDA Windows, OpenGL, Open Inventor, Postscript, Geomview...). Partial interfaces with Python, SCILAB and the Ipe drawing editor are now also available.

GEOMETRICA is particularly involved in general maintenance, in the arithmetic issues that arise in the treatment of robustness issues, in the kernel, in triangulation packages and their close applications such as alpha shapes, in meshes... Three researchers of GEOMETRICA are members of the CGAL Editorial Board, whose main responsibilities are the control of the quality of CGAL, making decisions about technical matters, coordinating communication and promotion of CGAL.

CGAL is about 700,000 lines of code and supports various platforms: GCC (Linux, Mac OS X, Cygwin...), Visual C++ (Windows), Intel C++. A new version of CGAL is released twice a year, and it is downloaded about 10000 times a year. Moreover, CGAL is directly available as packages for the Debian, Ubuntu and Fedora Linux distributions.
More numbers about CGAL: there are now 13 editors in the editorial board, with approximately 20 additional developers. The user discussion mailing-list has more than 1000 subscribers with a relatively high traffic of 5-10 mails a day. The announcement mailing-list has more than 3000 subscribers.
5. Software

5.1. Pari/Gp

Participants: Karim Belabas [correspondant], Bill Allombert, Henri Cohen, Andreas Enge.

http://pari.math.u-bordeaux.fr/

PARI/GP is a widely used computer algebra system designed for fast computations in number theory (factorisation, algebraic number theory, elliptic curves, ...), but it also contains a large number of other useful functions to compute with mathematical entities such as matrices, polynomials, power series, algebraic numbers, etc., and many transcendental functions.

- PARI is a C library, allowing fast computations.
- GP is an easy-to-use interactive shell giving access to the PARI functions.
- gp2c, the GP-to-C compiler, combines the best of both worlds by compiling GP scripts to the C language and transparently loading the resulting functions into GP; scripts compiled by gp2c will typically run three to four times faster.

2011 has seen the release of the next major stable version, 2.5, ending the 2.3 release series started in 2007.

- Version of PARI/GP: 2.5.0
- Version of gp2c: 0.0.7pl11
- License: GPL v2+
- Programming language: C

5.2. MPC

Participants: Andreas Enge [correspondant], Mickaël Gastineau, Philippe Théveny, Paul Zimmermann [INRIA project-team CARAMEL].

http://mpc.multiprecision.org/.

MPC is a C library for the arithmetic of complex numbers with arbitrarily high precision and correct rounding of the result. It is built upon and follows the same principles as MPFR.

It is a prerequisite for the GNU compiler collection GCC since version 4.5, where it is used in the C and Fortran frontends for constant folding, the evaluation of constant mathematical expressions during the compilation of a program. Since 2011, it is an official GNU project.

- Version: 0.9 Epilobium montanum
- License: LGPL v2.1+
- ACM: G.1.0 (Multiple precision arithmetic)
- AMS: 30.04 Explicit machine computation and programs
- APP: Dépôt APP le 2003-02-05 sous le numéro IDDN FR 001 060029 000 R P 2003 000 10000
- Programming language: C

5.3. MPFRRCX

Participant: Andreas Enge.

http://mpfrcx.multiprecision.org/
MPFRCX is a library for the arithmetic of univariate polynomials over arbitrary precision real (MPFR) or complex (MPC) numbers, without control on the rounding. For the time being, only the few functions needed to implement the floating point approach to complex multiplication are implemented. On the other hand, these comprise asymptotically fast multiplication routines such as Toom-Cook and the FFT.

- Version: 0.3.1 Banane
- License: LGPL v2.1+
- Programming language: C

5.4. CM

**Participant:** Andreas Enge.

http://cm.multiprecision.org/

The CM software implements the construction of ring class fields of imaginary quadratic number fields and of elliptic curves with complex multiplication via floating point approximations. It consists of libraries that can be called from within a C program and of executable command line applications. For the implemented algorithms, see [9].

- Version: 0.1 Apfelkraut
- License: LGPL v2+
- Programming language: C

5.5. AVIsogenies

**Participants:** Damien Robert [correspondant], Gaëtan Bisson, Romain Cosset [INRIA project-team CARAMEL].

http://avisogenies.gforge.inria.fr/.

AVISOSGENIES (Abelian Varieties and Isogenies) is a MAGMA package for working with abelian varieties, with a particular emphasis on explicit isogeny computation.

Its prominent feature is the computation of $(\ell, \ell)$-isogenies between Jacobian varieties of genus-two hyper-elliptic curves over finite fields of characteristic coprime to $\ell$; practical runs have used values of $\ell$ in the hundreds.

It can also be used to compute endomorphism rings of abelian surfaces, and find complete addition laws on them.

- Version: 0.4
- License: LGPL v2.1+
- Programming language: Magma

5.6. Cubic

**Participant:** Karim Belabas.

http://www.math.u-bordeaux1.fr/~belabas/research/software/cubic-1.2.tgz

CUBIC is a standalone program that prints out generating equations for cubic fields of either signature and bounded discriminant. It depends on the PARI library. The algorithm has quasi-linear time complexity in the size of the output.

- Version: 1.2
- License: GPL v2+
- Programming language: C
SALSA Project-Team

5. Software

5.1. FGb

Participant: J.C. Faugère [contact].

FGb/Gb is a powerful software for computing Gröbner bases; it is written in C/C++ (approximately 250000 lines counting the old Gb software).

5.2. FGb

Participant: Jean-Charles Faugère [correspondant].

FGb is a powerful software for computing Groebner bases. It includes the new generation of algorithms for computing Gröbner bases polynomial systems (mainly the F4,F5 and FGLM algorithms). It is implemented in C/C++ (approximately 250000 lines), standalone servers are available on demand. Since 2006, FGb is dynamically linked with Maple software (version 11 and higher) and is part of the official distribution of this software. See also the web page http://www-salsa.lip6.fr/~jcf/Software/FGb/index.html.

- ACM: I.1.2 Algebraic algorithms
- Programming language: C/C++

5.3. RAGlib

Participant: M. Safey El Din [contact].

RAGLib is a Maple library for computing sampling points in semi-algebraic sets.

5.4. Epsilon

Participant: D. Wang [contact].

Epsilon is a library of functions implemented in Maple and Java for polynomial elimination and decomposition with (geometric) applications.
SECRET Project-Team (section vide)
5. Software

5.1. ECPP

F. Morain has been continuously improving his primality proving algorithm called ECPP, originally developed in the early 1990s. Binaries for version 6.4.5 have been available since 2001 on his web page. Proving the primality of a 512 bit number requires less than a second on an average PC. His personal record is around 25,000 decimal digits, with the fast version he started developing in 2003. All of the code is written in C, and based on publicly available packages (GMP, mpfr, mpc, mpfrcx).

5.2. SEA

Together with E. Schost and L. DeFeo, F. Morain has developed a new implementation of the SEA algorithm that computes the cardinality of elliptic curves over finite fields (large prime case, case \( p = 2 \)). It uses NTL and includes the more recent algorithms for solving all subtasks. The large prime case is relevant to cryptographical needs. The \( p = 2 \) case, though not directly useful, is a good testbed for the FAAST program of LDeFeo (see 5.4 ). This program forms a gforge project.

5.3. TIFA

The TIFA library (short for Tools for Integer FActorization) was initially developed in 2006 and has been continuously improved during the last few years. TIFA is made up of a base library written in C99 using the GMP library, together with stand-alone factorization programs and a basic benchmarking framework to assess the performance of each algorithm.

As of November 2011, the library includes the following algorithms:

- CFRAC (Continued FRACtion factorization [64])
- ECM (Elliptic Curve Method)
- Fermat (McKee’s “fast” variant of Fermat’s algorithm [62])
- SIQS (Self-Initializing Quadratic Sieve [35])
- SQUFOF (SQUare FOrm Factorization [54])

The complete TIFA package has been registered at the French Agency for Software Protection (APP – http://app.legalis.net/) on June, 1st 2011 with the Inter Deposit Digital Number:

IDDN.FR.001.220019.000.S.A.2011.000.31235.

It is now available online at http://www.lix.polytechnique.fr/Labo/Jerome.Milan/tifa/tifa.xhtml and distributed under the Lesser General Public License, version 2.1 or later.

5.4. FAAST

The FAAST library is developed in C++ by L. De Feo and makes use of the NTL library. It implements the algorithms presented in [4], plus other algorithms needed by the author for his research on explicit isogenies.

Version 0.2.0, released on July 11, 2009, is available at http://www.lix.polytechnique.fr/Labo/Luca.De-Feo/FAAST/. The source code is distributed under the General Public License version 2 or higher.

FAAST is a very efficient library for lattices of extensions of finite fields. Our aim is to add support for arbitrary finite fields, making it an essential building block for efficient computer algebra systems.
5.5. Quintix

The Quintix library is a Mathemagix package available at http://www.mathemagix.org/www/main/index.en.html. It is developed in C++ within the Mathemagix computer algebra system. It implements basic arithmetic for Galois rings and their unramified extensions, basic functions for the manipulation of Reed-Solomon codes and the complete Sudan list-decoding algorithm. It also implements the root-finding algorithms presented in [30]. The source code is distributed under the General Public License version 2 or higher.

Quintix is a very efficient library for Galois rings, extensions of Galois rings and root-finding in Galois rings.

5.6. APIP

As part of his activity in the PACE ANR, J. Milan completed, under the supervision of A. Enge, the development of APIP (Another Pairings Implementation in PARI), a PARI/GP module to compute state-of-the-art cryptographic pairings over elliptic curves. This module was intended to be an experimental framework for comparing the performances of the main cryptographic pairings with an emphasis on the standard 128, 192 and 256 bit high security levels.

APIP implements the Tate, Weil, ate and twisted ate pairings together with some optimal variants of the ate and twisted ate pairings for some elliptic curve families. Due to its very flexible architecture, it makes it easy to select several algorithm variants for each step of a pairing computation for a finer analysis.

Due to its emphasis on pairings for cryptographic purposes only, it is doubtful that the APIP module will be integrated in the upstream PARI/GP code base. We hope to be able to distribute APIP as an independent module in the near future, ideally under an open-source licence.
5. Software

5.1. QI: Quadrics Intersection

QI stands for “Quadrics Intersection”. QI is the first exact, robust, efficient and usable implementation of an algorithm for parameterizing the intersection of two arbitrary quadrics, given in implicit form, with integer coefficients. This implementation is based on the parameterization method described in [10], [32], [33], [34] and represents the first complete and robust solution to what is perhaps the most basic problem of solid modeling by implicit curved surfaces.

QI is written in C++ and builds upon the LiDIA computational number theory library [27] bundled with the GMP multi-precision integer arithmetic [26]. QI can routinely compute parameterizations of quadrics having coefficients with up to 50 digits in less than 100 milliseconds on an average PC; see [10] for detailed benchmarks.

Our implementation consists of roughly 18,000 lines of source code. QI has being registered at the Agence pour la Protection des Programmes (APP). It is distributed under the free for non-commercial use INRIA license and will be distributed under the QPL license in the next release. The implementation can also be queried via a web interface [28].

Since its official first release in June 2004, QI has been downloaded six times a month on average and it has been included in the geometric library EXACUS developed at the Max-Planck-Institut für Informatik (Saarbrücken, Germany). QI is also used in a broad range of applications; for instance, it is used in photochemistry for studying the interactions between potential energy surfaces, in computer vision for computing the image of conics seen by a catadioptric camera with a paraboloidal mirror, and in mathematics for computing flows of hypersurfaces of revolution based on constant-volume average curvature.

5.2. Isotop: Topology and Geometry of Planar Algebraic Curves

Isotop is a Maple software for computing the topology of an algebraic plane curve, that is, for computing an arrangement of polylines isotopic to the input curve. This problem is a necessary key step for computing arrangements of algebraic curves and has also applications for curve plotting. This software has been developed since 2007 in collaboration with F. Rouillier from INRIA Paris - Rocquencourt. It is based on the method described in [31] which incorporates several improvements over previous methods. In particular, our approach does not require generic position.

Isotop is registered at the APP (June 15th 2011) with reference IDDN.FR.001.240007.000.S.P.2011.000.10000. This version is competitive with other implementations (such as ALCI.X and INSULATE developed at MPII Saarbrücken, Germany and TOP developed at Santander Univ., Spain). It performs similarly for small-degree curves and performs significantly better for higher degrees, in particular when the curves are not in generic position.

We are currently working on an improved version integrating our new bivariate polynomial solver [22].

5.3. CGAL: Computational Geometry Algorithms Library

Born as a European project, CGAL (http://www.cgal.org) has become the standard library for computational geometry. It offers easy access to efficient and reliable geometric algorithms in the form of a C++ library. CGAL is used in various areas needing geometric computation, such as: computer graphics, scientific visualization, computer aided design and modeling, geographic information systems, molecular biology, medical imaging, robotics and motion planning, mesh generation, numerical methods...
In computational geometry, many problems lead to standard, though difficult, algebraic questions such as computing the real roots of a system of equations, computing the sign of a polynomial at the roots of a system, or determining the dimension of a set of solutions. We want to make state-of-the-art algebraic software more accessible to the computational geometry community, in particular, through the computational geometric library CGAL. On this line, we contributed a model of the *Univariate Algebraic Kernel* concept for algebraic computations \[30\] (see Sections 8.2.2 and 8.4). This CGAL package improves, for instance, the efficiency of the computation of arrangements of polynomial functions in CGAL \[36\]. We are currently developing a model of the *Bivariate Algebraic Kernel* based on our new bivariate polynomial solver \[22\]. This work is done in collaboration with F. Rouillier at INRIA Paris - Rocquencourt and L. Peñaranda at the university of Athens.
5. Software

5.1. Panorama

The ALF team is developing several software prototypes for research purposes: compilers, architectural simulators, programming environments, ...,

Among the many prototypes developed in the project, we describe here ATMI, a microarchitecture temperature model for processor simulation, STiMuL, a temperature model for steady state studies, ATC, an address trace compressor, HAVEGE, an unpredictable random number generator and tiptop, a user-level Linux utility that collects data from hardware performance counters for running tasks, software developed by the team.

5.2. ATMI

Participant: Pierre Michaud.

Contact : Pierre Michaud

Status : Registered with APP Number IDDN.FR.001.250021.000.S.P.2006.000.10600, Available under GNU General Public License

Research on temperature-aware computer architecture requires a chip temperature model. General purpose models based on classical numerical methods like finite differences or finite elements are not appropriate for such research, because they are generally too slow for modeling the time-varying thermal behavior of a processing chip.

We have developed an ad hoc temperature model, ATMI (Analytical model of Temperature in Microprocessors), for studying thermal behaviors over a time scale ranging from microseconds to several minutes. ATMI is based on an explicit solution to the heat equation and on the principle of superposition. ATMI can model any power density map that can be described as a superposition of rectangle sources, which is appropriate for modeling the microarchitectural units of a microprocessor.

Visit http://www.irisa.fr/alf/ATMI or contact Pierre Michaud.

5.3. STiMuL

Participant: Pierre Michaud.

Status: Registered with APP Number IDDN.FR.001.220013.000.S.P.2010.000.31235, Available under GNU General Public License

Some recent research has started investigating the microarchitectural implications of 3D circuits, for which the thermal constraint is stronger than for conventional 2D circuits.

STiMuL can be used to model steady-state temperature in 3D circuits consisting of several layers of different materials. STiMuL is based on a rigorous solution to the Laplace equation [9]. The number and characteristics of layers can be defined by the user. The boundary conditions can also be defined by the user. In particular, STiMuL can be used along with thermal imaging to obtain the power density inside an integrated circuit. This power density could be used for instance in a dynamic simulation oriented temperature modeling such as ATMI.

STiMuL is written in C and uses the FFTW library for discrete Fourier transforms computations.

Visit http://www.irisa.fr/alf/stimul or contact Pierre Michaud.
5.4. ATC

Participant: Pierre Michaud.

Contact: Pierre Michaud

Status: registered with APP number IDDN.FR.001.160031.000.S.P.2009.000.10800, available under GNU LGPL License.

Trace-driven simulation is an important tool in the computer architect's toolbox. However, one drawback of trace-driven simulation is the large amount of storage that may be necessary to store traces. Trace compression techniques are useful for decreasing the storage space requirement. But general-purpose compression techniques are generally not optimal for compressing traces because they do not take advantage of certain characteristics of traces. By specializing the compression method and taking advantages of known trace characteristics, it is possible to obtain a better tradeoff between the compression ratio, the memory consumption and the compression and decompression speed.

ATC is a utility and a C library for compressing/decompressing address traces. It implements a new lossless transformation, Bytesort, that exploits spatial locality in address traces. ATC leverages existing general-purpose compressors such as gzip and bzip2. ATC also provides a lossy compression mode that yields higher compression ratios while preserving certain important characteristics of the original trace.

Visit http://www.irisa.fr/alf/atc or contact Pierre Michaud.

5.5. HAVEGE

Participant: André Seznec.

Contact: André Seznec

Status: Registered with APP Number IDDN.FR.001.500017.001.S.P.2001.000.10000. Available under the LGPL license.

An unpredictable random number generator is a practical approximation of a truly random number generator. Such unpredictable random number generators are needed for cryptography. HAVEGE (HArdware V olatile Entropy Gathering and Expansion) is a user-level software unpredictable random number generator for general-purpose computers that exploits the continuous modifications of the internal volatile hardware states in the processor as a source of uncertainty [16]. HAVEGE combines on-the-fly hardware volatile entropy gathering with pseudo-random number generation.

The internal state of HAVEGE includes thousands of internal volatile hardware states and is merely unmonitorable. HAVEGE can reach an unprecedented throughput for a software unpredictable random number generator: several hundreds of megabits per second on current workstations and PCs.

The throughput of HAVEGE favorably competes with usual pseudo-random number generators such as rand() or random(). While HAVEGE was initially designed for cryptology-like applications, this high throughput makes HAVEGE usable for all application domains demanding high performance and high quality random number generators, e.g., Monte Carlo simulations.

Visit http://www.irisa.fr/alf/HAVEGE or contact André Seznec.

5.6. Tiptop

Participant: Erven Rohou.

Status: Registered with APP (Agence de Protection des Programmes). Available under GNU General Public License v2.
Tiptop is a new simple and flexible user-level tool that collects hardware counter data on Linux platforms (version 2.6.31+). The goal is to make the collection of performance and bottleneck data as simple as possible, including simple installation and usage. In particular, we stress the following points.

- Installation is only a matter of compiling the source code. No patching of the Linux kernel is needed, and no special-purpose module needs to be loaded.
- No privilege is required, any user can run `tiptop` — non-privileged users can only watch processes they own, ability to monitor anybody’s process opens the door to side-channel attacks.
- The usage is similar to `top`. There is no need for the source code of the applications of interest, making it possible to monitor proprietary applications or libraries. And since there is no probe to insert in the application, understanding of the structure and implementation of complex algorithms and code bases is not required.
- Applications do not need to be restarted, and monitoring can start at any time (obviously, only events that occur after the start of `tiptop` are observed).
- Events can be counted per thread, or per process.

Tiptop is written in C. It can take advantage of libncurses when available for pseudo-graphic display. For more information, please contact Erven Rohou.
CAIRN Project-Team

5. Software

5.1. Panorama

Besides the development of new reconfigurable architectures, the need for efficient compilation flow is stronger than ever. Challenges come from the high parallelism of these architectures and also from new constraints such as resource heterogeneity, memory hierarchy and power constraints and management. We aim at defining a highly effective software framework for the compilation of high-level specifications into optimized code executed on a reconfigurable hardware platform. Figure 2 shows the global framework that we are currently developing.

![Figure 2. CAIRN’s general software development framework](image)

Our approach assumes that the application is specified as a hierarchical block diagram of communicating tasks expressing data-flow or control, where each task is expressed using languages like C, Signal, Scilab or Matlab, and is then transformed into an internal representation by the compiler front-end. Then, our framework is based on applying some high-level transformations onto the internal representation.

Different internal representations are used depending on the targeted transformations or the targeted architectures.

- The classical Control and Data Flow Graph (CDFG) is the main internal formalism of our framework. It is the basis for transformations like code optimizations, fixed-point transformations, instruction-set extraction or scheduling. Gateways will be provided from CDFG to other supported formalisms.

- The Hierarchical Conditional Dependency Graph (HCDG) format is also used as the internal representation for pattern-based transformations.
• Other internal representations like Signal Flow Graphs (SFG) and Polyhedral Reduced Dependence Graph (PRDG) will be used respectively for application accuracy estimation and loop parallelization techniques.

Finally, back-end tools enable the generation of code like VHDL for the hardwired or reconfigurable blocks, C for embedded processor software, and SystemC for simulation purposes (e.g. fixed-point simulations). The compiler front-end, the back-end generators, the transformation toolbox as well as the different internal representations and their respective gateways are based on a single framework: the Gecos framework.

Besides CAIRN’s general design workflow, and in order to promote research undertaken by CAIRN, several hardware and software prototypes are developed. Among those, some distributed software are presented in this report: Gecos a flexible compilation platform, ID.Fix an infrastructure for the automatic transformation of software code aiming at the conversion of floating-point data types into a fixed-point representation, UPAK and Durase for the compilation and the synthesis targeting reconfigurable platforms, and Interconnect Explorer a high-level power and delay estimation tool for on-chip interconnects.

5.2. Gecos

Participants: Steven Derrien [correspondent], Daniel Menard, Kevin Martin, Maxime Naulet, Antoine Floch, Antoine Morvan, Clément Guy, Amit Kumar.

The Gecos (Generic Compiler Suite) project is an open source Eclipse-based C compiler infrastructure developed in the CAIRN group since 2004 that allows for fast prototyping of complex compiler passes. Gecos was designed so as to address part of the shortcomings of existing C/C++ infrastructures such as SUIF and LLVM.

Gecos is a 100% Java based implementation and is based on modern software engineering practices. It uses Eclipse plugin as an underlying infrastructure and thus takes benefits of its plugin mechanism to be easily extensible. Gecos follows Model Driven Software Engineering techniques and rely on Eclipse Modeling Framework. The framework is open-source and is hosted on the INRIA gforge at http://gecos.gforge.inria.fr. The Gecos infrastructure is still under very active development, and now serves as a backbone infrastructure to many group members (Upak, Durase, ID.Fix). In 2011, the work has focused on extending the loop analysis transformation framework, which now includes an OpenMP static analysis tool (developed jointly with Colorado State University) that was presented in June at the 7th International Workshop on OpenMP [39]. The software engineering challenges posed by optimizing compiler also happen to be a novel and promising application field for the MDE community, which led to joint publication [45] with members from CSU and the Triskell EPI team at the IEEE/ACM Models conference in October 2011. This cross fertilization between MDE and Compilers is the core topic of Clément Guy’s PhD thesis supervised by members of CAIRN (S. Derrien) and Triskell (J.M. Jezequel and B. Combemale).

5.3. ID.Fix: Infrastructure for the Design of Fixed-point Systems

Participants: Daniel Menard [correspondant], Olivier Senteys, Romuald Rocher, Nicolas Simon, Quentin Meunier.

The different techniques proposed by the team for fixed-point conversion are implemented on the ID.Fix infrastructure. The application is described with a C code using floating-point data types and different pragmas, used to specify parameters (dynamic, input/output word-length, delay operations) for the fixed-point conversion. This tool determines and optimizes the fixed-point specification and then, generates a C code using fixed-point data types (ac_fixed) from Mentor Graphics. The infrastructure is made-up of three main modules corresponding to the fixed-point conversion (Fix.Conv), the accuracy evaluation (Acc.Eval) and the dynamic range evaluation (Dyn.Eval).
The different developments carried out in 2011 allow obtaining a fixed-point conversion tool handling functions, conditional structures and repetitive structures having a fixed number of iterations during time. For the accuracy evaluation (Acc.Eval), conditional structures and correlation between noise sources have been considered. For the dynamic range evaluation (Dyn.Eval), the method based on the Karhunen-Loève Expansion (KLE) have been implemented. It allows determining the dynamic range for a given overflow probability.

The development of this tool has been achieved thanks to an INRIA post-doc in the context of S2S4HLS project until August 2011, and a University of Rennes graduate engineer from November 2011 in the context of DEFIS ANR project and different students during their training period.

5.4. UPaK: Abstract Unified Pattern-Based Synthesis Kernel for Hardware and Software Systems

Participants: Christophe Wolinski [correspondant], François Charot, Antoine Floch.

We are developing (with strong collaboration of Lund University, Sweden and Queensland University, Australia) UPaK Abstract Unified Pattern Based Synthesis Kernel for Hardware and Software Systems [119]. The preliminary experimental results obtained by the UPak system show that the methods employed in the systems enable a high coverage of application graphs with small quantities of patterns. Moreover, high application execution speed-ups are ensured, both for sequential and parallel application execution with processor extensions implementing the selected patterns. UPaK is one of the basis for our research on compilation and synthesis for reconfigurable platforms. It is based on the HCDG representation of the Polychrony software designed at INRIA-Rennes in the project-team Espresso.

5.5. DURASE: Automatic Synthesis of Application-Specific Processor Extensions

Participants: Christophe Wolinski [correspondant], François Charot, Antoine Floch.

We are developing a framework enabling the automatic synthesis of application specific processor extensions. It uses advanced technologies, such as algorithms for graph matching and graph merging together with constraints programming methods. The framework is organized around several modules.

- CoSaP: Constraint Satisfaction Problem. The goal of CoSaP is to decouple the statement of a constraint satisfaction problem from the solver used to solve it. The CoSaP model is an Eclipse plugin described using EMF to take advantage of the automatic code generation and of various EMF tools.
- HCDG: Hierarchical Conditional Dependency Graph. HCDG is an intermediate representation mixing control and data flow in a single acyclic representation. The control flow is represented as hierarchical guards specifying the execution or the definition conditions of nodes. It can be used in the Gecos compilation framework via a specific pass which translates a CDFG representation into an HCDG.
- Patterns: Flexible tools for identification of computational pattern in a graph and graph covering. These tools model the concept of pattern in a graph and provide generic algorithms for the identification of pattern and the covering of a graph. The following sub-problems are addressed: (sub)-graphs isomorphism, patterns generation under constraints, covering of a graph using a library of patterns. Most of the implemented algorithms use constraints programming and rely on the CoSaP module to solve the optimization problem.

5.6. PowWow: Power Optimized Hardware and Software FrameWork for Wireless Motes (AP-L-10-01)

Participants: Olivier Sentieys [correspondant], Olivier Berder, Romain Fontaine, Arnaud Carer, Samuel Mouget, Steven Derrien.
PowWow is an open-source hardware and software platform designed to handle wireless sensor network (WSN) protocols and related applications. Based on an optimized preamble sampling medium access (MAC) protocol, geographical routing and protothread library, PowWow requires a lighter hardware system than Zigbee [90] to be processed (memory usage including application is less than 10kb). Therefore, network lifetime is increased and price per node is significantly decreased.

CAIRN’s hardware platform (see Figure 3) is composed of:

- The motherboard, designed to reduce power consumption of sensor nodes, embeds an MSP430 microcontroller and all needed components to process PowWow protocol except radio chip. JTAG, RS232, and I2C interfaces are available on this board.
- The radio chip daughter board is currently based on a TI CC2420.
- The coprocessing daughter board includes a low-power FPGA which allows for hardware acceleration for some PowWow features and also includes dynamic voltage scaling features to increase power efficiency. The current version of PowWow integrates an Actel IGLOO AGL250 FPGA and a programmable DC-DC converter. We have shown that gains in energy of up to 700 can be obtained by using FPGA acceleration on functions like CRC-32 or error detection with regards to a software implementation on the MSP430.

PowWow distribution also includes a generic software architecture using event-driven programming and organized into protocol layers (PHY, MAC, LINK, NET and APP). The software is based on Contiki [100], and more precisely on the Protothread library which provides a sequential control flow without complex state machines or full multi-threading.

To optimize the network regarding a particular application and to define a global strategy to reduce energy, PowWow offers the following extra tools: over-the-air reprogramming (and soon reconfiguration), analytical power estimation based on software profiling and power measurements, a dedicated network analyzer to probe and fix transmissions errors in the network. More information can be found at http://powwow.gforge.inria.fr.

5.7. SoCLib: Open Platform for Virtual Prototyping of Multi-Processors System on Chip
Participants: François Charot [correspondant], Laurent Perraudeau, Charles Wagner.

SoCLib is an open platform for virtual prototyping of multi-processors system on chip (MP-SoC) developed in the framework of the SoCLib ANR project. The core of the platform is a library of SystemC simulation models for virtual components (IP cores), with a guaranteed path to silicon. All simulation models are written in SystemC, and can be simulated with the standard SystemC simulation environment distributed by the OSCI organization. Two types of models are available for each IP-core: CABA (Cycle Accurate / Bit Accurate), and TLM-DT (Transaction Level Modeling with Distributed Time). All simulation models are distributed as free software. We have developed the simulation model of the NIOSII processor, of the Altera Avalon interconnect, and of the TMS320C62 DSP processor from Texas Instruments. Find more information on its dedicated web page: http://www.soclib.fr.

5.8. OCHRE: On-Chip Randomness Extraction

Participants: Olivier Sentieys [correspondant], Arnaud Carer, Arnaud Tisserand.

Ochre is a set of synthesizable VHDL models for true and pseudo random number generation and hardware accelerated statistical tests. It includes IP cores of different oscillator-based TRNGs, different PRNGs (linear feedback shift registers, cellular automata, AES) and several statistical tests (FIPS 140-2, AIS31, Diehard). This set of IPs has been used to design Ochre V1 and V2 chips and were delivered under license to a company.
5. Software

5.1. PolyLib

PolyLib \(^8\) is a C library of polyhedral functions, that can manipulate unions of rational polyhedra of any dimension, through the following operations: intersection, difference, union, convex hull, simplify, image and preimage. It was the first to provide an implementation of the computation of parametric vertices of a parametric polyhedron, and the computation of an Ehrhart polynomial (expressing the number of integer points contained in a parametric polytope) based on an interpolation method.

It is used by an important community of researchers (in France and the rest of the world) in the area of compilation and optimization using the polyhedral model. Vincent Loechner is the maintainer of this software. It is distributed under GNU General Public License version 3 or later, and it has a Debian package maintained by Serge Guelton (Symbiose Projet, IRISA).

5.2. ZPolyTrans

ZPolyTrans \(^9\) is a C library and a set of executables, that permits to compute the integer transformation of a union of parametric \(\mathbb{Z}\)-polyhedra (the intersection between lattices and parametric polyhedra), as a union of parametric \(\mathbb{Z}\)-polyhedra. The number of integer points of the result can also be computed. It is build upon PolyLib and Barvinok library. This work is based on some theoretical results obtained by Rachid Seghir and Vincent Loechner, that will be published in ACM TACO in 2011 \(^{13}\).

It allows for example to compute the number of solutions of a Presburger formula by eliminating existencial integer variables, or to compute the number of different data accessed by some array accesses contained in an affine parametric loop nest.

The authors of this software are Rachid Seghir (Univ. Batna, Algeria) and Vincent Loechner. It is distributed under GNU General Public License version 3 or later.

5.3. NLR

We have developed a program implementing our loop-nest recognition algorithm, detailed in \(^{7}\). This standalone, filter-like application takes as input a raw trace and builds a sequence of loop nests that, when executed, reproduce the trace. It is also able to predict forthcoming values at an arbitrary distance in the future. Its simple, text-based input format makes it applicable to all kinds of data. These data can take the form of simple numeric values, or have more elaborate structure, and can include symbols. The program is written in standard ANSI C. The code can also be used as a library.

We have used this code to evaluate the compression potential of loop nest recognition on memory address traces, with very good results. We have also shown that the predictive power of our model is competitive with other models on average. The software is available upon request to anybody interested in trying to apply loop nest recognition. It has been distributed to a dozen of colleagues around the world.

We plan on using this software as the base for a new tool we currently design, for the analysis of parallel traces.

5.4. Dynamic version selector

We are developing a toolchain to automatically select between different versions of parallel loop nests, as described in subsection \(^6.2\). It generates the profiling code and selection code from a loop nest source code and different schedules, expressed in the CLooG format.

---

8 http://icps.u-strasbg.fr/PolyLib
9 http://ZPolyTrans.gforge.inria.fr
Benoît Pradelle (PhD) wrote this toolchain, based on python scripts. It is not yet distributed.

5.5. Binary files decompiler

Our research on efficient memory profiling has lead us to develop a sophisticated decompiler. This tool analyzes x86-64 binary programs and libraries, and extracts various structured representations of the code. It works on a routine per routine basis, and first builds a loop hierarchy to characterize the overall structure of the algorithm. It then puts the code into Static Single Assignment (SSA) form to highlight the fine-grain data-flow between registers and memory. Building on these, it performs the following analyzes:

- All memory addresses are expressed as symbolic expressions involving specific versions of register contents, as well as loop counters. Loop counter definitions are recovered by resolving linearly incremented registers and memory cells, i.e., registers that act as induction variables.
- Most conditional branches are also expressed symbolically (with registers, memory contents, and loop counters). This captures the control-flow of the program, but also helps in defining what amounts to loop “trip-counts”, even though our model is slightly more general, because it can represent any kind of iterative structure.

This tool embodies several passes that, as far as we know, do not exist in any existing similar tool. For instance, it is able to track data-flow through stack slots in most cases. It has been specially designed to extract a representation that can be useful in looking for parallel (or parallelizable) loops [48]. It is the basis of several of our studies.

Because binary program decompilation is especially useful to reduce the cost of memory profiling, our current implementation is based on the Pin binary instrumenter. It uses Pin’s API to analyze binary code, and directly interfaces with the upper layers we have developed (e.g., program skeletonization, or minimal profiling). However, we have been careful to clearly decouple the various layers, and to not use any specific mechanism in designing the binary analysis component. Therefore, we believe that it could be ported with minimal effort, by using a binary file format extractor and a suitable binary code parser. It is also designed to abstract away the detailed instruction set, and should be easy to port (even though we have no practical experience in doing so).

We feel that such a tool could be useful to other researchers, because it makes binary code available under abstractions that have been traditionally available for source code only. If sufficient interest emerges, e.g., from the embedded systems community, or from researchers working on WCET, or from teams working on software security, we are willing to distribute and/or to help make it available under other environments.

5.6. Dynamic dependency analyser

We have recently started developing a dynamic dependence analyzer. Such a tool consumes the trace of memory (or object) accesses, and uses the program structure to list all the data dependences appearing during execution. Data dependences in turn are central to the search for parallel sections of code, with the search for parallel loops being only a particular case of the general problem. Most current works of these questions are either specific to a particular analysis (e.g., computing dependence densities to select code portions for thread-level speculation), or restricted to particular forms of parallelism (e.g., typically to fully parallel loops). Our tool tries to generalize existing approaches, and focuses on the program structures to provide helpful feedback either to a user (as some kind of “smart profiler”), or to a compiler (for feedback-directed compilation). For example, the tool is able to produce a dependence schema for a complete loop nest (instead of just a loop). It also targets irregular parallelism, for example analyzing a loop execution to estimate the expected gain of parallelization strategies like inspector-executor.

We have developed this tool in relation to our minimal profiling research project. However, the tool itself has been kept independent of our profiling infrastructure, getting data from it via a well-defined trace format. This intentional design decision has been motivated by our work on distinct execution environments: first
on our usual x86-64 benchmark programs, and second on less regular, more often written in Java, real-world applications. The latter type of applications is likely the one that will most benefit from such tools, because their intrinsic execution environment does not offer enough structure to allow effective static analysis techniques. Parallelization efforts in this context will most likely rely on code annotations, or specific programming language constructs. Programmers will therefore need tools to help them choose between various constructs. Our tool has this ambition. We already have a working tool-chain for C/C++/Fortran programs (or any binary program). We are in the process of developing the necessary infrastructure to connect the dynamic dependence profiler to instrumented Java programs. Other managed execution environments could be targeted as well, e.g., Microsoft’s .Net architecture, but we have no time and/or workforce to devote to such time-consuming engineering efforts.

5.7. VMAD software and LLVM

For dynamic analysis and optimization of programs, we are developing a virtual machine called VMAD, and specific passes to the LLVM compiler suite, plus a modified Clang frontend. It is fully described in subsection 6.1.

We implemented for now a memory access predictor in loop nests, based on the computation of linear interpolation functions. The profiling is very fast compared to other existing tools, as it samples only the first few iterations of each loop in the nest, then it is deactivated to return to the original, faster version. Other tools like PIN or PEBIL do not support such activation/deactivation mechanism.

New annotations for the final user, taken as input by LLVM, and new VMAD modules will be developed, as these tools have been designed to be very evolving.

Alexandra Jimborean (PhD), Matthieu Herrmann (Master student) and Luis Mastrangelo (Master student) are the main contributors of this software. It is not yet distributed.

5.8. Polyhedral prover

Participants: Nicolas Magaud, Julien Narboux, Éric Violard [correspondant].

We are currently developing a formal proof of program transformations based on the polyhedral model. We use the CompCert verified compiler [54] as a framework. This tool is written in the specification language of Coq. It is connected to the activity described in section 6.6.
5. Software

5.1. Introduction

This section lists and briefly describes the software developments conducted within Compsys. Most are tools that we extend and maintain over the years. They now concern two activities only: a) the development of tools linked to polyhedra and loop/array transformations, b) the development of algorithms within the back-end compiler of STMicroelectronics.

Many tools based on the polyhedral representation of codes with nested loops are now available. They have been developed and maintained over the years by different teams, after the introduction of Paul Feautrier’s Pip, a tool for parametric integer linear programming. This “polytope model” view of codes is now widely accepted: it used by Inria projects-teams Cairn and Alchemy/Parkas, PIPS at École des Mines de Paris, Surf from Stanford University, Compaan at Berkeley and Leiden, PiCo from the HP Labs (continued as PicoExpress by Synfora and now Synopsis), the DTSE methodology at Imec, Sadayappan’s group at Ohio State University, Rajopadhye’s group at Colorado State’s University, etc. More recently, several compiler groups have shown their interest in polyhedral methods, e.g., the GCC group, IBM, and Reservoir Labs, a company that develops a compiler fully-based on the polytope model and on the techniques that we (the french community) introduced for loop and array transformations. Polyhedra are also used in test and certification projects (Verimag, Lande, Vertecs). Now that these techniques are well-established and disseminated in and by other groups, we prefer to focus on the development of new techniques and tools, which are described here.

The other activity concerns the developments within the compiler of STMicroelectronics. These are not stand-alone tools, which could be used externally, but algorithms and data structures implemented inside the LAO back-end compiler, year after year, with the help of STMicroelectronics colleagues. As these are also important developments, it is worth mentioning them in this section. They are also completed by important efforts for integration and evaluation within the complete STMicroelectronics toolchain. They concern exact methods (ILP-based), algorithms for aggressive optimizations, techniques for just-in-time compilation, and for improving the design of the compiler.

5.2. Pip

Participants: Cédric Bastoul [MCF, IUT d’Orsay], Paul Feautrier.

Paul Feautrier is the main developer of Pip (Parametric Integer Programming) since its inception in 1988. Basically, Pip is an “all integer” implementation of the Simplex, augmented for solving integer programming problems (the Gomory cuts method), which also accepts parameters in the non-homogeneous term. Pip is freely available under the GPL at http://www.piplib.org . Pip is widely used in the automatic parallelization community for testing dependences, scheduling, several kind of optimizations, code generation, and others. Beside being used in several parallelizing compilers, Pip has found applications in some unconnected domains, as for instance in the search for optimal polynomial approximations of elementary functions (see the Inria project Arénaire).

5.3. Syntol

Participants: Hadda Cherroun [Former PhD student in Compsys], Paul Feautrier.
Syntol is a modular process network scheduler. The source language is C augmented with specific constructs for representing communicating regular process (CRP) systems. The present version features a syntax analyzer, a semantic analyzer to identify DO loops in C code, a dependence computer, a modular scheduler, and interfaces for CLooG (loop generator developed by C. Bastoul) and Cl@k (see Sections 5.4 and 5.6). The dependence computer now handles casts, records (structures), and the modulo operator in subscripts and conditional expressions. The latest developments are, firstly, a new code generator, and secondly, several experimental tools for the construction of bounded parallelism programs.

- The new code generator, based on the ideas of Boulet and Feautrier [31], generates a counter automaton that can be presented as a C program, as a rudimentary VHDL program at the RTL level, as an automaton in the Aspic input format, or as a drawing specification for the DOT tool.

- Hardware synthesis can only be applied to bounded parallelism programs. Our present aim is to construct threads with the objective of minimizing communications and simplifying synchronization. The distribution of operations among threads is specified using a placement function, which is found using techniques of linear algebra and combinatorial optimization.

5.4. Cl@k

Participants: Christophe Alias, Fabrice Baray [Mentor, Former post-doc in Compsys], Alain Darte.

Cl@k (Critical LAttice Kernel) is a stand-alone optimization tool useful for the automatic derivation of array mappings that enable memory reuse, based on the notions of admissible lattice and of modular allocation (linear mapping plus modulo operations). It has been developed in 2005-2006 by Fabrice Baray, former post-doc Inria under Alain Darte’s supervision. It computes or approximates the critical lattice for a given 0-symmetric polytope. (An admissible lattice is a lattice whose intersection with the polytope is reduced to 0; a critical lattice is an admissible lattice with minimal determinant.)

Its application to array contraction has been implemented by Christophe Alias in a tool called Bee (see Section 5.6). Bee uses Rose as a parser, analyzes the lifetimes of the elements of the arrays to be compressed, and builds the necessary input for Cl@k, i.e., the 0-symmetric polytope of conflicting differences. Then, Bee computes the array contraction mapping from the lattice provided by Cl@k and generates the final program with contracted arrays. See previous reports for more details on the underlying theory. Cl@k can be viewed as a complement to the Polylib suite, enabling yet another kind of optimizations on polyhedra. Initially, Bee was the complement of Cl@k in terms of its application to memory reuse. Now, Bee is a stand-alone tool that contains more and more features for program analysis and loop transformations.

5.5. PoCo

Participant: Christophe Alias.

PoCo is a polyhedral compilation framework providing many features to quickly prototype program analysis and optimizations in the polyhedral model. Essentially, PoCo provides:

- C front-end extracting the polyhedral representation of the input program. The parser itself is based on EDG (via ROSE), an industrial C/C++ parser from Edison group used in Intel compilers.

- Extended language of pragmas to feed the source code with compilation directives (a schedule, for example).

- Symbolic layer on polyhedral libraries POLYLIB (set operations on polyhedra) and PIPLIB (parameterized ILP). This feature simplifies drastically the developer task.

- Dependence analysis (polyhedral dependence graph, array dataflow analysis), array region analysis, array liveness analysis.

- C and VHDL code generation based on the ideas of P. Boulet and P. Feautrier [31].
The array dataflow analysis (ADA) of PoCo has been extended to a FADA (Fuzzy ADA) by M. Belaoucha, former PhD student at Université de Versailles. FADALib is available at http://www.prism.uvsq.fr/~bem/fadalib/.

PoCo has been developed by Christophe Alias. It represents more than 19000 lines of C++ code. The tools Bee, Chuba, and RanK presented thereafter make an extensive use of PoCo abstractions.

5.6. Bee

Participants: Christophe Alias, Alain Darte.

Bee is a source-to-source optimizer that contracts the temporary arrays of a program under scheduling constraints. Bee bridges the gap between the mathematical optimization framework described in [32] and implemented in Cl@k (Section 5.4), and effective source-to-source array contraction. Bee applies a precise lifetime analysis for arrays to build the mathematical input of Cl@k. Then, Bee derives the array allocations from the basis found by Cl@k and generates the C code accordingly. Bee is – to our knowledge – the only complete array contraction tool.

Bee is sensitive to the program schedule. This latter feature enlarges the application field of array contraction to parallel programs. For instance, it is possible to mark a loop to be software-pipelined (with an affine schedule) and to let Bee find an optimized array contraction. But the most important application is the ability to optimize communicating regular processes (CRP). Given a schedule for every process, Bee can compute an optimized size for the channels, together with their access functions (the corresponding allocations). We currently use this feature in source-to-source transformations for high-level synthesis (see Section 3.3).

- Bee was made available to STMICROELECTRONICS as a binary.
- Bee will be transferred to the (incubated) start-up Zettice, initiated by Alexandru Plesco.
- Bee is used as an external tool by the compiler GECOS developed in the Cairn team at IRISA.

Bee has been implemented by Christophe Alias, using the compiler infrastructure PoCo. It represents more than 2400 lines of C++ code.

5.7. Chuba

Participants: Christophe Alias, Alain Darte, Alexandru Plesco.

Chuba is a source-level optimizer that improves a C program in the context of the high-level synthesis (HLS) of hardware. Chuba is an implementation of the work described in the PhD thesis of Alexandru Plesco. The optimized program specifies a system of multiple communicating accelerators, which optimize the data transfers with the external DDR memory. The program is divided into blocks of computations obtained thanks to tiling techniques, and, in each block, data are fetched by block to reduce the penalty due to line changes in the DDR accesses. Four accelerators achieve data transfers in a macro-pipeline fashion so that data transfers and computations (performed by a fifth accelerator) are overlapped.

So far, the back-end of Chuba is specific to the HLS tool C2H but the analysis is quite general and adapting Chuba to other HLS tools should be possible. Besides, it is interesting to mention that the program analysis and optimizations implemented in Chuba address a problem that is also very relevant in the context of GPGPUs.

Chuba has been implemented by Christophe Alias, using the compiler infrastructure PoCo. It represents more than 900 lines of C++. The reduced size of Chuba is mainly due to the high-level abstractions provided by PoCo.

5.8. C2fsm

Participant: Paul Feautrier.
C2fsm is a general tool that converts an arbitrary C program into a counter automaton. This tool reuses the parser and pre-processor of Syntol, which has been greatly extended to handle while and do while loops, goto, break, and continue statements. C2fsm reuses also part of the code generator of Syntol and has several output formats, including FAST (the input format of Aspic), a rudimentary VHDL generator, and a DOT generator which draws the output automaton. C2fsm is also able to do elementary transformations on the automaton, such as eliminating useless states, transitions and variables, simplifying guards, or selecting cut-points, i.e., program points on loops that can be used by RanK to prove program termination.

5.9. RanK

Participants: Christophe Alias, Alain Darte, Paul Feautrier, Laure Gonnord.

RanK is a software tool that can prove the termination of a program (in some cases) by computing a ranking function, i.e., a mapping from the operations of the program to a well-founded set that decreases as the computation advances. In case of success, RanK can also provide an upper bound of the worst-case time complexity of the program as a symbolic affine expression involving the input variables of the program (parameters), when it exists. In case of failure, RanK tries to prove the non-termination of the program and then to exhibit a counter-example input. This last feature is of great help for program understanding and debugging, and has already been experimented.

The input of RanK is an integer automaton, computed by C2fsm (see Section 5.8), representing the control structure of the program to check. RanK uses the Aspic tool, developed by Laure Gonnord during her PhD thesis, to compute automaton invariants. RanK has been used to discover successfully the worst-case time complexity of many benchmarks programs of the community. It uses the libraries Piplib and Polylib.

RanK has been implemented by Christophe Alias, using the compiler infrastructure PoCo. It represents more than 3000 lines of C++.

5.10. Simplifiers

Participant: Paul Feautrier.

The aim of the simple library is to simplify boolean formulas on affine inequalities. It works by detecting redundant inequalities in the representation of the subject formula as an ordered binary decision diagram. It uses PIP for testing the feasibility – or unfeasibility – of a conjunction of affine inequality.

The library is written in Java and is presented as a collection of class files. For experimentation, several front-ends have been written They differ mainly in their input syntax, among which are a C like syntax, the Mathematica and SMTLib syntaxes, and an ad hoc Quast syntax.

5.11. LAO Developments in Aggressive Compilation

Participants: Benoît Boissinot, Florent Bouchez, Florian Brandner, Quentin Colombet, Alain Darte, Benoît Dupont de Dinechin [Kalray], Christophe Guillon [STMicroelectronics], Sebastian Hack [Former post-doc in Compsys], Fabrice Rastello, Cédric Vincent [Former student in Compsys].

Our aggressive optimization techniques are all implemented in stand-alone experimental tools (as for example for register coalescing algorithms) or within LAO, the back-end compiler of STMicroelectronics, or both. They concern SSA construction and destruction, instruction-cache optimizations, register allocation. Here, we report only our more recent activities, which concern register allocation.
Our developments on register allocation within the STMicroelectronics compiler started when Cédric Vincent (bachelor degree, under Alain Darte supervision) developed a complete register allocator in LAO, the assembly-code optimizer of STMicroelectronics. This was the first time a complete implementation was done with success, outside the MCDT (now CEC) team, in their optimizer. Since then, new developments are constantly done, in particular by Florent Bouchez, advised by Alain Darte and Fabrice Rastello, as part of his master internship and PhD thesis. In 2009, Quentin Colombet started to develop and integrate into the main trunk of LAO a full implementation of a two-phases register allocation. This implementation now includes two different decoupled spilling phases, the first one as described in Sebastian Hack’s PhD thesis and a new ILP-based solution (see Section 6.2). It also includes an up-to-date graph-based register coalescing. Finally, since all these optimizations take place under SSA form, it includes also a mechanism for going out of colored-SSA (register-allocated SSA) form that can handle critical edges and does further optimizations (see for example Section 6.3).

5.12. LAO Developments in JIT Compilation

Participants: Benoît Boissinot, Florian Brandner, Alain Darte, Benoît Dupont de Dinechin [Kalray], Christophe Guillon [STMicroelectronics], Fabrice Rastello.

The other side of our work in the STMicroelectronics compiler LAO has been to adapt the compiler to make it more suitable for JIT compilation. This means lowering the time and space complexity of several algorithms. In particular we implemented our fast out-of-SSA translation method, and we programmed and tested various ways to compute the liveness information as described in Section 6.6. Recent efforts (see Section 6.4) also focused on developing a tree-scan register allocator for the JIT part of the compiler, in particular a JIT conservative coalescing. The technique is to bias the tree-scan coalescing, taking into account register constraints, with the result of a JIT aggressive coalescing.

5.13. Low-Level Exchange Format (TireX) and Minimalist Intermediate Representation (MinIR)

Participants: Christophe Guillon [STMicroelectronics], Fabrice Rastello, Benoît Dupont de Dinechin [Kalray].

Most compilers define their own intermediate representation (IR) to be able to work on a program. Sometimes, they even use a different representation for each representation level, from source code parsing to the final object code generation. MinIR (Minimalist Intermediate Representation) is a new intermediate representation, designed to ease the interconnection of compilers, static analyzers, code generators, and other tools. In addition to the specification of MinIR, generic core tools have been developed to offer a basic toolkit and to help the connection of client tools. MinIR generators exist for several compilers, and different analyzers are developed as a testbed to rapidly prototype different static analyses over SSA code. This new common format enables the comparison of the code generator of several production compilers, and simplifies the connection of external tools to existing compilers.

MinIR has been extended into TireX, a Textual Intermediate Representation for EXchanging target-level information between compiler optimizers and whole or parts of code generators (aka compiler back-end). The first motivation for this intermediate representation is to factor target-specific compiler optimizations into a single component, in case several compilers need to be maintained for a particular target (e.g., operating system compiler and application code compiler). Another motivation is to reduce the run-time cost of JIT compilation and of mixed mode execution, since the program to compile is already in a representation lowered to the level of the target processor. Besides the lowering at the target level, the extensions of MinIR include the program data stream and loop scoped information. TireX is currently produced by the Open64/Path64 and the LLVM compilers, with a GCC producer under work. It is consumed by the LAO code generator.

Detailed information, generic core tools, and LLVM IR based generator for MinIR are available at [http://www.assembla.com/spaces/minir-dev/wiki](http://www.assembla.com/spaces/minir-dev/wiki). Open64/Path64 emitter for TireX and its LAO back-end are available at [https://compilation.ens-lyon.fr/](https://compilation.ens-lyon.fr/). MinIR was presented at WIR’11 [17].
5. Software

5.1. TimeSquare

Participants: Charles André, Nicolas Chleq, Julien Deantoni, Benoît Ferrero, Frédéric Mallet [correspondant].

TimeSquare is a software environment for the modeling and analysis of timing constraints in embedded systems. It relies specifically on the Time Model of the MARTE UML profile (see section 3.2), and more accurately on the associated Clock Constraint Specification Language (CCSL) for the expression of timing constraints.

TimeSquare offers four main functionalities:

1. graphical and/or textual interactive specification of logical clocks and relative constraints between them;
2. definition and handling of user-defined clock constraint libraries;
3. automated simulation of concurrent behavior traces respecting such constraints, using a Boolean solver for consistent trace extraction;
4. call-back mechanisms for the traceability of results (animation of models, display and interaction with waveform representations, generation of sequence diagrams...).

In practice TimeSquare is a plug-in developed with Eclipse modeling tools. The software is registered by the Agence pour la Protection des Programmes, under number IDDN.FR.001.170007.000.S.P.2009.001.10600. It can be downloaded from the site http://timesquare.inria.fr/. It has been integrated in the OpenEmbeDD ANR RNTL platform, and other such actions are under way.

5.2. K-Passa

Participants: Anthony Coadou, Jean-Vivien Millo [correspondant], Robert de Simone.

This software is dedicated to the simulation, analysis, and static scheduling scheduling of Event/Marked Graphs, SDF and KRG extensions. A graphical interface allows to edit the Process Networks and their time annotations (latency, ...). Symbolic simulation and graph-theoretic analysis methods allow to compute and optimize static schedules, with best throughputs and minimal buffer sizes. In the case of KRG the (ultimately k-periodic) routing patterns can also be provided and transformed for optimal combination of switching and scheduling when channels are shared. KPASSA also allows for import/export of specific description formats such as UML-MARTE, to and from our other TimeSquare tool.

The tool was originally developed mainly as support for experimentations following our research results on the topic of Latency-Insensitive Design. This research was conducted and funded in part in the context of the CIM PACA initiative, with initial support from ST Microelectronics and Texas Instruments.

KPASSA is registered by the Agence pour la Protection des Programmes, under the number IDDN.FR.001.310003.000.S.P.2009.000.20700. it can be downloaded from the site http://www-sop.inria.fr/aoste/index.php?page=software/kpassa.

5.3. SynDEx

Participants: Maxence Guesdon, Yves Sorel [correspondant], Cécile Stentzel, Meriem Zidouni.

SynDEx is a system level CAD software implementing the AAA methodology for rapid prototyping and for optimizing distributed real-time embedded applications. Developed in OCaML it can be downloaded free of charge, under INRIA copyright, from the general SynDEx site http://www.syndex.org.
The AAA methodology is described in section 3.3. Accordingly, SYNDEX explores the space of possible allocations (spatial distribution and temporal scheduling), from application elements to architecture resources and services, in order to match real-time requirements; it does so by using schedulability analyses and heuristic techniques. Ultimately it generates automatically distributed real-time code running on real embedded platforms. The last major release of SYNDEX (V7) allows the specification of multi-periodic applications.

Application algorithms can be edited graphically as directed acyclic task graphs (DAG) where each edge represent a data dependence between tasks, or they may be obtained by translations from several formalisms such as Scicos (http://www.scicos.org), Signal/Polychrony (http://www.irisa.fr/espresso/Polychrony), or UML2/MARTE models (http://www.omg.org/technology/documents/profile_catalog.htm).

Architectures are represented as graphical block diagrams composed of programmable (processors) and non-programmable (ASIC, FPGA) computing components, interconnected by communication media (shared memories, links and busses for message passing). In order to deal with heterogeneous architectures it may feature several components of the same kind but with different characteristics.

Two types of non-functional properties can be specified for each task of the algorithm graph. First, a period that does not depend on the hardware architecture. Second, real-time features that depend on the different types of hardware components, ranging amongst execution and data transfer time, memory, etc. Requirements are generally constraints on deadline equal to period, latency between any pair of tasks in the algorithm graph, dependence between tasks, etc.

Exploration of alternative allocations of the algorithm onto the architecture may be performed manually and/or automatically. The latter is achieved by performing real-time multiprocessor schedulability analyses and optimization heuristics based on the minimization of temporal or resource criteria. For example while satisfying deadlines and latencies constraints they can minimize the total execution time (makespan) of the application onto the given architecture, as well as the amount of memory. The results of each exploration is visualized as timing diagrams simulating the distributed real-time implementation.

Finally, real-time distributed embedded code can be automatically generated for dedicated distributed real-time executives, possibly calling services of resident real-time operating systems such as Linux/RTAI or Osek for instance. These executives are deadlock-free, based on off-line scheduling policies. Dedicated executives induce minimal overhead, and are built from processor-dependent executive kernels. To this date, executives kernels are provided for: TMS320C40, PIC18F2680, i80386, MC68332, MPC555, i80C196 and Unix/Linux workstations. Executive kernels for other processors can be achieved at reasonable cost following these examples as patterns.

5.4. SAS

Participants: Daniel de Rauglaudre [correspondant], Yves Sorel.

The SAS (Simulation and Analysis of Scheduling) software allows the user to perform the schedulability analysis of periodic task systems in the monoprocessor case.

The main contribution of SAS, when compared to other commercial and academic softwares of the same kind, is that it takes into account the exact preemption cost between tasks during the schedulability analysis. Beside usual real-time constraints (precedence, strict periodicity, latency, etc.) and fixed-priority scheduling policies (Rate Monotonic, Deadline Monotonic, Audsley++, User priorities), SAS additionally allows to select dynamic scheduling policy algorithms such as Earliest Deadline First (EDF). The resulting schedule is displayed as a typical Gantt chart with a transient and a permanent phase, or as a disk shape called "dameid", which clearly highlights the idle slots of the processor in the permanent phase.

For a schedulable task system under EDF, when the exact preemption cost is considered, the period of the permanent phase may be much longer than the least common multiple (LCM) of the periods of all tasks, as often found in traditional scheduling theory. Specific effort has been made to improve display in this case. The classical utilization factor, the permanent exact utilization factor, the preemption cost in the permanent phase, and the worst response time for each task are all displayed when the system is schedulable. Response times of each task relative time can also be displayed (separately).
SAS is written in OCaml, using CAMLP5 (syntactic preprocessor) and OLIBRT (a graphic toolkit under X). Both are written by Daniel de Rauglaudre.
4. Software

4.1. Gaspard 2

Participants: Jean-Luc Dekeyser [correspondant], All DaRT team.

Gaspard2 is an Integrated Development Environment (IDE) for SoC visual co-modeling. It allows or will allow modeling, simulation, testing and code generation of SoC applications and hardware architectures. Its purpose is to provide a single environment for all the SoC development processes:

- High level modeling of applications and hardware architectures
- Application and hardware architecture association (mapping and scheduling)
- Application refactoring
- Deployment specification
- Model to model transformation (to automatically produce models for several target platforms)
- Code generation
- Simulation
- Reification of any stages of the development

The Gaspard2 tool is based on the Eclipse [62] IDE. A set of plugins provides the different functionalities. Gaspard2 provides an internal engine to execute transformation chains. This engine is able to run either QVT (OMG standard) or Java transformations. It is also able to run model-to-text transformations based on Acceleo [64]. The Gaspard2 engine is defined to execute models conform to an internal transformation chains meta-model. A GUI has been developed to specify transformation chain models by drawing them. For the final user, application, hardware architecture, association, deployment and technology models are specified and manipulated by the developer through UML diagrams, and saved by the UML tool in an XMI file format. Gaspard2 manipulates these models through repositories (Java interfaces and implementations) automatically generated thanks to the Ecore specification. Several transformation chains are provided with Gaspard2 to target, from UML models, several execution or simulation platforms (OpenMP, OpenCL, Pthread, SystemC, VHDL, ...). This input language is based on the MARTE UML profile. A tool to generate SIMD configurations derived from the mppSoC model was developed. It allows to automatically generate the VHDL code from a high specification modeled at a high abstraction level (UML model using MARTE profile) based on the IP mppSoC library. The developed tool facilitates to the user to choose a SIMD configuration adapted to his application needs. It has been integrated in the Gaspard environment. Gaspard2 as an educational resource. The Gaspard2 platform was one of the topics taught in the context of the courses on embedded systems in Telecom Lille and in a Master 2 (TNSI) lecture "Design tools for embedded systems" at the University of Valenciennes. These lectures focused on the potentiality to generate several targets from a subset of the Marte profile and the ability to target system on chip architectures at the TLM level respectively. Furthermore, the model driven engineering characteristics of Gaspard2 are largely detailed in the lecture of Software engineering at Polytech Lille and in the Master of research at university of Lille too.

- See also the web page http://www.gaspard2.org/
- Inria software evaluation: A-2, SO-4, SM-2, EM-1, SDL-2, DA-4, CD-4, MS-4, TPM4
- Version: 2.1.0

4.2. Papyrus

Participants: Cédric Dumoulin [correspondant], Amine El Kouhen, Rahma Yangui.
The Papyrus tool is an UML Development Environment fully compliant with the UML standard and providing all UML diagrams. It is now an Eclipse project (in the incubator state). Papyrus Eclipse can easily be installed in Eclipse from the Eclipse update site. The Papyrus Tool is developed under an Open source license in collaboration with CEA, Atos, Airbus, LIFL.

- See also the web page http://www.eclipse.org/papyrus/
- Software data: plugins number > 150, lines number > 1 million
- Inria software evaluation: A-5, SO-4, SM-4, EM-4, SDL-5, DA-4, CD-4, MS-4, TPM3
- Version: 0.9.0

4.3. Model Driven Factory

Participants: Alexis Muller, Anne Etien [correspondant], Thomas Legrand.

MDFactory is a Model Driven Engineering environment to design, develop and run software production chains. This tool supports our approach based on localized transformation and our Extend operator [96]. It provides a graphical editor to build such production chains with drag and drop from a reusable transformation library. MDFactory is based on the Eclipse platform and the Eclipse Modeling Framework (EMF). It is used to build Gaspard2 integrated transformation chains. This software will be transfered to the start up company Axellience.

- Software data: plugins number around 75
- Evaluation of the software: A 4; SO 4; SM 2; EM 3; SDL 3; DA 4; CD 3; MS 2; TPM 2
- Version: 1.0

4.4. OMEGSI

Participant: Amen Souissi [correspondant].

OMEGSI is an integrated development environment (IDE) for collaborative portals. It allows business process-centered modeling, process simulation, process optimization and full code generation for collaborative portals. The OMEGSI tool is based on the Eclipse IDE. A set of plugins provides the different functionalities. OMEGSI provides an internal engine to execute interactives transformation strategies. This engine (TranS) is written in QVT transformation and able to run any transformation type (QVT, JAV, Acceleo...). Currently one transformation strategy is provided with OMEGSI to target, from an UML model, the Dolmen execution platform. This input language is based on the MACoP (Modeling and Analysis of Collaborative Portal) UML profile. The fully functional OMEGSI Beginning version still available on Ecreall website.

- See also the web page http://omegsi.ecreall.com/
- Inria software evaluation: A-3, SO-3, SM-1, EM-2, SDL-4, DA-4, CD-4, MS-4, TPM4
- Version:
ESPRESSO Project-Team

4. Software

4.1. The Polychrony toolset

Participants: Loïc Besnard, Thierry Gautier, Paul Le Guernic.

The Polychrony toolset is an Open Source development environment for critical/embedded systems. It is based on Signal, a real-time polychronous data-flow language. It provides a unified model-driven environment to perform design exploration by using top-down and bottom-up design methodologies formally supported by design model transformations from specification to implementation and from synchrony to asynchrony. It can be included in heterogeneous design systems with various input formalisms and output languages.

The Polychrony toolset provides a formal framework:

- to validate a design at different levels, by the way of formal verification and/or simulation,
- to refine descriptions in a top-down approach,
- to abstract properties needed for black-box composition,
- to assemble heterogeneous predefined components (bottom-up with COTS),
- to generate executable code for various architectures.

The Polychrony toolset contains three main components and an experimental interface to GNU Compiler Collection (GCC):

- The Signal toolbox, a batch compiler for the Signal language, and a structured API that provides a set of program transformations. The Signal toolbox can be installed without the other components. The Signal toolbox is distributed under GPL V2 license.
- The Signal GUI, a Graphical User Interface to the Signal toolbox (editor + interactive access to compiling functionalities). The Signal GUI is distributed under GPL V2 license.
- The SME platform, a front-end to the Signal toolbox in the Eclipse environment. The SME platform is distributed under EPL license.
- GCCst, a back-end to GCC that generates Signal programs (not yet available for download).

The Polychrony toolset also provides:

- libraries of Signal programs,
- a set of Signal program examples,
- user oriented and implementation documentations,
- facilities to generate new versions.

The Polychrony toolset can be freely downloaded on the following web sites:

- The Polychrony toolset public web site: http://www.irisa.fr/espresso/Polychrony. This site, intended for users and for developers, contains downloadable executable and source versions of the software for different platforms, user documentation, examples, libraries, scientific publications and implementation documentation. In particular, this is the site for the new open-source distribution of Polychrony.
- The INRIAGForge: https://gforge.inria.fr. This site, intended for internal developers, contains the whole sources of the environment and their documentation.
- The TOPCASED distribution site: http://www.topcased.org. This site provides the current reference version of the SME platform, including the executable of the Signal toolbox.
4.2. The Eclipse interface

**Participants:** Loïc Besnard, Yann Glouche, Huafeng Yu, François Fabre, Yue Ma.

We have developed a meta-model and interactive editor of Polychrony in Eclipse. Signal-Meta is the metamodel of the Signal language implemented with Eclipse/eCore. It describes all syntactic elements specified in [35]: all Signal operators (e.g. arithmetic, clock synchronization), model (e.g. process frame, module), and construction (e.g. iteration, type declaration).

The meta-model primarily aims at making the language and services of the Polychrony environment available to inter-operation and composition with other components (e.g. AADL, Simulink, GeneAuto) within an Eclipse-based development tool-chain. Polychrony now comprises the capability to directly import and export eCore models instead of textual Signal programs, in order to facilitate interaction between components within such a tool-chain.

It also provides a graphical modeling framework allowing to design applications using a component-based approach. Application architectures can be easily described by just selecting components via drag and drop, creating some connections between them and specifying their parameters as component attributes. Using the modeling facilities provided with the Topcased framework, we have created a graphical environment for Polychrony (see figure 7) called SME (Signal-Meta under Eclipse). To highlight the different parts of the modeling in Signal, we split the modeling of a Signal process in three diagrams: one to model the interface of the process, one to model the computation (or dataflow) part, and one to model all explicit clock relations and dependences. The SME environment is available through the Espresso update site [27], in the current OpenEmbeDD distribution [26], or in the TopCased distribution [28].

4.3. Integrated Modular Avionics design using Polychrony

**Participants:** Thierry Gautier, Paul Le Guernic, Jean-Pierre Talpin.
The Apex interface, defined in the ARINC standard [29], provides an avionics application software with the set of basic services to access the operating-system and other system-specific resources. Its definition relies on the Integrated Modular Avionics approach (IMA [30]). A main feature in an IMA architecture is that several avionics applications (possibly with different critical levels) can be hosted on a single, shared computer system. Of course, a critical issue is to ensure safe allocation of shared computer resources in order to prevent fault propagations from one hosted application to another. This is addressed through a functional partitioning of the applications with respect to available time and memory resources. The allocation unit that results from this decomposition is the partition.

A partition is composed of processes which represent the executive units (an ARINC partition/process is akin to a Unix process/task). When a partition is activated, its owned processes run concurrently to perform the functions associated with the partition. The process scheduling policy is priority preemptive. Each partition is allocated to a processor for a fixed time window within a major time frame maintained by the operating system. Suitable mechanisms and devices are provided for communication and synchronization between processes (e.g. buffer, event, semaphore) and partitions (e.g. ports and channels). The specification of the ARINC 651-653 services in Signal [7] is now part of the Polychrony distribution and offers a complete implementation of the Apex communication, synchronization, process management and partitioning services. Its Signal implementation consists of a library of generic, parameterizable Signal modules.

4.4. Multi-clocked mode automata

Participants: Jean-Pierre Talpin, Thierry Gautier, Christian Brunette.

Gathering advantages of declarative and imperative approaches, mode automata were originally proposed by Maraninchi et al. to extend the functionality-oriented data-flow paradigm with the capability to model transition systems easily and provide an additional imperative flavor. Similar variants and extensions of the same approach to mix multiple programming paradigms or heterogeneous models of computation [36] have been proposed until recently, the latest advance being the combination of stream functions with automata in [38]. Nowadays, commercial toolsets such as the Esterel Studio’s Scade or Matlab/Simulink’s Stateflow are largely inspired from similar concepts.

While the introduction of preemption mechanism in the multi-clocked data-flow formalism Signal was previously studied by Rutten et al. in [51], no attempt has been made to extend mode automata with the capability to model multi-clocked systems and multi-rate systems. In [53], we extend Signal-Meta with an inherited meta-model of multi-clocked mode automata. A salient feature is the simplicity incurred by the separation of concerns between data-flow (that expresses structure) and control-flow (that expresses a timing model) that is characteristic to the design methodology of Signal.

While the specification of mode automata in related works requires a primary address on the semantics and on compilation of control, the use of Signal as a foundation allows to waive this specific issue to its analysis and code generation engine Polychrony and clearly exposes the semantics and transformation of mode automata in a much simpler way by making use of clearly separated concerns expressed by guarded commands (data-flow relations) and by clock equations (control-flow relations).

4.5. Hyper-text source documentation of Polychrony

Participants: Loïc Besnard, Thierry Gautier, Paul Le Guernic.

As part of its open-source release, the Polychrony toolset not only comprises source code libraries but also an important corpus of structured documentation, whose aim is not only to document each functionality and service, but also to help a potential developer to package a subset of these functionalities and services and adapt them to developing a new application-specific tool: a new language front-end, a new back-end compiler. This multi-scale, multi-purpose documentation aims to provide different views of the software, from a high-level structural view to low-level descriptions of basic modules. It supports a distribution of the software “by apartment” (a functionality or a set of functionalities) intended for developers who would only be interested by part of the services of the toolset.

A high-level architectural view of the Polychrony toolset is given in Figure 8.
Figure 8. The Polychrony toolset high-level architecture
LICIT Exploratory Action (section vide)
5. Software

5.1. Lucid Synchrone

**Participant:** Marc Pouzet [contact].

Lucid Synchrone is a language for the implementation of reactive systems. It is based on the synchronous model of time as provided by Lustre combined with features from ML languages. It provides powerful extensions such as type and clock inference, type-based causality and initialization analysis and allows to arbitrarily mix data-flow systems and hierarchical automata or flows and valued signals.


The language was used, from 1996 to 2006 as a laboratory to experiment various extensions of the language Lustre. Several programming constructs (e.g. merge, last, mix of data-flow and control-structures like automata), type-based program analysis (e.g., typing, clock calculus) and compilation methods, originally introduced in Lucid Synchrone are now integrated in the new SCADE 6 compiler developed at Esterel-Technologies and commercialized since 2008.

Three major release of the language has been done and the current version is V3 (dev. in 2006). The language is still used for teaching and in our research but we do not develop it anymore. Nonetheless, we have integrated several features from Lucid Synchrone in new research prototypes described below.

5.2. ReactiveML

**Participants:** Mehdi Dogguy, Louis Mandel [contact], Cédric Pasteur.

ReactiveML is a programming language dedicated to the implementation of interactive systems as found in graphical user interfaces, video games or simulation problems. ReactiveML is based on the synchronous reactive model due to Boussinot, integrated in an ML language (Objective Caml).

The Synchronous reactive model provides synchronous parallel composition and dynamic features like the dynamic creation of processes. In ReactiveML, the reactive model is integrated at the language level (not as a library) which leads to a safer and a more natural programming paradigm.

ReactiveML is distributed at URL [http://www.lri.fr/~mandel/rml](http://www.lri.fr/~mandel/rml). The compiler is distributed under the terms of the Q Public License and the library is distributed under the terms of the GNU Library General Public License. The development of ReactiveML started at the University Paris 6 (from 2002 to 2006).

The language was mainly used for the simulation of mobile ad hoc networks at the University Paris 6 and for the simulation of sensor networks at France Telecom and Verimag (CNRS, Grenoble).

5.3. Heptagon

**Participants:** Cédric Pasteur [contact], Brice Gelineau, Léonard Gérard, Adrien Guatto, Cédric Pasteur, Marc Pouzet.

Heptagon is an experimental language for the implementation of embedded real-time reactive systems. It is developed inside the Synchronics large-scale initiative, in collaboration with INRIA Rhônes-Alpes. It is essentially a subset of Lucid Synchrone, without type inference, type polymorphism and higher-order. It is thus a Lustre-like language extended with hierchical automata in a form very close to SCADE 6. The intention for making this new language and compiler is to develop new aggressive optimization techniques for sequential C code and compilation methods for generating parallel code for different platforms. This explains much of the simplifications we have made in order to ease the development of compilation techniques.
Some extensions have already been made, most notably automata. It’s currently used to experiment with linear typing for arrays and also to introduce a concept of asynchronous parallel computations. The compiler developed in our team generates C, java and VHDL code.

Heptagon is jointly developed by Gwenael Delaval and Alain Girault from the INRIA POP ART team (Grenoble).

5.4. Lucy-n

Participants: Louis Mandel [contact], Adrien Guatto, Marc Pouzet.

http://www.lri.fr/~mandel/lucy-n

Lucy-n is a language to program in the n-synchronous model. The language is similar to Lustre with a buffer construct. The Lucy-n compiler ensures that programs can be executed in bounded memory and automatically computes buffer sizes. Hence this language allows to program Kahn networks, the compiler being able to statically compute bounds for all FIFOs in the program.

5.5. ML-Sundials

Participants: Timothy Bourke, Marc Pouzet [contact].

ML-Sundials library provides an Ocaml interface to the Sundials numerical suite 10 (version 2.4.0). This library is used for solving and initial value problem and includes a zero-crossing detection mechanism. Only the CVODE solver with serial nvectors is currently supported. The structure and naming conventions largely follow the original libraries, both for ease of reading the existing documentation and for converting existing source code, but several changes have been made for programming convenience, namely:

- solver sessions are configured through algebraic data types rather than through multiple function calls,
- error conditions are signalled by exceptions rather than return codes (including in user-supplied callback routines),
- closures (partial applications of higher-order functions) are used to share user data between callback routines, and,
- explicit free commands are not necessary nor provided since Ocaml is a garbage-collected language.

The library is in use in a new synchronous hybrid language we are currently developing.

5.6. GCC

Participants: Albert Cohen [contact], Tobias Grosser, Antoniu Pop, Konrad Trifunovic, Feng Li, Riyadh Baghdadi, Cupertino Miranda.

http://gcc.gnu.org

Licence: GPLv3+ and LGPLv3+

The GNU Compiler Collection includes front ends for C, C++, Objective-C, Fortran, Java, Ada, and Go, as well as libraries for these languages (libstdc++, libgcj,...). GCC was originally written as the compiler for the GNU operating system. The GNU system was developed to be 100% free software, free in the sense that it respects the user’s freedom.

PARKAS contributes to the polyhedral compilation framework, also known as Graphite. We also distribute an experimental branch for a stream-programming extension of OpenMP, parallel data-flow programming, and automatic parallelization to a data-flow runtime or architecture. This experiment borrows key design elements to synchronous data-flow languages.

10 https://computation.llnl.gov/casc/sundials/main.html
Tobias Grosser is the maintainer of the Graphite optimization pass of GCC.

5.7. isl

**Participants:** Sven Verdoolaege [contact], Tobias Grosser, Albert Cohen.

http://freshmeat.net/projects/isl

Licence: LGPLv2.1+

isl is a library for manipulating sets and relations of integer points bounded by linear constraints. Supported operations on sets include intersection, union, set difference, emptiness check, convex hull, (integer) affine hull, integer projection, transitive closure (and over-approximation), computing the lexicographic minimum using parametric integer programming. It also includes an ILP solver based on generalized basis reduction. isl also supports affine transformations for polyhedral compilation.
5. Software

5.1. NBac

**Participant:** Bertrand Jeannet.

NBac (Numerical and Boolean Automaton Checker) \(^{13}\) is a verification/slicing tool for reactive systems containing combination of Boolean and numerical variables, and continuously interacting with an external environment. NBac can also handle the same class of hybrid systems as the HyTech tool \(^{76}\). It aims at handling efficiently systems combining a non-trivial numerical behaviour with a complex logical (Boolean) behaviour.

NBac is connected to two input languages: the synchronous dataflow language Lustre, and a symbolic automaton-based language, Autoc/Auto, where a system is defined by a set of symbolic hybrid automata communicating via valued channels. It can perform reachability analysis, co-reachability analysis, and combination of the above analyses. The result of an analysis is either a verdict to a verification problem, or a set of states together with a necessary condition to stay in this set during an execution. NBac is founded on the theory of abstract interpretation.

It has been used for verification and debugging of Lustre programs \(^{79}\) \(^{61}\). It is connected to the Lustre toolset \(^{14}\). It has also been used for controller synthesis of infinite-state systems. The fact that the analyses are approximated results simply in the obtention of a possibly non-optimal controller. In the context of conformance testing of reactive systems, it is used by the test generator STG \(^{49}\) \(^{80}\) for selecting test cases.

5.2. Prometheus

**Participant:** Gregor Goessler.

The BIP component model (Behavior, Interaction model, Priority) \(^{72}\) \(^{5}\) has been designed to support the construction of heterogeneous embedded systems involving different models of computation, communication, and execution, at different levels of abstraction. By separating the notions of behavior, interaction model, and execution model, it enables both heterogeneous modeling, and separation of concerns.

The verification and design tool Prometheus \(^{71}\) implements the BIP component framework. Prometheus is regularly updated to implement new developments in the framework and the analysis algorithms. It has allowed us to carry out several complex case studies from the system-on-chip and bioinformatics domains \(^{11}\).

5.3. Implementations of Synchronous Programs

**Participant:** Alain Girault.

---

\(^{13}\) [http://pop-art.inrialpes.fr/people/bjeannet/nbac/](http://pop-art.inrialpes.fr/people/bjeannet/nbac/)

\(^{14}\) [http://www-verimag.imag.fr/Lustre-V6.html](http://www-verimag.imag.fr/Lustre-V6.html)

5.3.1. Fault Tolerance

We have been cooperating for several years with the INRIA team AOSTE (INRIA Sophia-Antipolis and Rocquencourt) on the topic of fault tolerance and reliability of safety critical embedded systems. In particular, we have implemented several new heuristics for fault tolerance and reliability within their software SYNDEx\(^{16}\). Our first scheduling heuristic produces static multiprocessor schedules tolerant to a specified number of processor and communication link failures [64]. The basic principles upon which we rely to make the schedules fault tolerant is, on the one hand, the active replication of the operations [65], and on the other hand, the active replication of communications for point-to-point communication links, or their passive replication coupled with data fragmentation for multi-point communication media (i.e., buses) [66]. Our second scheduling heuristic is multi-criteria: it produces a static schedule multiprocessor schedule such that the reliability is maximized, the power consumption is minimized, and the execution time is minimized [3][17]. Our results on fault tolerance are summarized in a web page [17].

5.4. Apron and BddApron Libraries

**Participant:** Bertrand Jeannet.

5.4.1. Principles

The APRON library\(^{18}\) is dedicated to the static analysis of the numerical variables of a program by abstract interpretation [51]. Many abstract domains have been designed and implemented for analysing the possible values of numerical variables during the execution of a program (see Figure 1). However, their API diverge largely (datatypes, signatures, ...), and this does not ease their diffusion and experimental comparison w.r.t. efficiency and precision aspects.

The APRON library aims to provide:

- a uniform API for existing numerical abstract domains;
- a higher-level interface to the client tools, by factorizing functionalities that are largely independent of abstract domains.

From an abstract domain designer point of view, the benefits of the APRON library are:

- the ability to focus on core, low-level functionalities;
- the help of generic services adding higher-level services for free.

For the client static analysis community, the benefits are a unified, higher-level interface, which allows experimenting, comparing, and combining abstract domains.

In 2011, the Taylor1plus domain [62], which is the underlying abstract domain of the tool FLUCTUAT [58] has been improved. Glue code has also been added to enable the connection of an abstract domain implemented in OCaml to the APRON infrastructure written in C (this requires callbacks from C to OCaml that are safe w.r.t. garbage collection). This will enable the integration in APRON of the MaxPlus polyhedra library written by X. Allamigeon [38] in the context of the ANR ASOPT project.

The BDDAPRON library\(^{19}\) aims at a similar goal, by adding finite-types variables and expressions to the concrete semantics of APRON domains. It is built upon the APRON library and provides abstract domains for the combination of finite-type variables (Booleans, enumerated types, bitvectors) and numerical variables (integers, rationals, floating-point numbers). It first allows to manipulate expressions that freely mix, using BDDs and MTBDDs, finite-type and numerical APRON expressions and conditions. It then provides abstract domains that combines BDDs and APRON abstract values for representing invariants holding on both finite-type variables and numerical variables.

\(^{16}\) http://www-rocx.inria.fr/syndex

\(^{17}\) http://pop-art.inrialpes.fr/~girault/Projets/FT

\(^{18}\) http://apron.cri.ensmp.fr/library/

\(^{19}\) http://pop-art.inrialpes.fr/~bjeannet/bjeannet-forge/bddapron/index.html
5.4.2. Implementation and Distribution

The APRON library (Fig. 2) is written in ANSI C, with an object-oriented and thread-safe design. Both multi-precision and floating-point numbers are supported. A wrapper for the OCAML language is available, and a C++ wrapper is on the way. It has been distributed since June 2006 under the LGPL license and available at http://apron.cri.ensmp.fr. Its development has still progressed much since. There are already many external users (ProVal/Démons, LRI Orsay, France — CEA-LIST, Saclay, France — Analysis of Computer Systems Group, New-York University, USA — Sierum software analysis platform, Kansas State University, USA — NEC Labs, Princeton, USA — EADS CCR, Paris, France — IRIT, Toulouse, France) and is currently packaged as a REDHAT and DEBIAN package.

The BDDAPRON library is written in OCAML, using polymorphism features of OCAML to make it generic. It is also thread-safe. It provides two different implementations of the same domain, each one presenting pros and cons depending on the application. It is currently used by the CONCURINTERPROC interprocedural and concurrent program analyzer.

5.5. Prototypes

5.5.1. Logical Causality

Participants: Lacramioara Astefanoaei, Gregor Goessler [contact person].

We have developed LoCA, a new prototype tool written in Scala that implements the analysis of logical causality described in 6.6.2. LoCA currently supports causality analysis in BIP. The core analysis engine is implemented as an abstract class, such that support for other models of computation (MOC) can be added by instantiating the class with the basic operations of the MOC.

5.5.2. Automatic Controller Generation

Participants: Emil Dumitrescu, Alain Girault [contact person].

We have developed a software tool chain to allow the specification of models, the controller synthesis, and the execution or simulation of the results. It is based on existing synchronous tools, and thus consists primarily in the use and integration of SIGALI20 and Mode Automata21. It is the result of a collaboration with Eric Rutten from the SARDÉS team.

---

20 http://www.irisa.fr/vertecs/Logiciels/sigali.html
21 http://www-verimag.imag.fr
Useful component templates and relevant properties can be materialized, on one hand by libraries of task models, and, on the other hand, by properties and synthesis objectives.

5.5.3. Rapture

**Participant**: Bertrand Jeannet.

RAPTURE\textsuperscript{22} \cite{jeannet2003rapture} \cite{jeannet2005rapture} is a verification tool that was developed jointly by BRICS (Denmark) and INRIA in years 2000–2002. The tool is designed to verify reachability properties on Markov Decision Processes (MDP), also known as Probabilistic Transition Systems. This model can be viewed both as an extension to classical (finite-state) transition systems extended with probability distributions on successor states, or as an extension of Markov Chains with non-determinism. We have developed a simple automata language that allows the designer to describe a set of processes communicating over a set of channels à la CSP. Processes can also manipulate local and global variables of finite type. Probabilistic reachability properties are specified by defining two sets of initial and final states together with a probability bound. The originality of the tool is to provide two reduction techniques that limit the state space explosion problem: automatic abstraction and refinement algorithms, and the so-called essential states reduction.

5.5.4. The Interproc family of static analyzers

**Participants**: Bertrand Jeannet [contact person], Pascal Sotin.

These analyzers and libraries are of general use for people working in the static analysis and abstract interpretation community, and serve as an experimental platform for the ANR project ASOPT (see § 8.1.2).

a generic fix-point engine written in OCAML. It allows the user to solve systems of fix-point equations on a lattice, using a parameterized strategy for the iteration order and the application of widening. It also implements recent techniques for improving the precision of analysis by alternating

\textsuperscript{22} http://pop-art.inrialpes.fr/people/bjeannet/rapture/rapture.html
post-fixpoint computation with widening and descending iterations in a sound way [70].

a simple interprocedural static analyzer that infers properties on the numerical variables of programs in a toy language. It is aimed at demonstrating the use of the previous library and the above-described APRON library, and more generally at disseminating the knowledge in abstract interpretation. It is also deployed through a web-interface 25. It is used as the experimental platform of the ASOPT ANR project.

\textsc{FixpointInterproc} \textsc{ConcurInterproc} extends Interproc with concurrency, for the analysis of multithreaded programs interacting via shared global variables. It is also deployed through a web-interface 26.

\textsc{Pinterproc} extends Interproc with pointers to local variables. It is also deployed through a web-interface 27.

5.5.5. Heptagon/BZR

\textbf{Participant:} Gwenaël Delaval.

\textsc{Heptagon} is a dataflow synchronous language, inspired from \textsc{Lucid Synchrone} 28. Its compiler is meant to be simple and modular, allowing this language to be a good support for the prototyping of compilation methods of synchronous languages. It is developped within the \textsc{SynchroneNria} large-scale action. \textsc{Heptagon} has been used to built BZR 29, which is an extension of the former with contracts constructs. These contracts allow to express dynamic temporal properties on the inputs and outputs of \textsc{Heptagon} node. These properties are then enforced, within the compilation of a BZR program, by discrete controller synthesis, using the \textsc{Sigali} tool 30. The synthesized controller is itself generated in \textsc{Heptagon}, allowing its analysis and compilation towards different target languages (C, \textsc{Java}, \textsc{VHDL}).

\begin{itemize}
    \item [25] http://pop-art.inrialpes.fr/interproc/interprocweb.cgi
    \item [26] http://pop-art.inrialpes.fr/interproc/concurinterprocweb.cgi
    \item [27] http://pop-art.inrialpes.fr/interproc/pinterprocweb.cgi
    \item [28] http://www.di.ens.fr/~pouzet/lucid-synchrone
    \item [29] http://bzr.inria.fr
\end{itemize}
5. Software

5.1. Mica: A Modal Interface Compositional Analysis Toolbox

Participant: Benoît Caillaud.

http://www.irisa.fr/s4/tools/mica/

Mica is an Ocaml library developed by Benoît Caillaud implementing the Modal Interface algebra published in [18]. The purpose of Modal Interfaces is to provide a formal support to contract based design methods in the field of system engineering. Modal Interfaces enable compositional reasoning methods on I/O reactive systems.

In Mica, systems and interfaces are represented by extension. However, a careful design of the state and event heap enables the definition, composition and analysis of reasonably large systems and interfaces. The heap stores states and events in a hash table and ensures structural equality (there is no duplication). Therefore complex data-structures for states and events induce a very low overhead, as checking equality is done in constant time.

Thanks to the Inter module and the mica interactive environment, users can define complex systems and interfaces using Ocaml syntax. It is even possible to define parameterized components as Ocaml functions.

Mica is available as an open-source distribution, under the CeCILL-C Free Software License Agreement (http://www.cecill.info/licenses/Licence_CeCILL-C_V1-en.html).

5.2. Synet: A General Petri-Net Synthesis Toolbox

Participant: Benoît Caillaud.

http://www.irisa.fr/s4/tools/synet/

Synet is a software tool for the synthesis of bounded and unbounded Petri-nets, based on the theory of regions [31]. It can synthesize Petri-nets from automata or regular expression and can be configured by command-line options to synthesize nets modulo graph isomorphism or language equality. Petri nets computed by Synet can be displayed using the GraphViz 2D graph layout software, or saved to a file for further transformation and analysis.

The tool actually implements two linear-algebraic synthesis methods: A first method uses the simplex algorithm and the second one is based on the computation of extremal rays of polyhedral cones, using Chernikova’s algorithm [34]. Both methods imply that the input graphs are given by extension. Nevertheless, Synet yields good performances on many practical use-cases and is the only tool supporting unbounded net synthesis.

The main application of Synet is the synthesis of communicating distributed protocols and controllers [30]. Synthesis is constrained to produce so-called distributables nets [33], a class of nets that can be turned into networks of communicating automata by automated methods. This allows to divide the synthesis problem in two steps: Given the specification of a protocol as a finite automaton, (i) synthesize (if exists) a distributable net, and then (ii) derive a network of communicating automata from the distributable net. While the second step is automatic and straightforward, the first step is in essence a computer assisted design task, where the distributed Petri-net synthesis algorithm helps the designer to refine the protocol specification into a graph isomorphic to the marking graph of a distributable net.
TRIO Project-Team

5. Software

5.1. MPIGate: Multi-Protocols Interface and Gateway for telehomecare and environment monitoring and control

Participants: Shahram Nourizadeh, Hugo Cruz Sanchez, Ye-Qiong Song.

For developing AAL (Ambient Assisted Living) or more generally the environment monitoring and control systems, heterogeneous wireless and wired networks will be used. To solve firstly the interoperability problems, and then to ensure the application required QoS, we developed a software prototype called MPIGate. MPIGate includes two important components: a user interface for telehomecare and home automation, and a gateway for ensuring the interworking of the different networks. In 2010, MPIGate has been laureate of the 12th national contest for the creation of innovative technology companies by the ministry of higher education and research (“Emergence” category). During 2011, MPIGate has been implemented on an embedded linux board and integrated into LORIA smart room platform within CPER IS project (http://infositu.loria.fr) [25], [45]. In its current version, the gateway ensures the communication between IP (Ethernet and Wifi), home automation network (KNX), Bluetooth and Zigbee. Heterogeneous sensors can be now easily used through MPIGate interface for further building the activity monitoring of the elderly person living along at home or other application scenarios.

5.2. SAMOVAR

Participants: Adrien Guénard, Lionel Havet, Françoise Simonot-Lion.

Wireless Sensor and Actuator Networks (WSANs) combine sensors and actuators interconnected by wireless networks in order to perform distributed sensing and acting tasks. Closed-loop controllers can therefore be deployed on WSANs. Such systems have to meet specific requirements in terms of performance, dependability, energy and cost which raises great challenges due to the unreliability of wireless communications. A way to ensure that a system meets the required properties is to model it and go through its analysis. Building a model requires both deep knowledge on the system as well as on the used framework. Therefore there is a need for frameworks well-suited to the targeted systems and to the properties to verify. We proposed an approach meeting these conditions and a simulation framework, Samovar, based on Matlab / Simulink, allowing the modeling of the network protocols (Mac and routing services) and the resources sharing policy thanks to the TrueTime toolbox. Several classes of components (application, nodes, networks and middleware) and a clear semantics for their composition are identified. Furthermore, the design of Samovar was also driven by the need to easily transfer software component model between the concrete systems and its simulated model. The modeling and simulation method as well as the Samovar framework were assessed on several case studies: cooperating robots, intelligent living environment, embedded controllers on UAV robots... The simulation framework is available from http://samovar.loria.fr/. This work is supported by INRIA through the ADT SAMOVAR.

5.3. ANR Open-PEOPLE platform

Participants: Sophie Alexandre, Jonathan Ponroy, Kévin Roussel, Olivier Zendra.

The aim of Open-PEOPLE is to provide a platform for estimating and optimizing the power and energy consumption of systems. The Open-PEOPLE project formally started in April 2009. Two systems administrator and software developers had been hired initially: Sophie Alexandre and Kévin Roussel. Another system administrator and software developer, Jonathan Ponroy, joined them in 2010 when he finished his work on the ANR MORE project where he worked previously. Sophie Alexandre contract ended in February 2011.
Since the beginning of the Open-PEOPLE project, we had made significant progress in setting up the infrastructure for the software part of the platform, for which INRIA Nancy Grand Est is responsible. We had included new features to be able to fully integrate and test software developed as Eclipse plugins, relying on the Buckminster tool. We had also created a specific extension set for SVN and Hudson, called OPCIM (Open-PEOPLE Continuous Integration Mechanism). OPCIM had been registered at APP on 13/04/2010 with number IDDN.FR.001.150008.000.S.P.2010.000.10000.

Concerning the Open-PEOPLE platform itself, we had first tackled the high-level work, working with our partners on the definition of the requirements of the platform according to the needs of industry. We had then realized the specification work to define the global perimeter of our platform, according to the previous requirements. As part of this work had also been designed exchanges formats between the various tools. We had also designed at INRIA Nancy Grand Est a Tools integration Protocol, which specified requirements for external tools to be integrated in our platform. All this design work had been materialized in several reports which were deliveries provided to ANR.

We had also designed and developed an authentication component (Eclipse plugin) for the platform, so as to be able to provide a unique, secured access gate to the platform to all the tools that are or shall be integrated into it.

We had also started and almost finished developing an Internet portal giving access and control to the Open-PEOPLE Hardware Platform, located at our partner’s UBS in Lorient. Our portal features included user account management facilities, on the admin side, and on the user side, the ability to create, save, edit, reuse and of course submit jobs, make reservations for the hardware platform resources and get back tests results.

Finally, we had started working on two important parts of the software platform.

First, a way to unify the user experience despite the fact the platform federates several tools which were not developed to interact together. This implied an important and in-depth study of the wanted ergonomics for the platform, which involved taking into account both user needs and habits and the features of the available software tools.

The second work which had begun in 2011 was the design (then implementation) of the communications of between the various tools of the platform. This skeleton will be a key part of our platform, and the quality of its design will have a tremendous impact on its maintainability and its extensibility.

Note that the Open-PEOPLE project had been successfully evaluated on 14/09/2010 by ANR. Developments done during the first two years in the project are detailed in the 2009 and 2010 activity reports. In 2011, these developments went on.

We continued the work to solidify our development platform supporting our work and that of our partners. We produced a finer grained definition of the software platform functionalities, and a more precise definition of the tools integration protocol. We worked towards the corresponding implementation documents, adding two new deliverables about the architecture of the software platform and the ergonomics of the software platform. For the latter, we extensively interviewed user about ergonomics and designed several GUI mockups. We progressed on the implementation of the software platform, especially with respect to the internet portal to remote-control the hardware platform. We participated to the definition of the hardware platform and its functionalities, and participated actively to the work on the Specification document for HW / SW interfacing. We provided the first concrete design and implementation of the HW/SW platform interfacing, with our implementation of the remote control portal for the HW platform. This remote control module was completed in Fall 2011.

We also participated to the work pertaining to basic components model homogenization, by reviewing this in the context of the software platform architecture and implementation, which resulted in several incremental improvements of the underlying models. Finally, progressing towards the first release of the software part of the Open-PEOPLE platform, we realized an ergonomic study for the consumption laws editors, with mockups and user interviews and validation. We worked on the implementation of the editors for the consumption laws, which required learning new environments and development tools (related to the EMF framework and the AADL, QUDV and MathML models). As a consequence, we completed the implementation of the GUI and
engine to create units and quantities. We finalized the architecture needed to integrate external modules in the platform.

With this progress, the first release of the whole Open-PEOPE software platform is expected early 2012.

5.4. VITRAIL

Participants: Damien Bodenes, Pierre Caserta, Olivier Zendra.

The aim of the VITRAIL operation is to provide tools for the advanced and immersive visualization of programs. It partners with the University of Montréal, University of Montpellier and Pareo team of INRIA Nancy Grand Est.

Last years, in VITRAIL, we had developed software to instrument and trace Java programs at the bytecode level. We then had developed an analysis tool able to exploit these traces to compute relevant software metrics. We had hired Damien Bodenes as software developer, and had begun the work on a prototype able to render a 3D world, symbolizing software, onto various visualization hardware, with the possibility to change the display metaphor. The main part of our development work had been in 2009 the choice and validation of the technology, and a first architecture. In 2010, the development had go on at a good pace, building on chosen technologies and architecture. This had brought new experience, and with the first actual runs of our platform, we had realized that with the Irrlicht platform we had chosen, we could reach unforeseeable problem when scaling up. We had thus decided to reverse our choice to the Ogre3D 3D engine at the beginning of 2010. Our development had then progressed steadily.

We had released in 2010 a first prototype of our platform, with all the underlying architecture, able to provide navigation features and interaction capacities limited to the driving of the navigation, as per our plans. This had included dual screen management.

Our first prototype, using 2 large 2D screens, with a city metaphor, had been demonstrated during the "Fête de la Science" in November 2010 and had received a lot of attention and enthusiasm from the general public. About 55 persons per day had visited our booth and got demonstrations.

We had also progressed significantly in our Java bytecode tracer, by improving its granularity, the completeness of the traced information, and its performance as well. We have a unique tool which is able to trace both program classes and JDK classes, at basic block level. In addition, it does so with a dynamic instrumentation of classes, which means there is no need to have an instrumented version of the class files on disk. This is very convenient, especially when changing machine of JVM, or when upgrading either the JDK or the program itself. In addition, the performance is good enough that the instrumented programs are still fully usable in an interactive way, without bothering the user. To the best of our knowledge, this is the only Java bytecode tracer that offers these features nowadays.

Our software development had lead to several registrations with APP:

- VITRAIL - Visualizer had been first registered on 29/12/2009 under number IDDN.FR.001.530021.000.S.P.2009.000.10000.
- VITRAIL - Tracer, was registered at APP on 20/09/2010 with number IDDN.FR.001.380001.000.S.P.2010.000.10000.

In 2011, we acquired a workstation and three 30 inches computer screens, to be able to set up a "boxed 3D workstation", that would provide display in front and on both sides of the operator. This would constitute the next step in our experiments, by improving immersion with a larger field of vision (on the sides). The software developments to do this are ongoing. We also integrated a WiiMote interaction device to our system, but our experiments found that its spacial resolution was too poor for our needs.

We finally improved significantly our VITRAIL prototype in 2011, especially by designing and implementing a new representation for the relations between software (hence visual) elements, with limited clutter and the possibility to regroup links and see their direction.
VASY Project-Team

5. Software

5.1. The CADP Toolbox

Participants: Iker Bellicot, Hubert Garavel [contact person], Yann Genevois, Rémi Hérilier, Frédéric Lang, Radu Mateescu, Christine McKinty, Wendelin Serwe, Damien Thivolle.

We maintain and enhance CADP (Construction and Analysis of Distributed Processes – formerly known as CÆSAR/ALDÉBARAN Development Package) [9], a toolbox for protocols and distributed systems engineering (see http://cadp.inria.fr). In this toolbox, we develop and maintain the following tools:

- **CÆSAR.ADT** [3] is a compiler that translates LOTOS abstract data types into C types and C functions. The translation involves pattern-matching compiling techniques and automatic recognition of usual types (integers, enumerations, tuples, etc.), which are implemented optimally.
- **CÆSAR** [11] is a compiler that translates LOTOS processes into either C code (for rapid prototyping and testing purposes) or finite graphs (for verification purpose). The translation is done using several intermediate steps, among which the construction of a Petri net extended with typed variables, data handling features, and atomic transitions.
- **OPEN/CÆSAR** [4] is a generic software environment for developing tools that explore graphs on the fly (for instance, simulation, verification, and test generation tools). Such tools can be developed independently of any particular high level language. In this respect, OPEN/CÆSAR plays a central role in CADP by connecting language-oriented tools with model-oriented tools. OPEN/CÆSAR consists of a set of 16 code libraries with their programming interfaces, such as:
  - **CAESAR_GRAPH**, which provides the programming interface for graph exploration,
  - **CAESAR_HASH**, which contains several hash functions,
  - **CAESAR_SOLVE**, which resolves boolean equation systems on the fly,
  - **CAESAR_STACK**, which implements stacks for depth-first search exploration, and
  - **CAESAR_TABLE**, which handles tables of states, transitions, labels, etc.

A number of tools have been developed within the OPEN/CÆSAR environment, among which:

- **BISIMULATOR**, which checks bisimulation equivalences and preorders,
- **CUNCTATOR**, which performs on-the-fly steady-state simulation of continuous-time Markov chains,
- **DETERMINATOR**, which eliminates stochastic nondeterminism in normal, probabilistic, or stochastic systems,
- **DISTRIBUTOR**, which generates the graph of reachable states using several machines,
- **EVALUATOR**, which evaluates regular alternation-free μ-calculus formulas,
- **EXECUTOR**, which performs random execution,
- **EXHIBITOR**, which searches for execution sequences matching a given regular expression,
- **GENERATOR**, which constructs the graph of reachable states,
- **PROJECTOR**, which computes abstractions of communicating systems,
- **REDUCTOR**, which constructs and minimizes the graph of reachable states modulo various equivalence relations,
- **SIMULATOR, XSIMULATOR, and OCIS**, which allow interactive simulation, and
– TERMINATOR, which searches for deadlock states.

• BCG (Binary Coded Graphs) is both a file format for storing very large graphs on disk (using efficient compression techniques) and a software environment for handling this format. BCG also plays a key role in CADP as many tools rely on this format for their inputs/outputs. The BCG environment consists of various libraries with their programming interfaces, and of several tools, such as:
  – BCG_DRAW, which builds a two-dimensional view of a graph,
  – BCG_EDIT, which allows to modify interactively the graph layout produced by BCG_DRAW,
  – BCG_GRAPH, which generates various forms of practically useful graphs,
  – BCG_INFO, which displays various statistical information about a graph,
  – BCG_IO, which performs conversions between BCG and many other graph formats,
  – BCG_LABELS, which hides and/or renames (using regular expressions) the transition labels of a graph,
  – BCG_MERGE, which gathers graph fragments obtained from distributed graph construction,
  – BCG_MIN, which minimizes a graph modulo strong or branching equivalences (and can also deal with probabilistic and stochastic systems),
  – BCG_STEADY, which performs steady-state numerical analysis of (extended) continuous-time Markov chains,
  – BCG_TRANSIENT, which performs transient numerical analysis of (extended) continuous-time Markov chains, and
  – XTL (eXecutable Temporal Language), which is a high level, functional language for programming exploration algorithms on BCG graphs. XTL provides primitives to handle states, transitions, labels, successor and predecessor functions, etc.

For instance, one can define recursive functions on sets of states, which allow to specify in XTL evaluation and diagnostic generation fixed point algorithms for usual temporal logics (such as HML [60], CTL [53], ACTL [55], etc.).

• The connection between explicit models (such as BCG graphs) and implicit models (explored on the fly) is ensured by OPEN/CÆSAR-compliant compilers, e.g.:
  – BCG_OPEN, for models represented as BCG graphs,
  – CÆSAR.OPEN, for models expressed as LOTOS descriptions,
  – EXP.OPEN, for models expressed as communicating automata,
  – FSP.OPEN, for models expressed as FSP [66] descriptions,
  – LNT.OPEN, for models expressed as LOTOS NT descriptions, and
  – SEQ.OPEN, for models represented as sets of execution trace.

The CADP toolbox also includes TGV (Test Generation based on Verification), developed by the VERIMAG laboratory (Grenoble) and the VERTECS project team at INRIA Rennes.

The CADP tools are well-integrated and can be accessed easily using either the EUCALYPTUS graphical interface or the SVL [6] scripting language. Both EUCALYPTUS and SVL provide users with an easy and uniform access to the CADP tools by performing file format conversions automatically whenever needed and by supplying appropriate command-line options as the tools are invoked.

5.2. The TRAIAN Compiler

Participants: Hubert Garavel [contact person], Frédéric Lang.
We develop a compiler named TRAIAN for translating descriptions written in the LOTOS NT language (see § 3.2) into C programs, which will be used for simulation, rapid prototyping, verification, and testing.

The current version of TRAIAN performs lexical analysis, syntactic analysis, abstract syntax tree construction, static semantics analysis, and C code generation for LOTOS NT types and functions.

Although this version of TRAIAN is still incomplete (it does not handle LOTOS NT processes), it already has useful applications in compiler construction [8]. The recent compilers developed by the VASY project team — including AL, EVALUATOR 4.0 (see § 6.1.6), EXP.OPEN 2.0 (see § 6.1.4), LNT2LOTOS (see § 6.2.2), NTIF (see § 3.2), PIC2LNT (see § 6.2.3), and SVL (see § 6.1.4) — all contain a large amount of LOTOS NT code, which is then translated into C code by TRAIAN.

Our approach consists in using the SYNTAX tool (developed at INRIA Rocquencourt) for lexical and syntactic analysis together with LOTOS NT for semantical aspects, in particular the definition, construction, and traversal of abstract trees. Some involved parts of the compiler can also be written directly in C if necessary. The combined use of SYNTAX, LOTOS NT, and TRAIAN proves to be satisfactory, in terms of both the rapidity of development and the quality of the resulting compilers.

The TRAIAN compiler can be freely downloaded from the VASY Web site (see http://vasy.inria.fr/traian).
5. Software

5.1. TGV

Participant: Thierry Jéron.

TGV (Test Generation with Verification technology) is a tool for test generation of conformance test suites from specifications of reactive systems [4]. It is based on the IOLTS model, a well defined theory of testing, and on-the-fly test generation algorithms coming from verification technology. Originally, TGV allows test generation focused on well defined behaviors formalized by test purposes. The main operations of TGV are (1) a synchronous product which identifies sequences of the specification accepted by a test purpose, (2) abstraction and determinisation for the computation of next visible actions, (3) selection of test cases by the computation of reachable states from the initial states and co-reachable states from accepting states. TGV has been developed in collaboration with Vérimag Grenoble and uses libraries of the CADP toolbox (VERIMAG and VASY). TGV can be seen as a library that can be linked to different simulation tools through well defined APIs. An academic version of TGV is distributed in the CADP toolbox and allows test generation from Lotos specifications by a connection to its simulator API. TGV has been registered at APP (Agence de Protection des Programmes) under deposit number IDDN.FR.001.310012.00.R.P.1997.000.2090.

5.2. STG

Participant: Thierry Jéron.

STG (Symbolic Test Generation) is a prototype tool for the generation and execution of test cases using symbolic techniques. It takes as input a specification and a test purpose described as IOSTS, and generates a test case program also in the form of IOSTS. Test generation in STG is based on a syntactic product of the specification and test purpose IOSTS, an extraction of the subgraph corresponding to the test purpose, elimination of internal actions, determinisation, and simplification. The simplification phase now relies on NBAC, which approximates reachable and coreachable states using abstract interpretation. It is used to eliminate unreachable states, and to strengthen the guards of system inputs in order to eliminate some Inconclusive verdicts. After a translation into C++ or Java, test cases can be executed on an implementation in the corresponding language. Constraints on system input parameters are solved on-the-fly (i.e. during execution) using a constraint solver. The first version of STG was developed in C++, using Omega as constraint solver during execution. This version has been deposited at APP under number IDDN.FR.001.510006.000.S.P.2004.000.10600. A new version in OCaml has been developed in the last years. This version is more generic and will serve as a library for symbolic operations on IOSTS. Most functionalities of the C++ version have been re-implemented. Also a new translation of abstract test cases into Java executable tests has been developed, in which the constraint solver is LUCKYDRAW (VERIMAG). This version has also been deposit at APP and is available for download on the web as well as its documentation and some examples.

Finally, in collaboration with ULB, we implemented a prototype SMACS, derived from STG, that is devoted to the control of infinite system modeled by STS.

5.3. SIGALI

Participant: Hervé Marchand.
SIGALI is a model-checking tool that operates on ILTS (Implicit Labeled Transition Systems, an equational representation of an automaton), an intermediate model for discrete event systems. It offers functionalities for verification of reactive systems and discrete controller synthesis. It is developed jointly by the ESPRESSO and VERTECS teams. The techniques used consist in manipulating the system of equations instead of the set of solutions, which avoids the enumeration of the state space. Each set of states is uniquely characterized by a predicate and the operations on sets can be equivalently performed on the associated predicates. Therefore, a wide spectrum of properties, such as liveness, invariance, reachability and attractivity, can be checked. Algorithms for the computation of predicates on states are also available [6] [28]. SIGALI is connected with the Polychrony environment (ESPRESSO project-team) as well as the Matou environment (VERIMAG), thus allowing the modeling of reactive systems by means of Signal Specification or Mode Automata and the visualization of the synthesized controller by an interactive simulation of the controlled system. SIGALI is registered at APP.
ABSTRACTION Project-Team

5. Software

5.1. The Apron Numerical Abstract Domain Library

Participants: Antoine Miné [correspondent], Bertrand Jeannet [team PopArt, INRIA-RA].

The APRON library is dedicated to the static analysis of the numerical variables of a program by abstract interpretation. Its goal is threefold: provide ready-to-use numerical abstractions under a common API for analysis implementers, encourage the research in numerical abstract domains by providing a platform for integration and comparison of domains, and provide a teaching and demonstration tool to disseminate knowledge on abstract interpretation.

The APRON library is not tied to a particular numerical abstraction but instead provides several domains with various precision versus cost trade-offs (including intervals, octagons, linear equalities and polyhedra). A specific C API was designed for domain developers to minimize the effort when incorporating a new abstract domain: only few domain-specific functions need to be implemented while the library provides various generic services and fallback methods (such as scalar and interval operations for most numerical data-types, parametric reduced products, and generic transfer functions for non-linear expressions). For the analysis designer, the APRON library exposes a higher-level API with C, C++, OCaml, and Java bindings. This API is domain-neutral and supports a rich set of semantic operations, including parallel assignments (useful to analyze automata), substitutions (useful for backward analysis), non-linear numerical expressions, and IEEE floating-point arithmetic.

The APRON library is freely available on the web at http://apron.cri.ensmp.fr/library; it is distributed under the LGPL license and is hosted at INRIAForge. Packages exist for the Debian and Fedora Linux distributions. In order to help disseminate the knowledge on abstract interpretation, a simple inter-procedural static analyzer for a toy language is included. An on-line version is deployed at http://pop-art.inrialpes.fr/interproc/interprocweb.cgi.

The APRON library is developed since 2006 and currently consists of 130,000 lines of C, C++, OCaml, and Java.

Current and past external library users include the Constraint team (LINA, Nantes, France), the Proval/Démon team (LRI Orsay, France), the Analysis of Computer Systems Group (New-York University, USA), the Sierum software analysis platform (Kansas State University, USA), NEC Labs (Princeton, USA), EADS CCR (Paris, France), IRIT (Toulouse, France), ONERA (Toulouse, France), CEA LIST (Saclay, France), VERIMAG (Grenoble, France), ENSMP CRI (Fontainebleau, France), the IBM T.J. Watson Research Center (USA), the University of Edinburgh (UK).

5.2. The Astrée Static Analyzer of Synchronous Software

Participants: Patrick Cousot [project scientifique leader, correspondent], Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, Xavier Rival.

ASTRÉE is a static analyzer for sequential programs based on abstract interpretation [37]. [28]. [38]. [30].

The ASTRÉE static analyzer [27]. [43] [1] www.astree.ens.fr aims at proving the absence of runtime errors in programs written in the C programming language.

ASTRÉE analyzes structured C programs, with complex memory usages, but without dynamic memory allocation nor recursion. This encompasses many embedded programs as found in earth transportation, nuclear energy, medical instrumentation, and aerospace applications, in particular synchronous control/command. The whole analysis process is entirely automatic.
ASTRÉE discovers all runtime errors including:

- undefined behaviors in the terms of the ANSI C99 norm of the C language (such as division by 0 or out of bounds array indexing);
- any violation of the implementation-specific behavior as defined in the relevant Application Binary Interface (such as the size of integers and arithmetic overflows);
- any potentially harmful or incorrect use of C violating optional user-defined programming guidelines (such as no modular arithmetic for integers, even though this might be the hardware choice);
- failure of user-defined assertions.

The analyzer performs an abstract interpretation of the programs being analyzed, using a parametric domain (ASTRÉE is able to choose the right instantiation of the domain for wide families of software). This analysis produces abstract invariants, which over-approximate the reachable states of the program, so that it is possible to derive an over-approximation of the dangerous states (defined as states where any runtime error mentioned above may occur) that the program may reach, and produces alarms for each such possible runtime error. Thus the analysis is sound (it correctly discovers all runtime errors), yet incomplete, that is it may report false alarms (i.e., alarms that correspond to no real program execution). However, the design of the analyzer ensures a high level of precision on domain-specific families of software, which means that the analyzer produces few or no false alarms on such programs.

In order to achieve this high level of precision, ASTRÉE uses a large number of expressive abstract domains, which allow expressing and inferring complex properties about the programs being analyzed, such as numerical properties (digital filters, floating-point computations), Boolean control properties, and properties based on the history of program executions.

ASTRÉE has achieved the following two unprecedented results:

- **A340–300.** In Nov. 2003, ASTRÉE was able to prove completely automatically the absence of any RTE in the primary flight control software of the Airbus A340 fly-by-wire system, a program of 132,000 lines of C analyzed in 1h20 on a 2.8 GHz 32-bit PC using 300 MB of memory (and 50mn on a 64-bit AMD Athlon 64 using 580 MB of memory).

- **A380.** From Jan. 2004 on, ASTRÉE was extended to analyze the electric flight control codes then in development and test for the A380 series. The operational application by Airbus France at the end of 2004 was just in time before the A380 maiden flight on Wednesday, 27 April, 2005.

These research and development successes have led to consider the inclusion of ASTRÉE in the production of the critical software for the A350. ASTRÉE is currently industrialized by AbsInt Angewandte Informatik GmbH and is commercially available.

### 5.3. The AstréeA Static Analyzer of Asynchronous Software

**Participants:** Patrick Cousot [project scientifique leader, correspondant], Radhia Cousot, Jérôme Feret, Antoine Miné, Xavier Rival.

ASTRÉEA is a static analyzer prototype for parallel software based on abstract interpretation [39], [40], [32]. It started with support from THÉSÉE ANR project (2006–2010) and is continuing within the ASTRÉEA project (2012–2015).

The ASTRÉEA prototype www.astrea.ens.fr is a fork of the ASTRÉE static analyzer (see 5.2) that adds support for analyzing parallel embedded C software.
ASTRÉE A analyzes C programs composed of a fixed set of threads that communicate through a shared memory and synchronization primitives (mutexes, FIFOs, blackboards, etc.), but without recursion nor dynamic creation of memory, threads nor synchronization objects. ASTRÉE A assumes a real-time scheduler, where thread scheduling strictly obeys the fixed priority of threads. Our model follows the ARINC 653 OS specification used in embedded industrial aeronautical software. Additionally, ASTRÉE A employs a weakly-consistent memory semantics to model memory accesses not protected by a mutex, in order to take into account soundly hardware and compiler-level program transformations (such as optimizations). ASTRÉE A checks for the same run-time errors as ASTRÉE, with the addition of data-races.

Compared to ASTRÉE, ASTRÉE A features: a new iterator to compute thread interactions, a refined memory abstraction that takes into account the effect of interfering threads, and a new scheduler partitioning domain. This last domain allows discovering and exploiting mutual exclusion properties (enforced either explicitly through synchronization primitives, or implicitly by thread priorities) to achieve a precise analysis.

ASTRÉE A is currently being applied to analyze a large industrial avionic software: 1.6 MLines of C and 15 threads, completed with a 2,500-line model of the ARINC 653 OS developed for the analysis. The analysis currently takes 29h on a 2.66 GHz 64-bit intel server using one core and generates around 1,800 alarms. The low computation time (only a few times larger than the analysis time by ASTRÉE of synchronous programs of a similar size and structure) shows the scalability of the approach (in particular, we avoid the usual combinatorial explosion associated to thread interleavings). Precision-wise, the result, while not as impressive as that of ASTRÉE, is quite encouraging. Improvements were made this year concerning the precision of ASTRÉE A (from 7,600 alarms in 2010 to 1,800 now) and will continue within the scope of the ASTRÉE A ANR project (Section 8.1.1.2).

The details of the analysis are described in [22].

5.4. OpenKappa

Participants: Monte Brown [Harvard Medical School], Vincent Danos [University of Edinburgh], Jérôme Feret [Correspondent], Walter Fontana [Harvard Medical School], Russ Harmer [Harvard Medical School], Jean Krivine [Paris VII].

OpenKappa is a collection of tools to build, debug and run models of biological pathways. It contains a compiler for the Kappa Language [49], a static analyzer [48] (for debugging models), a simulator [47], a compression tool for causal traces [46], and a model reduction tool [4], [45], [50].

OpenKappa is developed since 2007 and, the OCaml version currently consists of 46 000 lines of OCaml. Software are available in OCaml and in Java. Moreover, an Eclipse plugin is available. OpenKappa is freely available on the web at http://kappalanguage.org under the LGPL license. Discussion groups are also available on line.

Current external users include the Ecole Polytechnique Federale de Lausanne, the UNAM-Genomics Mexico team. It is used as pedagogical material in graduate lessons at Harvard Medical School, and at the Interdisciplinary Approaches to Life science (AIV) Master Program (Université de Médecine Paris-Descartes).

5.5. Translation Validation

Participant: Xavier Rival [correspondent].

The main goal of this software project is to make it possible to certify automatically the compilation of large safety critical software, by proving that the compiled code is correct with respect to the source code: When the proof succeeds, this guarantees that no compiler bug did cause incorrect code be generated. Furthermore, this approach should allow to meet some domain specific software qualification criteria (such as those in DO-178 regulations for avionics software), since it allows proving that successive development levels are correct with respect to each other i.e., that they implement the same specification. Last, this technique also justifies the use of source level static analyses, even when an assembly level certification would be required, since it establishes separately that the source and the compiled code are equivalent.
The compilation certification process is performed automatically, thanks to a prover designed specifically. The automatic proof is done at a level of abstraction which has been defined so that the result of the proof of equivalence is strong enough for the goals mentioned above and so that the proof obligations can be solved by efficient algorithms.

The current software features both a C to Power-PC compilation certifier and an interface for an alternate source language frontend, which can be provided by an end-user.

5.6. Zarith

Participants: Antoine Miné [Correspondent], Xavier Leroy [INRIA Paris-Rocquencourt], Pascal Cuoq [CEA LIST].

ZARITH is a small (10K lines) OCaml library that implements arithmetic and logical operations over arbitrary-precision integers. It is based on the GNU MP library to efficiently implement arithmetic over big integers. Special care has been taken to ensure the efficiency of the library also for small integers: small integers are represented as Caml unboxed integers and use a specific C code path. Moreover, optimized assembly versions of small integer operations are provided for a few common architectures.

ZARITH is an open-source project hosted at OCamlForge (http://forge.ocamlcore.org/projects/zarith) and distributed under a modified LGPL license.

ZARITH is currently used in the ASTRÉE analyzer to enable the sound analysis of programs featuring 64-bit (or larger) integers. It is also used in the Frama-C analyzer platform developed at CEA LIST and INRIA Saclay.
ATEAMS Project-Team

5. Software

5.1. AmbiDexter

Participants: Bas Basten [correspondent], Jurgen Vinju.

Characterization: A-3-up4, SO-4, SM-2-up3, EM-2-up3, SDL-4-up5, OC-DA-3-CD-3-MS-3-TPM-3.
WWW: http://homepages.cwi.nl/~basten/ambiguity/
Objective: Statically detect ambiguity of context-free grammars for programming languages, as fast and precise as possible.
Users: Authors of context-free grammars of programming languages in SDF2, Rascal, ANTLR, etc
Impact: This is the first usable ambiguity detection tool, aiming to solve the Achilles’ heel of context-free general parsing.
Competition: AmbiDexter is the fastest and most accurate tool currently available.
Engineering: AmbiDexter was developed by one person and will be maintained by another. It is 25 LOC in Java and distributed as a component of the Rascal IDE.
Publications: [14], [9] [2], [1]

5.2. Derric

Participants: Tijs van der Storm, Jeroen van den Bos [correspondent].

Characterization: A-2-up3, SO-4, SM-2-up3, EM-3, SDL-3-up4, OC-DA-3-CD-3-MS-3-TPM-3.
WWW: http://svn.rascal-mpl.org/derric/
Objective: Encapsulate all the variability in the construction of so-called “carving” algorithms, then generate the fastest and most accurate implementations. Carving algorithms recover information that has been deleted or otherwise scrambled on digital media such as hard-disks, usb sticks and mobile phones.
Users: Digital forensic investigation specialists
Impact: Derric has the potential of revolutionizing the carving area. It does in 1500 lines of code what other systems need tens of thousands of lines for with the same accuracy. Derric will be an enabler for faster, more specialized and more successful location of important evidence material.
Competition: Derric competes in a small market of specialized open-source and commercial carving tools.
Engineering: Derric is a Rascal program of 1.5 kloc designed by two persons.
Publications: [27], [13]

5.3. Pacioli

Participants: Tijs van der Storm, Paul Griffioen [correspondent].

Characterization: A-2-up3, SO-4, SM-2, EM-3, SDL-3-up4, OC-DA-3-CD-3-MS-3-TPM-3.
WWW: http://svn.rascal-mpl.org/pacioli/
Objective: Encapsulate all the variability in the construction of modeling and analysis tools in computational auditing
Users: Financial auditing experts
Impact: Pacioli is an experiment with a big potential in the field of computational auditing. It operates as a vehicle now for experimenting with new ideas in this field. The goal is to tackle the enormous complexity in the (trading) of companies using high level modeling and analysis techniques.
Competition: Pacioli competes with less specialized and less formal business analysis tooling, mostly based on spreadsheets.
Engineering: Pacioli is a part Java, part Rascal project written by one person.
5.4. Rascal

**Participants:** Paul Klint, Jurgen Vinju [correspondent], Tijs van der Storm, Bas Basten, Jeroen van den Bos, Mark Hills, Bert Lisser, Arnold Lankamp, Atze van der Ploeg, Vadim Zaytsev, Anastasia Izmaylova, Anya Helene Bagge.

Characterization: A5, SO-4, SM-4, EM-4, SDL-4-up5, OC-DA-3-CD-3-MS-3-TPM-3.

WWW: [http://www.rascal-mpl.org](http://www.rascal-mpl.org)

Objective: Provide a completely integrated programming language parametric meta programming language for the construction of any kind of meta program for any kind of programming language: analysis, transformation, generation, visualization.

Users: Researchers in model driven engineering, programming languages, software engineering, software analysis, as well as practitioners that need specialized tools.

Impact: Rascal is making the mechanics of meta programming into a non-issue. We can now focus on the interesting details of the particular fact extraction, model, source analysis, domain analysis as opposed to being distracted by the engineering details. Simple things are easy in Rascal and complex things are manageable, due to the integration, the general type system and high-level programming features.

Competition: There is a plethora of meta programming toolboxes and frameworks available, ranging from plain parser generators to fully integrated environments. Rascal is distinguished because it is a programming language rather than a specification formalism and because it completely integrates different technical domains (syntax definition, term rewriting, relational calculus). For simple tools, Rascal competes with scripting languages and for complex tools it competes context-free general parser generators, with query engines based on relational calculus and with term rewriting and strategic programming languages.

Engineering: Rascal is about 100 kLOC of Java code, designed by a core team of three and with a team of around 8 phd students and post-docs contributing to its design, implementation and maintenance. The goal is to work towards more bootstrapping and less Java code as the project continues.

Publications: [21], [28], [29], [22][6], [7]

5.4.1. Novelties

- Re-design of embedded grammar formalism including semantic disambiguation facilities.
- Extremely fast top-down context-free general parsing algorithm in cubic time and space.
- Parse error reporting via partial parse trees (useful in incremental syntax highlighting and incremental type analysis).
- Auto-indent feature for code generation templates.
- Significant extensions and improvements of software visualization library, such as hierarchical graphs and smaller set of more powerful primitives for charts and interactive features.
- Significant improvements to online documentation and interactive tutor environment.
- “ToLaTex” mode to include Rascal code in papers.
- ShellExec library for inter-acting via pipes with external programs
- Bridge to Maude and K.
- Generalized function dispatch to arbitrary pattern dispatch.
- New module composition mechanism “extend” next to “import”.
- Ambiguity diagnostics library and parse tree visualizations as a first step towards more grammarware in the IDE.
- A command-line interface to run a single Rascal program.
• Fixed a number of memory leaks in the IDE.
• IDE features for mixed Java/Rascal projects.
• Rational numbers.
• Formal concept analysis library.
• Enhanced SDF2 to Rascal translation.
• Redesigned and simplified abstract grammar format.
• Added “break”, “continue” and “fail” statements for back-tracking and continuation control.
• Radically changed internal design from Visitor to Interpreter design pattern (using an automated refactoring).

5.5. IDE Meta-tooling Platform

Participants: Jurgen Vinju [correspondent], Arnold Lankamp, Anya Helene Bagge.

IMP, the IDE meta tooling platform is an Eclipse plugin developed mainly by the team of Robert M. Fuhrer at IBM TJ Watson Research institute. It is both an abstract layer for Eclipse, allowing rapid development of Eclipse based IDEs for programming languages, and a collection of meta programming tools for generating source code analysis and transformation tools.

Characterization: A5, SO-3, SM4-up5, EM-4, SDL-5, DA-2-CD-2-MS-2-TPM-2

WWW: http://www.eclipse.org/imp

Objective: The IDE Meta Tooling Platform (IMP) provides a high-level abstraction over the Eclipse API such that programmers can extend Eclipse with new programming languages or domain specific languages in a few simple steps. IMP also provides a number of standard meta tools such as a parser generator and a domain specific language for formal specifications of configuration parameters.

Users: Designers and implementers of IDEs for programming languages and domain specific languages. Also, designers and implementers of meta programming tools.

Impact: IMP is popular among meta programmers especially for it provides the right level of abstraction.

Competition: IMP competes with other Eclipse plugins for meta programming (such as Model Driven Engineering tools), but its API is more general and more flexible. IMP is a programmers framework rather than a set of generators.

Engineering: IMP is a long-lived project of many contributors, which is managed as an Eclipse incubation project at eclipse.org.

Publications: [3]

Jurgen Vinju and Arnold Lankamp contribute significantly to the development of IMP. Their effort is focused on the maintenance and optimization of a general purpose symbolic representation library for source code artifacts, called “PDB”. PDB stands for Program DataBase. For more information, please visit http://www.eclipse.org/imp.

The Rascal language itself was accepted by Eclipse as a contribution to the IMP project. This will further strengthen the collaboration between the IMP and the Rascal team as well as generate a wider audience for Rascal.
5.6. **Ensō**

**Participant:** Tijs van der Storm [correspondent].

**Characterization:** A5, SO-4, SM-3-up-4, EM-2-up-4, SDL-4, OC-DA-4-CD-4-MS-4-TPM-4

**WWW:** http://www.enso-lang.org

**Objective:** Together with Prof. Dr. William R. Cook of the University of Texas at Austin, Tijs van der Storm has been designing and implementing a new programming system, called Ensō. Ensō is theoretically sound and practical reformulation of model-based development. It is based on model-interpretation as opposed to model transformation and code generation. Currently, the system already supports models for schemas (data models), web applications, context-free grammars, diagram editors and security.

**Users:** All programmers.

**Impact:** Ensō has the potential to revolutionize the activity of programming. By looking at model driven engineering from a completely fresh perspective, with as key ingredients interpreters and partial evaluation, it may make higher level (domain level) program construction and maintenance as effective as normal programming.

**Competition:** Ensō competes as a programming paradigm with model driven engineering tools and generic programming and languages that provide syntax macros and language extensions.

**Engineering:** Ensō is less than 7000 lines of (bootstrapped) Ruby code.

5.7. **Software Language Processing Suite**

**Participant:** Vadim Zaytsev [correspondent].

**Characterization:** A3-up4, SO-4, SM-3, EM-2up3, SDL-2, OC-DA-4-CD-4-MS-4-TPM-4

**WWW:** http://slps.sourceforge.net

**Objective:** The project facilitates exposition and comparison of approaches and techniques on language processing.

**Users:** Computer science students, teachers, engineers and practitioners

**Impact:** SLPS contains the largest collection of grammars for programming languages directly recovered from documentation, as well as the largest collection of source-to-source grammar formalisms translators and other related grammarware.

**Engineering:** SLPS is a large collection of scripts and programs written by Ralf Lämmel and Vadim Zaytsev.

### 5.7.1. Novelties

- New grammars: Ada, Dart, Eiffel, Fortran, Modula, Mediawiki, ...(now a total of 41)
- Grammar Tank: a new collection of 54 small grammars for research purposes
- TestMatch: a tool for grammar-based differential testing of ANTLR grammars and for nonterminal matching based on parsing generated test data (in collaboration with Ralf Lämmel).
- Grammar Hunter: a tool for automated notation-parametric grammar recovery (will also be a Rascal library).

5.8. **Demo Light for Composing Models**

**Participants:** Jan van Eijck [correspondent], Floor Sietsma.

**Characterization:** A2,SO-3,SM-1,EM-2,SDL-2,OC-4

**WWW:** http://homepages.cwi.nl/~jve/software/demolight0/

**Objective:** Demonstrate epistemic modeling and reasoning

**Users:** Students and researchers in application of epistemic logic

**Impact:** Demo light makes the theory of epistemic reasoning insightful by offering a Haskell library for experimenting with it.

**Engineering:** Demo Light is a Haskell library.
5. Software

5.1. Morphus/MMDEX
   An anti-virus software based on morphological analysis, Dépôt APP du logiciel MMDEX, 2009, IDDN.FR.001.300033.000.R.P.2009.000.10000

5.2. PYMS
   Online disassembler. http://pym86.appspot.com/

5.3. TraceSurfer

5.4. Crème Brûlée
   Crème Brûlée is an experimental Javascript dynamic instrumentation engine. http://code.google.com/p/cremebrulee/
5. Software

5.1. Protocol Verification Tools

Participants: Pierre-Cyrille Héam, Olga Kouchnarenko, Michaël Rusinowitch, Mathieu Turuani, Laurent Vigneron.

5.1.1. AVISPA

Cassis has been one of the 4 partners involved in the European project AVISPA, which has resulted in the distribution of a tool for automated verification of security protocols, named AVISPA Tool. It is freely available on the web and it is well supported. The AVISPA Tool compares favourably to related systems in scope, effectiveness, and performance, by (i) providing a modular and expressive formal language for specifying security protocols and properties, and (ii) integrating 4 back-ends that implement automatic analysis techniques ranging from protocol falsification (by finding an attack on the input protocol) to abstraction-based verification methods for both finite and infinite numbers of sessions.

5.1.2. CL-AtSe

We develop, as a first back-end of AVISPA, CL-AtSe, a Constraint Logic based Attack Searcher for cryptographic protocols. The CL-AtSe approach to verification consists in a symbolic state exploration of the protocol execution, for a bounded number of sessions. This necessary restriction (for decidability, see [85]) allows CL-AtSe to be correct and complete, i.e., any attack found by CL-AtSe is a valid attack, and if no attack is found, then the protocol is secure for the given number of sessions. Each protocol step is represented by a constraint on the protocol state. These constraints are checked lazily for satisfiability, where satisfiability means reachability of the protocol state. CL-AtSe includes a proper handling of sets (operations and tests), choice points, specification of any attack states through a language for expressing secrecy, authentication, fairness, non-abuse freeness, advanced protocol simplifications and optimizations to reduce the problem complexity, and protocol analysis modulo the algebraic properties of cryptographic operators such as XOR (exclusive or) and Exp (modular exponentiation). The handling of XOR and Exp has required to implement an optimized version of the combination algorithm of Baader & Schulz [76] for solving unification problems in disjoint unions of arbitrary theories.

CL-AtSe has been successfully used [75] to analyse France Telecom R&D, Siemens AG, IETF, or Gemalto protocols in funded projects. It is also employed by external users, e.g., from the AVISPA’s community. Moreover, CL-AtSe achieves very good analysis times, comparable and sometimes better than state-of-the-art tools in the domain (see [90] for tool details and precise benchmarks).

5.1.3. TA4SP

We have developed, as a second back-end of AVISPA, TA4SP (Tree Automata based on Automatic Approximations for the Analysis of Security Protocols), an automata based tool dedicated to the validation of security protocols for an unbounded number of sessions. This tool provides automatic computations of over- and under-approximations of the knowledge accessible by an intruder. This knowledge is encoded as a regular tree language and protocol steps and intruder abilities are encoded as a term rewriting system. When given a reachability problem such as secrecy, TA4SP reports that (1) the protocol is safe if it manages to compute an over-approximation of intruder’s knowledge that does not contain a secret term or (2) the protocol is unsafe in the rewrite model if it manages to compute an underapproximation of intruder’s knowledge containing a secret term or (3) I don’t know otherwise. TA4SP has verified 28 industrial protocols and case (3) occurred only once, for Kaochow protocol version 2.

http://www.avispa-project.org
TA4SP handles protocols using operators with algebraic properties. Thanks to a recent quadratic completion algorithm new experimental results have been obtained, for example for the Encrypted Key Exchange protocol (EKE2) using the exponential operator. Recently, TA4SP was used in [89] to analyse a hierarchy of authentication properties.

5.2. Testing Tools

Participants: Fabrice Bouquet, Frédéric Dadeau, Philippe Paquelier.

In December 2008, we have started the redevelopment of our original testing tools environment, with two objectives: first, refactoring the existing developments, and, second, providing an open platform aiming at gathering together the various developments, increasing the reusability of components. The resulting platform, named Hydra, is a Eclipse-like platform, based on Plug-ins architecture. Plug-ins can be of five kinds: parser is used to analyze source files and build an intermediate format representation of the source; translator is used to translate from a format to another or to a specific file; service denotes the application itself, i.e. the interface with the user; library denotes an internal service that can be used by a service, or by other libraries; tool encapsulates an external tool. The following services have been developed so far:

- BZPAnimator: performs the animation of a BZP model (a B-like intermediate format);
- Angluin: makes it possible to perform a machine learning algorithm (à la Angluin) in order to extract an abstraction of a system behavior;
- UML2SMT: aims at extracting first order logic formulas from the UML Diagrams and OCL code of a UML/OCL model to check them with a SMT solver.

These services involve various libraries (sometimes reusing each other), and rely on several tool plug-ins that are: SMTProver (encapsulating Z3 solver), PrologTools (encapsulating CLPS-B solver), Grappa (encapsulating a graph library). We are currently working on transferring the existing work on test generation from B abstract machines, JML, and statecharts using constraint solving techniques.

5.3. Collaborative Tools

Participants: Abdessamad Imine, Asma Cherif.

The collaborative tools allow us to manage collaborative works on shared documents using flexible access control models. These tools have been developed in order to validate and evaluate our approach on combining collaborative edition with optimistic access control.

- **P2PEdit.** This prototype is implemented in Java and supports the collaborative editing of HTML pages and it is deployed on P2P JXTA platform[3]. In our prototype, a user can create a HTML page from scratch by opening a new collaboration group. Other users (peers) may join the group to participate in HTML page editing, as they may leave this group at any time. Each user can dynamically add and remove different authorizations for accessing to the shared document according the contribution and the competence of users participating in the group. Using JXTA platform, users exchange their operations in real-time in order to support WYSIWIS (What You See Is What I See) principle. Furthermore, the shared HTML document and its authorization policy are replicated at the local memory of each user. To deal with latency and dynamic access changes, an optimistic access control technique is used where enforcement of authorizations is retroactive.

- **P2PCalendar.** To extend our collaboration and access control models to mobile devices, we implemented a shared calendar on iPhone OS which is decentralized and scalable (i.e. it can be used over both P2P and ad-hoc networks) [58]. This application aims to make a collaborative calendar where users can simultaneously modify events (or appointments) and control access on events. The access rights are determined by the owner of an event. The owner decides who is allowed to access the event and what privileges they have. Likewise to our previous tool, the calendar and its authorization policy are replicated at every mobile device.

---

5.4. Other Tools

Several software tools described in previous sections are using tools that we have developed in the past. For instance BZ-TT uses the set constraints solver CLPS. Note that the development of the SMT prover haRVey has been stopped. The successor of haRVey is called veriT and is developed by David Déharbe (UFRN Natal, Brasil) and Pascal Fontaine (Veridis team).
4. Software

4.1. Javalib

Participants: Frédéric Besson [correspondant], David Pichardie, Vincent Monfort.

Javalib is an efficient library to parse Java .class files into OCaml data structures, thus enabling the OCaml programmer to extract information from class files, to manipulate and to generate valid .class files.

See also the web page http://sawja.inria.fr/.

- Version: 2.2
- Programming language: Ocaml

4.2. SAWJA

Participants: Frédéric Besson [correspondant], David Pichardie, Vincent Monfort.

Sawja is a library written in OCaml, relying on Javalib to provide a high level representation of Java bytecode programs. It name comes from Static Analysis Workshop for JAva. Whereas Javalib is dedicated to isolated classes, Sawja handles bytecode programs with their class hierarchy and with control flow algorithms.

Moreover, Sawja provides some stackless intermediate representations of code, called JBir and A3Bir. The transformation algorithm, common to these representations, has been formalized and proved to be semantics-preserving.

See also the web page http://sawja.inria.fr/.

- Version: 1.2
- Programming language: Ocaml

4.3. Timbuk

Participant: Thomas Genet [correspondant].

Timbuk is a library of OCAML functions for manipulating tree automata. More precisely, Timbuk deals with finite bottom-up tree automata (deterministic or not). This library provides the classical operations over tree automata (intersection, union, complement, emptiness decision) as well as exact or approximated sets of terms reachable by a given term rewriting system. This last operation can be certified using a checker extracted from a Coq specification.

- Version: 3.1
- Programming language: Ocaml
5. Software

5.1. A model checker for the probabilistic asynchronous $\pi$-calculus

**Participants:** Miguel Andrés [correspondant], Catuscia Palamidessi.

In collaborations with Dave Parker and Marta Kwiatkowska, we are developing a model checker for the probabilistic asynchronous $\pi$-calculus. Case studies with Fair Exchange and MUTE, an anonymous peer-to-peer file sharing system, are in progress.

Technically we use MMC as a compiler to encode the probabilistic $\pi$-calculus into certain PRISM representation, which will then be verified against PCTL using PRISM. The transitional semantics defined in MMC can be reused to derive the symbolic transition graphs of a probabilistic process. The code for derivation will work as an add-on to MMC under XSB and invoke a graph traversal to enumerate all reachable nodes and transitions of the probabilistic process.

In the meanwhile we are also attempting a direct and more flexible approach to the development of a model checker for the probabilistic $\pi$-calculus, using OCaml. This should allow to extend the language more easily, to include cryptographic primitives and other features useful for the specification of security protocols. As the result of our preliminary steps in this direction we have developed a rudimentary model checker, available at the following URL: [http://vamp.gforge.inria.fr/](http://vamp.gforge.inria.fr/) .

5.2. PRISM model generator

**Participants:** Konstantinos Chatzikokolakis [correspondant], Catuscia Palamidessi.

This software generates PRISM models for the Dining Cryptographers and Crowds protocols. It can also use PRISM to calculate the capacity of the corresponding channels. More information can be found in [33] and in the file README file width instructions at the URL [http://www.lix.polytechnique.fr/comete/software/README-anonmodels.html](http://www.lix.polytechnique.fr/comete/software/README-anonmodels.html).

The software can be downloaded at [http://www.lix.polytechnique.fr/comete/software/anonmodels.tar.gz](http://www.lix.polytechnique.fr/comete/software/anonmodels.tar.gz). These scripts require Perl to run and have been tested in Linux. The GUI of the corners tool also requires the Perl/Tk library. Finally some parts of the model generator tool require PRISM and gnuplot to be installed.

5.3. Calculating the set of corner points of a channel

**Participants:** Konstantinos Chatzikokolakis [correspondant], Catuscia Palamidessi.

The corner points can be used to compute the maximum probability of error and to improve the Hellman-Raviv and Santhi-Vardy bounds. More information can be found in [34] and in the file README file width instructions at the URL [http://www.lix.polytechnique.fr/comete/software/README-corners.html](http://www.lix.polytechnique.fr/comete/software/README-corners.html).

The software can be downloaded at [http://www.lix.polytechnique.fr/comete/software/corners.tar.gz](http://www.lix.polytechnique.fr/comete/software/corners.tar.gz). These scripts require Perl to run and have been tested in Linux. The GUI of the corners tool also requires the Perl/Tk library. Finally some parts of the model generator tool require PRISM and gnuplot to be installed.

5.4. MMCsp, a compiler for the $\pi$-calculus

**Participants:** Peng Wu [correspondant], Catuscia Palamidessi.

MMCsp is a compiler from a simple probabilistic $\pi$-calculus to PRISM ([http://www.prismmodelchecker.org/manual/Main/Introduction](http://www.prismmodelchecker.org/manual/Main/Introduction)), models. It is built on XSB ([http://xsbs.sourceforge.net/](http://xsbs.sourceforge.net/)), a tabled logic programming system, and generates the symbolic semantic representation of a probabilistic pi-calculus term in text. A separate Java program then translates this semantic representation into a probabilistic model for PRISM.
The tool was developed by Peng Wu during his postdoc period in Comète in the context of the collaboration between the teams Comète and PRISM under the INRIA/ARC Project ProNoBib (http://www.lsv.ens-cachan.fr/~goubault/ProNobis/index.html). It is based on the papers [41] and [38].

The source code is free and can be downloaded from http://www.cs.ucl.ac.uk/staff/p.wu/mmc_sp_manual.html.
5. Software

5.1. BIOCHAM

**Participants:** François Fages, Steven Gay, Dragana Jovanovska, Aurélien Rizk, Sylvain Soliman.

The Biochemical Abstract Machine BIOCHAM [29] is a modeling environment for systems biology distributed as open-source since 2003. Current version is v3.3. BIOCHAM uses a compositional rule-based language for modeling biochemical systems, allowing patterns for expressing set of rules in a compact form. This rule-based language is compatible with the Systems Biology Markup Language (SBML) and is interpreted with three semantics corresponding to three abstraction levels:

1. the boolean semantics (presence or absence of molecules),
2. the differential semantics (concentrations of molecules),
3. the stochastic semantics (discrete numbers of molecules).

Based on this formal framework, BIOCHAM features:

- Boolean and numerical simulators (Rosenbrock’s method for the differential semantics, Gillespie’s algorithm with tau lipping for the stochastic semantics);
- a temporal logic language (CTL for qualitative models and QFLTL(R) with numerical constraints for quantitative models) for formalizing biological properties such as reachability, checkpoints, oscillations or stability, and checking them automatically with model-checking techniques;
- automatic search procedures to infer parameter values, initial conditions and even reaction rules from temporal logic properties;
- automatic detection of invariants, through constraint-based analysis of the underlying Petri net;
- an SBGN-compatible reaction graph editor;
- an event handler allowing the encoding of hybrid models and formalisms [30].

BIOCHAM is implemented in GNU-Prolog and interfaced to the symbolic model checker NuSMV and to the continuous optimization tool CMAES developed by the EPI TAO.

5.2. Nicotine

**Participant:** Sylvain Soliman.

Nicotine is a GNU Prolog framework dedicated to the analysis of Petri nets. It was originally built for the computation of invariants using GNU Prolog’s CLP(FD) solver but has been further extended to allow import/export of various Petri nets formats. It provides as independent modules different features that can sometimes also be integrated in BIOCHAM, like SEPI comutation, or left aside, like unambiguous ODE to Petri net conversion, since a more general heuristic conversion is developed for BIOCHAM.

5.3. Spatio-temporal simulation environment (STSE)

**Participant:** Szymon Stoma.

The overall goal of this project is to provide a software platform gathering a set of open-source tools and workflows facilitating spatio-temporal simulations (preferably of biological systems) based on microscopy data. The framework currently contains modules to digitize, represent, analyze, and model spatial distributions of molecules in static and dynamic structures (e.g. growing). A strong accent is put on the experimental verification of biological models by actual, spatio-temporal data acquired using microscopy techniques. Project was initially started at Humboldt University Berlin and moved to INRIA with its founder. Project webpage is: http://stse-software.org.
5.4. YeastTracker  
**Participant:** Jannis Uhlendorf.

YeastTracker is a software to follow single cells in movies and to quantify fluorescent images based on this tracking. It has been developed for yeast cells, but is also applicable to other cells that have a defined round shape. The software is written in Matlab and uses a circular Hough transform and binary integer programming to detect and follow cells. It allows to quantify the mean fluorescence of each cell as well as the co-localization of two different fluorescent markers. The software is available on request (jannis.uhlendorf@inria.fr).

5.5. Rules2CP  
**Participants:** François Fages, Raphaël Martin.

Rules2CP is a rule-based modeling language for constraint programming. It is distributed since 2009 as open-source. Unlike other modeling languages for constraint programming, Rules2CP adopts a single knowledge representation paradigm based on rules without recursion, and a restricted set of data structures based on records and enumerated lists given with iterators. This allows us to model complex constraint satisfaction problems together with search strategies, where search trees are expressed by logical formulae and heuristic choice criteria are defined with preference orderings by pattern-matching on the rules’ left-hand sides.

The expressiveness of Rules2CP has been illustrated in the FP6 Strep project Net-WMS by a complete library for packing problems, called PKML (Packing Knowledge Modeling Library), which, in addition to pure bin packing and bin design problems, can deal with common sense rules about weights, stability, as well as specific packing business rules.

5.6. SiLCC  
**Participant:** Thierry Martinez.

SiLCC is an extensible modular concurrent constraint programming language relying upon linear logic. It is a complete implementation of the Linear logic Concurrent Constraint programming paradigm of Saraswat and Lincoln using the formal semantics of Fages, Ruet and Soliman. It is a single-paradigm logical language, enjoying concurrency, imperative traits, and a clean module system allowing to develop hierarchies of constraint systems within the language.

This software prototype is used to study the design of hierarchies of extensible libraries of constraint solvers. SiLCC is also considered as a possible implementation language for restructuring the code of BIOCHAM.

5.7. EMoP  
**Participant:** Thierry Martinez.

EMoP is an extension of Prolog with first-class modules. These modules have the formal semantics of the LCC modules and provide Prolog with notions of namespaces, closures and objects within a simple programming model. Modules are also the support for user-definition of macros and modular syntax extensions. EMoP is bootstrapped and uses the GNU Prolog compilation chain as back-end.

5.8. CHRat  
**Participant:** Thierry Martinez.

CHRat is a modular version of the well known Constraint Handling Rules language CHR, called for CHRat for CHR with ask and tell. Inspired by the LCC framework, this extension of CHR makes it possible to reuse CHRat components both in rules and guards in other CHRat components, and define hierarchies of constraint solvers. CHRat is a bootstrapped preprocessor for CHR which generates code for SWI/Prolog.
5.9. CLPGUI

**Participant:** François Fages.

CLPGUI is a generic graphical user interface written in Java for constraint logic programming. It is available for GNU-Prolog and SICStus Prolog. CLPGUI has been developed both for teaching purposes and for debugging complex programs. The graphical user interface is composed of several windows: one main console and several dynamic 2D and 3D viewers of the search tree and of finite domain variables. With CLPGUI it is possible to execute incrementally any goal, backtrack or recompute any state represented as a node in the search tree. The level of granularity for displaying the search tree is defined by annotations in the CLP program.

CLPGUI has been mainly developed in 2001 and is distributed as third-party software on GNU-Prolog and SICStus Prolog web sites. In 2009, CLPGUI has been interfaced to Rules2CP/PKML and used in the FP6 Strep Net-WMS with a non-released version.
FORMES Team

5. Software

5.1. aCiNO
Participants: Fei He [correspondent], Min Zhou.

aCiNO is an SMT (Satisfiability Modulo Theory) solver based on a Nelson-Oppen [65] architecture, and written in C++. Currently, two popular theories are considered: linear real arithmetic (LRA) and uninterpreted functions (UF). A lazy approach is used for solving SMT problem. For efficiency consideration, the solver is implemented in an incremental way. It also invokes an online SAT solver, which is now a modified MiniSAT, so that recovery from conflict is possible.

5.2. CoLoR and Rainbow
Participants: Frédéric Blanqui [correspondent], Kim-Quyen Ly, Sidi Ould Biha.

CoLoR is a Coq [44] library on rewriting theory and termination of nearly 70,000 lines of code [11]. It provides definitions and theorems for:

- Mathematical structures: relations, (ordered) semi-rings.
- Data structures: lists, vectors, polynomials with multiple variables, finite multisets, matrices.
- Term structures: strings, algebraic terms with symbols of fixed arity, algebraic terms with varyadic symbols, simply typed lambda-terms.
- Transformation techniques: conversion from strings to algebraic terms, conversion from algebraic to varyadic terms, arguments filtering, rule elimination, dependency pairs, dependency graph decomposition, semantic labelling.
- Termination criteria: polynomial interpretations, multiset ordering, lexicographic ordering, first and higher order recursive path ordering, matrix interpretations.

Rainbow is a tool for automatically certifying termination certificates expressed in the CPF XML format [29] used in the termination competition on termination [32]. Termination certificates are translated and checked in Coq by using the CoLoR library.

CoLoR and Rainbow are distributed under the CeCILL license on http://color.inria.fr/. Various people participated to its development (see the website for more information).

5.3. EDOLA
Participants: Hehua Zhang [correspondent], Ming Gu, Hui Kong, Yu Jiang.

Joint work with Jiaguang Sun (Tsinghua University, China).

EDOLA [26] is an integrated tool for domain-specific modeling and verification of PLC applications [74]. It is based on a domain-specific modeling language to describe system models. It supports both model checking and automatic theorem proving techniques for verification. The goal of this tool is to possess both the usability in domain modeling, the reusability in its architecture and the capability of automatic verification.

For the moment, we have developed a prototype of the EDOLA language, which can easily describe the features of PLC applications like the scan cycle mechanism, the pattern of environment model, time constraints and five property patterns. TLA+ [59] was chosen as the intermediate language to implement the automatic verification of EDOLA models. A prototype of EDOLA has also been developed, which comes along with an editor to help writing EDOLA models. To automatically verify properties on EDOLA models, it provides the interface for both a model checker TLC [59] and a first-order theorem prover SPASS [75].
5.4. Moca

**Participant:** Frédéric Blanqui [correspondant].

Joint work with Pierre Weis (INRIA Rocquencourt) and Richard Bonichon (CEA).

Moca is a construction functions generator for OCaml [60] data types with invariants. It allows the high-level definition and automatic management of complex invariants for data types. In addition, it provides the automatic generation of maximally shared values, independently or in conjunction with the declared invariants.

A relational data type is a concrete data type that declares invariants or relations that are verified by its constructors. For each relational data type definition, Moca compiles a set of construction functions that implements the declared relations.

Moca supports two kinds of relations:

- predefined algebraic relations (such as associativity or commutativity of a binary constructor),
- user-defined rewrite rules that map some pattern of constructors and variables to some arbitrary user’s define expression.

The properties that user-defined rules should satisfy (completeness, termination, and confluence of the resulting term rewriting system) must be verified by a programmer’s proof before compilation. For the predefined relations, Moca generates construction functions that allow each equivalence class to be uniquely represented by their canonical value.

Moca is distributed under QPL on [http://moca.inria.fr/](http://moca.inria.fr/).

5.5. SimSoC

**Participant:** Vania Joloboff [correspondant].

SimSoC is an infrastructure to run simulation models which comes along with a library of simulation models. SimSoC allows its users to experiment various system architectures, study hardware/software partition, and develop embedded software in a co-design environment before the hardware is ready to be used. SimSoC aims at providing high performance, yet accurate simulation, and provide tools to evaluate performance and functional or non-functional properties of the simulated system.

SimSoC is based on SystemC standard and uses Transaction Level Modeling for interactions between the simulation models. The current version of SimSoC is based on the open source libraries from the OSCI Consortium: SystemC version 2.2 and TLM 2.0.1 [54], [33]. Hardware components are modeled as TLM models, and since TLM is itself based on SystemC, the simulation is driven by the SystemC kernel. We use standard, unmodified, SystemC (version 2.2), hence the simulator has a single simulation loop.

The second open source version of SimSoC, SimSoC v0.7.1, has been released in November 2010. It contains a full simulator for ARM V5 and PowerPC both running at an average speed of about 80 Millions instructions per second in, and a simulator for the MIPS architecture with an average speed of 20 Mips in mode DT1. It represents about 70,000 lines of source code and includes:

- Instruction Set Simulators. The ARM Version 5 architecture has been implemented with DT0, DT1, DT2 mode. The ARM and PowerPC 600 architecture with DT0 and DT1 mode. For both architectures, complete simulation models of the processor and MMU are provided, making it possible to run operating systems of the simulated platform. MIPS architecture in DT0 mode is under development.
- A dynamic translator from binary programs to an internal representation. For the ARM architecture a compiler has been developed that generates the C++ translated code (for DT2), using parametrized specialization options.
• Peripheral models including a serial line controller, a flash memory controller, an interrupt controller.
• A utility to generate permanent storage for flash memory simulation; a compiler tool to generate instruction binary decoder.
• Examples illustrating the use of the library and infrastructure.

SimSoC is distributed under LGPL on https://gforge.inria.fr/projects/simsoc .

5.6. SimSoC-Cert

Participants: Frédéric Blanqui, Vania Joloboff, Jean-François Monin [correspondant], Xiaomu Shi.

SimSoC-Cert is a set of tools that can automatically generate in various target languages (Coq and C) the decoding functions and the state transition functions of each instruction and addressing mode of the ARMv6 architecture manual [28] (implemented by the ARM11 processor family) but the Thumb and coprocessor instructions. The input of SimSoC-Cert is the ARMv6 architecture manual itself.

Based on this, we first developed simlight (8000 generated lines of C, plus 1500 hand-written lines of C), a simulator for ARMv6 programs using no peripheral and no coprocessor. Next, we developed simlight2, a fast ARMv6 simulator integrated inside a SystemC/TLM module, now part of SimSoC v0.7.

We can also generate similar programs for SH4 [31] but this is still under test.
GALLIUM Project-Team

5. Software

5.1. OCaml

Participants: Xavier Leroy [correspondant], Xavier Clerc [team SED], Damien Doligez, Alain Frisch [LexiFi], Jacques Garrigue [Nagoya University], Maxence Guédon [team SED], Luc Maranget [EPI Moscova], Michel Mauny [ENSTA], Nicolas Pouillard, Pierre Weis [EPI Estime].

OCaml, formerly known as Objective Caml, is our flagship implementation of the Caml language. From a language standpoint, it extends the core Caml language with a fully-fledged object and class layer, as well as a powerful module system, all joined together by a sound, polymorphic type system featuring type inference. The OCaml system is an industrial-strength implementation of this language, featuring a high-performance native-code compiler for several processor architectures (IA32, AMD64, PowerPC, ARM, etc) as well as a bytecode compiler and interactive loop for quick development and portability. The OCaml distribution includes a standard library and a number of programming tools: replay debugger, lexer and parser generators, documentation generator, compilation manager, and the Camlp4 source pre-processor.

Web site: http://caml.inria.fr/ .

5.2. CompCert C

Participants: Xavier Leroy [correspondant], Sandrine Blazy [EPI Celtique], Alexandre Pilkiewicz.

The CompCert C verified compiler is a compiler for a large subset of the C programming language that generates code for the PowerPC, ARM and x86 processors. The distinguishing feature of CompCert is that it has been formally verified using the Coq proof assistant: the generated assembly code is formally guaranteed to behave as prescribed by the semantics of the source C code. The subset of C supported is quite large, including all C types except float, long, long double, all C operators, almost all control structures (the only exception is unstructured switch), and the full power of functions (including function pointers and recursive functions but not variadic functions). The generated PowerPC code runs 2–3 times faster than that generated by GCC without optimizations, and only 7% (resp. 12%) slower than GCC at optimization level 1 (resp. 2).


5.3. Zenon

Participant: Damien Doligez.

Zenon is an automatic theorem prover based on the tableaux method. Given a first-order statement as input, it outputs a fully formal proof in the form of a Coq proof script. It has special rules for efficient handling of equality and arbitrary transitive relations. Although still in the prototype stage, it already gives satisfying results on standard automatic-proving benchmarks.

Zenon is designed to be easy to interface with front-end tools (for example integration in an interactive proof assistant), and also to be easily retargetted to output scripts for different frameworks (for example, Isabelle).


5.4. Menhir

Participants: François Pottier [correspondant], Yann Régis-Gianas [U. Paris Diderot].

Menhir is a new LR(1) parser generator for Objective Caml. Menhir improves on its predecessor, ocamllexacc, in many ways: more expressive language of grammars, including EBNF syntax and the ability to parameterize a non-terminal by other symbols; support for full LR(1) parsing, not just LALR(1); ability to explain conflicts in terms of the grammar.

MARELLE Project-Team

5. Software

5.1. Semantics

Participant: Yves Bertot [correspondant].

This is a library for the Coq system, where the description of a toy programming language is presented. The value of this library is that it can be re-used in classrooms to teach programming language semantics or the Coq system. The topics covered include introductory notions to domain theory, pre and post-conditions, abstract interpretation, and the proofs of consistency between all these point of views on the same programming language. Standalone tools for the object programming language can be derived from this development. See also the web page http://coq.inria.fr/pylons/pylons/contribs/view/Semantics/v8.3.

• ACM: F3.2 F4.1
• AMS: 68N30
• Programming language: Coq

5.2. Certicrypt

Participants: Gilles Barthe [IMDEA Software institute], Juan Manuel Crespo [IMDEA Software institute], Benjamin Grégoire [correspondant], Sylvain Heraud, César Kunz [IMDEA Software institute], Federico Olmedo [IMDEA Software institute], Santiago Zanella Béguelin [IMDEA Software institute].

CertiCrypt takes a language-based approach to cryptography: the security of a cryptographic scheme and the cryptographic assumptions upon which its security relies are expressed by means of probabilistic programs, called games; in a similar way, adversarial models are specified in terms of complexity classes, e.g. probabilistic polynomial-time programs. This code-centric view leads to statements that are amenable to formalization and tool-assisted verification. CertiCrypt instruments a rich set of verification techniques for probabilistic programs, including equational theories of observational equivalence, relational Hoare logic, data-flow analysis-based program transformations, and game-based techniques such as eager/lazy sampling and failure events. See also the web page http://easycrypt.gforge.inria.fr/.
MEXICO Project-Team

5. Software

5.1. Software

5.1.1. libalf: the Automata Learning Framework

Participant: Benedikt Bollig [correspondent].

libalf is a comprehensive, open-source library for learning finite-state automata covering various well-known learning techniques (such as, Angluin’s L^*, Biermann, and RPNI, as well as a novel learning algorithm for NFA. libalf is highly flexible and allows for facilely interchanging learning algorithms and combining domain-specific features in a plug-and-play fashion. Its modular design avirtual plantnd its implementation in C++ make it a flexible platform for adding and engineering further, efficient learning algorithms for new target models (e.g., Büchi automata).

Details on libalf can be found at http://libalf.informatik.rwth-aachen.de/

5.1.2. Mole/Cunf: unfolders for Petri Nets

Participants: Stefan Schwoon [correspondent], César Rodríguez.

Mole computes, given a safe Petri net, a finite prefix of its unfolding. It is designed to be compatible with other tools, such as PEP and the Model-Checking Kit, which are using the resulting unfolding for reachability checking and other analyses. The tool Mole arose out of earlier work on Petri nets. Details on Mole can be found at http://www.lsv.ens-cachan.fr/~schwoon/tools/mole/

In the context of MExICO, we have created a new tool called Cunf, which is able to handle contextual nets (Petri nets with read arcs). Recent work carried out within MExICO [53] has transformed a preliminary implementation into an efficient tool. While in principle every contextual net can be transformed into an “equivalent” Petri net and then unfolded using Mole, Cunf can take advantage of their special features to do the job faster. More details can be found at http://www.lsv.ens-cachan.fr/~rodrigue/tools/cunf/

5.1.3. COSMOS: a Statistical Model Checker for the Hybrid Automata Stochastic Logic

Participants: Hilal Djafri [correspondent], Benoît Barbot.

COSMOS is a statistical model checker for the Hybrid Automata Stochastic Logic (HASL). HASL employs Linear Hybrid Automata (LHA), a generalization of Deterministic Timed Automata (DTA), to describe accepting execution paths of a Discrete Event Stochastic Process (DESP), a class of stochastic models which includes, but is not limited to, Markov chains. As a result HASL verification turns out to be a unifying framework where sophisticated temporal reasoning is naturally blended with elaborate reward-based analysis. COSMOS takes as input a DESP (described in terms of a Generalized Stochastic Petri Net), an LHA and an expression Z representing the quantity to be estimated. It returns a confidence interval estimation of Z. COSMOS is written in C++ and is freely available to the research community.

Details on COSMOS can be found at http://www.lsv.ens-cachan.fr/~barbot/cosmos/
5. Software

5.1. F7: Refinement Types for F#

Participants: Karthikeyan Bhargavan [correspondant], Cédric Fournet [MSR Cambridge], Andrew D. Gordon [MSR Cambridge].

F7 is an enhanced typechecker for the F# programming language that enables static checking of properties expressed as refinement types.

A refinement type is a base type qualified with a logical formula; the formula can express invariants, preconditions, and postconditions. F7 relies on type annotations, including refinements, provided in specific interface files. While checking code, F7 generates many logical problems which it solves by submitting to Z3, an external theorem prover for first-order logic (de Moura and Bjørner 2008). Finally, F7 erases all refinements and yields ordinary F# modules and interfaces.

Our main aim is to use F7 for the verification of security-critical programs. We have used it to verify implementations of access control mechanisms, multi-party secure sessions, cryptographic protocols for web services security and federated authentication, and secure audit logs.

A first version of F7 was released in 2008. In 2011, we revised the F7 libraries and typechecker and ported it to the released version of F# for .NET 4.0. The second version of F7 was released in December 2011.

The typechecker is written in 16000 lines of F#, with an additional cryptographic library of 9000 lines, and sample code of more than 12000 lines.

5.2. JSTY: Logical Auditing of JavaScript Programs

Participants: Karthikeyan Bhargavan [correspondant], Sergio Maffeis [Imperial College], Ravinder Shankesi [U. of Illinois at Urbana Champain].

JSTY is a runtime monitoring and logical auditing framework for JavaScript web applications. It has three components: (1) a contract language for JavaScript that enables programmers to annotate their scripts with assumptions and goals written as first-order logic pre- and post-conditions; (2) a runtime monitor implemented as a browser extension in the web browser Chrome that interprets these contracts at runtime and generates proof obligations for an SMT solver; (3) a logical auditor that checks proof obligations and maps counterexamples to violations of program correctness goals.

The target applications for JSTY include browser extensions as well as website scripts. In the case of browser extensions, our goal is to help extension writers to test their code by annotating it with logical contracts and auditing the code with JSTY. For website scripts, our goal is to check whether a website script obeys a generic security policy expressed as pre-conditions on functions in the browser or DOM API. We have used JSTY to analyze a variety of security-critical browser extensions and website scripts and found several vulnerabilities. We are currently incorporating static checking into JSTY.

JSTY is written in about 1000 lines of JavaScript and we plan a public release in 2012.

5.3. OTT: Tool support for the working semanticist

Participants: Peter Sewell [U. of Cambridge], Francesco Zappa Nardelli [correspondant].
Ott is a tool for writing definitions of programming languages and calculi. It takes as input a definition of a language syntax and semantics, in a concise and readable ASCII notation that is close to what one would write in informal mathematics. It generates output:

1. a LaTeX source file that defines commands to build a typeset version of the definition;
2. a Coq version of the definition;
3. an Isabelle version of the definition; and
4. a HOL version of the definition.

Additionally, it can be run as a filter, taking a LaTeX/Coq/Isabelle/HOL source file with embedded (symbolic) terms of the defined language, parsing them and replacing them by typeset terms.

The main goal of the Ott tool is to support work on large programming language definitions, where the scale makes it hard to keep a definition internally consistent, and to keep a tight correspondence between a definition and implementations. We also wish to ease rapid prototyping work with smaller calculi, and to make it easier to exchange definitions and definition fragments between groups. The theorem-prover backends should enable a smooth transition between use of informal and formal mathematics.

In collaboration with Peter Sewell (Cambridge University).

The current version of Ott is about 30000 lines of OCaml. The tool is available from http://moscova.inria.fr/~zappa/software/ott (BSD licence).

Since its release in December 2007, the tool has been used in several projects, including a large proof of type preservation for the OCaml language (without modules) done by Scott Owens.

In 2011, apart from minor bug-fixes and features added, we implemented several performance improvements which result in a up-to 6x speed-up, and kept the Isabelle and Coq backend up-to date with the theorem prover evolution.

The currently released version is 0.21.1.

5.4. Lem, a tool for lightweight executable mathematics

**Participants:** Scott Owens [U. of Cambridge], Peter Sewell [U. of Cambridge], Francesco Zappa Nardelli [correspondant].

Lem is a lightweight tool for writing, managing, and publishing large scale semantic definitions. It is also intended as an intermediate language for generating definitions from domain-specific tools, and for porting definitions between interactive theorem proving systems (such as Coq, HOL4, and Isabelle). As such it is a complementary tool to Ott.

Lem resembles a pure subset of Objective Caml, supporting typical functional programming constructs, including top-level parametric polymorphism, datatypes, records, higher-order functions, and pattern matching. It also supports common logical mechanisms including list and set comprehensions, universal and existential quantifiers, and inductively defined relations. From this, Lem generates OCaml, HOL4 and Isabelle code; the OCaml backend uses a finite set library (and does not yet support inductive relations). A Coq backend is in development.

Lem is already in use at Cambridge and INRIA for research on relaxed-memory concurrency. We are currently preparing a feature-complete release with back-ends for HOL4, Isabelle/HOL, Coq, OCaml, and LaTeX. The project web-page is http://www.cl.cam.ac.uk/~so294/lem/ . A paper on a Lem prototype appeared in ITP 2011, in the “rough diamond” category [ 25 ].

5.5. Memevents-Litmus-Diy-Dont

**Participants:** Jade Alglave, Luc Maranget [correspondant], Susmit Sarkar [U. of Cambridge, UK], Peter Sewell [U. of Cambridge, UK].
Luc Maranget is the main developer of the tools suite of project “Weak Memory Models” (cf. the relevant section).

This suite features three subtools *memevents* (model checker), *litmus* (runs tests on actual machines) and *diy* (generate tests from concise specifications). This year saw a new tool and one official releases (with documentation) [33] — see also [http://diy.inria.fr](http://diy.inria.fr). The releases feature all tools except *memevents*, which we wish to keep for ourselves.

This year main extensions are the handling of the ARM architecture and more collaboration between tools. For the latter, the test generator *diy* enrichs tests with meta-data that are exploited by *litmus*, so as to perform binding of test threads to machine processors, intelligent prefetch of data etc. We plan a new release of the tool suite early next year.

A new, independent, “proof of concept” tool, *offence* was written by J. Alglave and L. Maranget, as a support of our publication [30].

This software is available at [http://diy.inria.fr/offence](http://diy.inria.fr/offence).

### 5.6. Jocaml

**Participants:** Luc Maranget, Xavier Clerc [correspondant].

Jocaml is an implementation of the join-calculus integrated into Ocaml. With respect to previous join-language prototypes, the most salient feature of the new prototype is a better integration into Ocaml. We achieve binary compatibility with Ocaml, moreover Jocaml releases now follow Ocaml releases. See previous year reports for details on Jocaml. The current version is 3.12.1 (released in September [34]) is available at [http://jocaml.inria.fr](http://jocaml.inria.fr).

This new release features an extended Jocaml specific library that provide programmers with an easier access to concurrency and distribution:

1. Some utilities to parse command line, organize client-server connection, etc. This code was written partly by Xavier Clerc, engineer at INRIA SED department.
2. Some new abstractions of text channels help for writing text oriented applications.

### 5.7. Hevea

**Participant:** Luc Maranget [correspondant].

Hevea is a fast translator from full LaTeX to HTML, written in Ocaml. Hevea is highly configurable with commands written in LaTeX. Mathematics are rendered with UNICODE characters for symbols and HTML tables for formatting. Hevea produces HTML 4.0, enriched by css files. Hevea comes with Hacha companion, which produces a set of HTML pages (for instance, one page per chapter). Since it is very efficient and configurable, Hevea is adequate for on-line manuals or teaching courses.

This year saw a few developments around Hevea, mostly for maintenance. Hevea is available at [http://hevea.inria.fr](http://hevea.inria.fr).
PAREO Project-Team

5. Software

5.1. ATerm

Participant: Pierre-Etienne Moreau [correspondant].

ATerm (short for Annotated Term) is an abstract data type designed for the exchange of tree-like data structures between distributed applications.

The ATerm library forms a comprehensive procedural interface which enables creation and manipulation of ATerms in C and Java. The ATerm implementation is based on maximal subterm sharing and automatic garbage collection.

A binary exchange format for the concise representation of ATerms (sharing preserved) allows the fast exchange of ATerms between applications. In a typical application—parse trees which contain considerable redundant information—less than 2 bytes are needed to represent a node in memory, and less than 2 bits are needed to represent it in binary format. The implementation of ATerms scales up to the manipulation of ATerms in the giga-byte range.

The ATerm library provides a comprehensive interface in C and Java to handle the annotated term data-type in an efficient manner.

We are involved (with the CWI) in the implementation of the Java version, as well as in the garbage collector of the C version. The Java version of the ATerm library is used in particular by Tom.

The ATerm library is documented, maintained, and available at the following address: http://www.meta-environment.org/Meta-Environment/ATerms.

5.2. Tom

Participants: Jean-Christophe Bach, Horatiu Cirstea, Pierre-Etienne Moreau [correspondant], Claudia Tavares.

Since 2002, we have developed a new system called Tom [49], presented in [27], [28]. This system consists of a pattern matching compiler which is particularly well-suited for programming various transformations on trees/terms and XML documents. Its design follows our experiences on the efficient compilation of rule-based systems [45]. The main originality of this system is to be language and data-structure independent. This means that the Tom technology can be used in a C, C++ or Java environment. The tool can be seen as a Yacc-like compiler translating patterns into executable pattern matching automata. Similarly to Yacc, when a match is found, the corresponding semantic action (a sequence of instructions written in the chosen underlying language) is triggered and executed. Tom supports sophisticated matching theories such as associative matching with neutral element (also known as list-matching). This kind of matching theory is particularly well-suited to perform list or XML based transformations for example.

In addition to the notion of rule, Tom offers a sophisticated way of controlling their application: a strategy language. Based on a clear semantics, this language allows to define classical traversal strategies such as innermost, outermost, etc.. Moreover, Tom provides an extension of pattern matching, called anti-pattern matching. This corresponds to a natural way to specify complements (i.e. what should not be there to fire a rule). Tom also supports the definition of cyclic graph data-structures, as well as matching algorithm and rewriting rules for term-graphs.

5.3. Cat

**Participant:** Yves Guiraud [correspondant].

Cat is a library for polygraphic calculus, written in Caml. It has been used, in a joint work with F. Blanqui, to produce an automatic termination prover for first-order functional programs. It translates such a rewriting system into a polygraph and tries to find a derivation proving its termination, using the results of [6], [39]. If possible, it seeks a derivation that proves that the program is polynomial [29], [3]. Cat is also at the basis of Catex.

5.4. Catex

**Participant:** Yves Guiraud [correspondant].

Catex is a tool for (pdf)LaTeX, used in the same way as Bibtex, that automatically produces string diagrams from their algebraic expression. It follows the same design as Tom, a Catex file being a LaTeX file enriched with formal islands corresponding to those algebraic expressions, such as:

```latex
\deftwocell[red]{delta : 1 -> 2}
\deftwocell[orange]{mu : 2 -> 1}
\deftwocell[\textbf{crossing}]{\tau : 2 -> 2}
\twocell{(\delta *0 \delta) *1 (1 *0 \tau *0 1) *1 (\mu *0 \mu)}
```

Catex dissolves such an island into LaTeX code, using the PGF/Tikz package. Executed on the result, (pdf)LaTeX produces the following diagram:

![Diagram](image.png)

*Figure 4.*

Catex is distributed through the page: [http://www.loria.fr/~guiraudy/catex](http://www.loria.fr/~guiraudy/catex). We want to extend Catex in two directions. First, to produce diagrams not only for LaTeX but also for web publications. Then, Catex will be adapted to a tool for the automatic production, in scientific papers, of certified algebraic computations, which are a three-dimensional equivalent of string diagrams.
5. Software

5.1. Profound

Participants: Kaustuv Chaudhuri [correspondant], Nicolas Guenot, Lutz Straßburger.

Profound is an interactive proof-development tool based on the focused calculus of structures [19]. It allows the user to build proofs using direct manipulation of the current proof state using the cursor keys and the mouse, instead of learning a formal textual proof interaction language. The tool checks all user actions dynamically with the aid of a theorem prover.

We plan to investigate adaptations of a tool such as Profound for proof development in other interactive proof development systems such as Abella or Coq. We also plan to use the high degree of proof compression that is enabled by the calculus of structures to create efficient proof certificates for exchange between different proof development systems.

The first public release of Profound is expected in December 2011. The development can be followed on INRIA GForge.

5.2. Abella

Participants: Andrew Gacek, Dale Miller.

The earliest versions of the Abella theorem prover was written while Gacek was a PhD student at the University of Minnesota. Two years ago, Gacek was a post doc in the Parsifal team and more features were added to this prover. During 2011, Chaudhuri and three interns (Andrew Cave from McGill and Salil Joshi and Chris Martens from CMU) developed some news designs and new prototypes of needed features for Abella. These features will provide this prover with better ways to manipulate specification-logic contexts in the reasoning-logic and a means for outputting proper proof object (different from the scripts that are used to find a proof).

For more information, see the Abella home page: http://abella.cs.umn.edu/.

5.3. Bedwyr

Participants: Kaustuv Chaudhuri, Quentin Heath, Dale Miller [correspondant].

During 2011, our close colleagues Alwen Tui (Australian National University) and David Baelde (INRIA team Proval) have made some improvements to the Bedwyr system. In the case of Tui, he made these changes in order to build SPEC, a model checker for the spi-calculus, on top of Bedwyr.

Starting in September, Quentin Heath has joined the team as a technical staff member. He is currently working on the Bedwyr code so that it can share files with the Abella system. These two provers work within a logic that is essentially the same: Heath is working to ensure that the concrete syntax and static semantics for the logical expressions on which they work is also the same. Thus, we expect to have our model checker (Bedwyr) and interactive theorem prover (Abella) share theories and proofs.

The work of Heath is being done in the context of the BATT ADJ project funded by INRIA. The boarder goals of the BATT project is to get four software systems (Bedwyr, Abella, Tac, and Teyjus) to inter-operate.

See also the web page http://slimmer.gforge.inria.fr/bedwyr/.
5. Software


Participants: Bruno Barras [TypiCal team, Saclay], Yves Bertot [Marelle team, Sophia], Frédéric Besson [Lande team, Rennes], Pierre Bouillier, Xavier Clerc [SED team], Pierre Corbineau [University Joseph Fourier, Grenoble], Pierre Courtieu [CNAM], Julien Forest [CNAM], Stéphane Glondu, Benjamin Grégoire [Marelle team, Sophia], Vincent Gross, Hugo Herbelin [correspondant], Stéphane Lescuyer [ProVal team, Saclay], Pierre Letouzey, Assia Mahboubi [TypiCal team, Saclay], Julien Narboux [University of Strasbourg], Jean-Marc Notin [TypiCal team, Saclay], Christine Paulin [Proval team, Saclay], Loïc Pottier [Marelle team, Sophia], Matthias Puech, Yann Régis-Gianas, Vincent Siles, Elie Soubiran, Matthieu Sozeau, Arnaud Spiwack, Pierre-Yves Strub [Formes team, Beijing], Laurent Théry [Marelle team, Sophia], Benjamin Werner [TypiCal team, Saclay].

5.1.1. Version 8.4

Version 8.4 beta was released in December 2011. It introduces a new proof engine designed and implemented by Arnaud Spiwack and a new extensive modular library of arithmetic contributed by Pierre Letouzey. It also includes an extension of the underlying logic with \( \eta \)-conversion by Hugo Herbelin and “commutative-cuts compliant guard condition” by Pierre Bouillier, an extension of the pattern-matching compilation algorithm by Hugo Herbelin, an extension of the procedure of simplification of polynomial expressions by Loïc Pottier, a refinement of the type classes mechanism by Matthieu Sozeau, a new communication model by Vincent Gross for the graphical user interface CoqIDE, that Pierre Letouzey, Pierre Bouillier and Pierre-Marie Pédrot further extended.

Several users gracefully contributed improvements of various features (Tom Prince, Enrico Tassi, Daniel Grayson, Hendrik Tews, ...).

5.1.2. Graphical user interface

Pierre Letouzey has finalized the work initiated by Vincent Gross (former ADT engineer) concerning the CoqIDE user interface: CoqIDE and Coq are now separate unix processes, enhancing the reliability and improving the user experience.

5.1.3. Type inference, tactics, unification and type classes

Matthieu Sozeau corrected important issues with the unification algorithm and enhanced it to support universes. He improved the type-class implementation, adding support for forward-reasoning instances.

To improve the power of induction tactics, Hugo Herbelin added new heuristics for second-order pattern-matching based on ideas from Chung-Kil Hur’s Ieq plugin.

Pierre Letouzey extended the pattern-matching feature of the tactic language.

5.1.4. Internal architecture of the Coq software

Pierre Letouzey also initiated a large reorganization of the internal components of Coq, since these components are currently too much interdependent. This work aims at better isolating components and explicating the interfaces between them. In addition to the initial goal of simplifying the compilation of Coq, having a clearer architecture is also expected to help new contributors when they discover and interact with this large and complex code-base. It also brings new prospects in direct communications between tools developed around Coq. This is a long-term effort that extends beyond the Coq 8.4 release.
Pierre Boutillier worked on the build system generator for Coq users contributions. It now handles correctly developments involving ML files. Files to build developments are also used by CoqIDE to infer the required arguments when it opens a file of a development.

### 5.1.5. Efficiency

Pierre Letouzey has pursued his effort concerning the improvement of many aspects of the internals of Coq. In particular, with Yann Régis-Gianas, he enabled a faster load of libraries by default, thanks to laziness, and also a better sharing of structures in memory (via better hash-consing), with lower memory footprint and some speedup as visible result. Many bugs have also been addressed.

Starting from September, Xavier Clerc has worked on the codebase in order to profile typical executions. Some hotspots were identified, most notably in comparison functions: some minor modifications led to a gain of a few percents on average. Some tests led to envision the use of a Coq-specialized version of comparison functions, superseding the generic OCaml ones.

### 5.1.6. General maintenance

Hugo Herbelin, Pierre Letouzey, Pierre Boutillier, Stéphane Glondu and Matthieu Sozeau worked on the general maintenance of the system.

### 5.1.7. Development Action

A new “Action de Développement Technologique” about Coq has started September 2011. It gathers the πr² team, the Marelle team and the CPR team from CNAM, Hugo Herbelin acting as the coordinator. It supports visits and meetings between developers and aims at strengthening the community of Coq users and contributors.

### 5.1.8. Formalisation in Coq

Stéphane Glondu is working with Mehdi Dogguy on the formalisation in Coq of a type system for a timed asynchronous π-calculus that guarantees confluence.

### 5.2. Pangolin

**Participant:** Yann Régis-Gianas.

Yann Régis-Gianas maintained a prototype version of Pangolin. He used it to prove concrete complexity bounds for a set of functional programs using the method described in his FOPARA 2011 paper \[16\].

### 5.3. Other software developments

Stéphane Glondu is involved in the maintenance of OCaml-related packages in Debian, which include OCaml itself, Coq, Ssreflect (an extension of Coq developed at INRIA-MSR joint center) and Ocsigen (a web framework developed at PPS). The Ubuntu distribution naturally benefits from this work.

In collaboration with François Pottier (INRIA Gallium), Yann Régis-Gianas maintained Menhir, an LR parser generator for OCaml.
PROVAL Project-Team

5. Software

5.1. The CiME rewrite toolbox

Participants: Évelyne Contejean [contact], Claude Marché, Andrei Paskevich, Xavier Urbain.

CiME is a rewriting toolbox. Distributed since 1996 as open source, at URL http://cime.lri.fr. Beyond a few dozens of users, CiME is used as back-end for other tools such as the TALP tool developed by Enno Ohlebusch at Bielefeld university for termination of logic programs; the MU-TERM tool (http://www.dsic.upv.es/~slucas/csr/termination/muterm/) for termination of context-sensitive rewriting; the CARIBOO tool (developed at INRIA Nancy Grand-Est) for termination of rewriting under strategies; and the MTT tool (http://www.lcc.uma.es/~duran/MITT/) for termination of Maude programs. CiME2 is no longer maintained, and the currently developed version is CiME3, available at http://a3pat.ensiie.fr/pub. The main new feature of CiME3 is the production of traces for Coq. CiME3 is also developed by the participants of the A3PAT project at the CNAM, and is distributed under the Cecill-C license.

5.2. The Why platform

Participants: Claude Marché [contact], Romain Bardou, François Bobot, Jean-Christophe Filliâtre, Guillaume Melquiond, Andrei Paskevich.


The Why platform is a set of tools for deductive verification of Java and C source code. In both cases, the requirements are specified as annotations in the source, in a special style of comments. For Java (and Java Card), these specifications are given in JML and are interpreted by the Krakatoa tool. Analysis of C code must be done using the external Frama-C environment, and its Jessie plugin which is distributed in Why. The platform is distributed as open source, under GPL license, at http://why.lri.fr/. The internal VC generator and the translators to external provers are no longer under active development, as superseded by the Why3 system described below.

The Krakatoa and Jessie front-ends are still maintained, although using now by default the Why3 VC generator. These front-ends are described in a specific web page http://krakatoa.lri.fr/. They are used for teaching (University of Evry, Ecole Polytechnique, etc.), used by several research groups in the world, e.g at Fraunhofer Institute in Berlin [71], and at Universidade do Minho in Portugal [50].

5.3. The Why3 system

Participants: Jean-Christophe Filliâtre [contact], François Bobot, Claude Marché, Guillaume Melquiond, Andrei Paskevich.


Why3 is the next generation of Why. Why3 clearly separates the purely logical specification part from generation of verification conditions for programs. It features a rich library of proof task transformations that can be chained to produce a suitable input for a large set of theorem provers, including SMT solvers, TPTP provers, as well as interactive proof assistants.

It is distributed as open source, under GPL license, at http://why3.lri.fr/.

Why3 is used as back-end of our own tools *Krakatoa* and *Jessie*, but also as back-end of the GNATprove tool (Adacore company), and in a near future of the WP plugin of Frama-C. Why3 has been used to develop and prove a significant part of the programs of our team gallery http://proval.lri.fr/gallery/index.en.html, and will be used soon for teaching (Master Parisien de Recherche en Informatique).

5.4. The Alt-Ergo theorem prover

**Participants**: Sylvain Conchon [contact], Évelyne Contejean, Stéphane Lescuyer, Alain Mebsout, Mohamed Iguernelala.

Criteria for Software Self-Assessment: A-3-up, SO-4, SM-4-up, EM-4, SDL-5, OC-4.

*Alt-Ergo* is an automatic, little engine of proof dedicated to program verification, whose development started in 2006. It is fully integrated in the program verification tool chain developed in our team. It solves goals that are directly written in the *Why*’s annotation language; this means that *Alt-Ergo* fully supports first order polymorphic logic with quantifiers. *Alt-Ergo* also supports the standard [90] defined by the SMT-lib initiative. It is currently used in our team to prove correctness of C and Java programs as part of the *Why* platform and the new *Why*3 system. *Alt-Ergo* is also called as an external prover by the Pangolin tool developed by Y. Regis Gianas, INRIA project-team Gallium http://code.google.com/p/pangolin-programming-language/. *Alt-Ergo* is usable as a back-end prover in the SPARK verifier for ADA programs, since Oct 2010. It is planed to be integrated in next generation of Airbus development process.

*Alt-Ergo* is distributed as open source, under the CeCILL-C license, at URL http://alt-ergo.lri.fr/.

5.5. Bibtex2html

**Participants**: Jean-Christophe Filliâtre [contact], Claude Marché.


Bibtex2html is a generator of HTML pages of bibliographic references. Distributed as open source since 1997, under the GPL license, at http://www.lri.fr/~filliatr/bibtex2html/. We estimate that between 10000 and 100000 web pages have been generated using Bibtex2html.

Bibtex2html is also distributed as a package in most Linux distributions. Package popularity contests show that it is among the 20% most often installed packages.

5.6. OCamlgraph

**Participants**: Jean-Christophe Filliâtre [contact], Sylvain Conchon.

OCamlgraph is a graph library for Objective Caml. It features many graph data structures, together with many graph algorithms. Data structures and algorithms are provided independently of each other, thanks to OCaml module system. OCamlgraph is distributed as open source, under the LGPL license, at http://ocamlgraph.lri.fr/. It is also distributed as a package in several Linux distributions. OCamlgraph is now widely spread among the community of OCaml developers.

5.7. Mlpost

**Participants**: Jean-Christophe Filliâtre [contact], Stéphane Lescuyer, Romain Bardou, François Bobot.

Mlpost is a tool to draw scientific figures to be integrated in LaTeX documents. Contrary to other tools such as TikZ or MetaPost, it does not introduce a new programming language; it is instead designed as a library of an existing programming language, namely Objective Caml. Yet it is based on MetaPost internally and thus provides high-quality PostScript figures and powerful features such as intersection points or clipping. Mlpost is distributed as open source, under the LGPL license, at http://mlpost.lri.fr/. Mlpost was presented at JFLA’09 [51].
5.8. Functory  

**Participants:** Jean-Christophe Filliâtre [contact], Kalyan Krishnamani.

Functory is a distributed computing library for Objective Caml. The main features of this library include (1) a polymorphic API, (2) several implementations to adapt to different deployment scenarios such as sequential, multi-core or network, and (3) a reliable fault-tolerance mechanism. Functory was presented at JFLA 2011 [31] and at TFP 2011 [27].

5.9. The Flocq library  

**Participants:** Sylvie Boldo [contact], Guillaume Melquiond.


The Flocq library for the Coq proof assistant is a comprehensive formalization of floating-point arithmetic: core definitions, axiomatic and computational rounding operations, high-level properties [23]. It provides a framework for developers to formally certify numerical applications.

It is distributed as open source, under a LGPL license, at http://flocq.gforge.inria.fr/. It was first released in 2010.

5.10. The Gappa tool  

**Participant:** Guillaume Melquiond [contact].


Given a logical property involving interval enclosures of mathematical expressions, Gappa tries to verify this property and generates a formal proof of its validity. This formal proof can be machine-checked by an independent tool like the Coq proof-checker, so as to reach a high level of confidence in the certification [66] [19].

Since these mathematical expressions can contain rounding operators in addition to usual arithmetic operators, Gappa is especially well suited to prove properties that arise when certifying a numerical application, be it floating-point or fixed-point. Gappa makes it easy to compute ranges of variables and bounds on absolute or relative roundoff errors.

Gappa is being used to certify parts of the mathematical libraries of several projects, including CRlibm, FLIP, and CGAL. It is distributed as open source, under a Cecill-B/GPL dual-license, at http://gappa.gforge.inria.fr/. Part of the work on this tool was done while in the Arénaire team (INRIA Rhône-Alpes), until 2008.

5.11. The Interval package for Coq  

**Participant:** Guillaume Melquiond [contact].


The Interval package provides several tactics for helping a Coq user to prove theorems on enclosures of real-valued expressions. The proofs are performed by an interval kernel which relies on a computable formalization of floating-point arithmetic in Coq.

It is distributed as open source, under a LGPL license, at http://www.lri.fr/~melquion/soft/coq-interval/. Part of the work on this tool was done while in the Mathematical Components team (Microsoft Research–INRIA Joint Research Center).

In 2010, the Flocq library was used to straighten and fill the floating-point proofs of the Interval package.

5.12. The Alea library for randomized algorithms  

**Participants:** Christine Paulin-Mohring [contact], David Baelde.

The ALEA library is a Coq development for modeling randomized functional programs as distributions using a monadic transformation. It contains an axiomatisation of the real interval $[0, 1]$ and its extension to positive real numbers. It introduces definition of distributions and general rules for approximating the probability that a program satisfies a given property.

It is distributed as open source, at http://www.lri.fr/~paulin/ALEA . It is currently used as a basis of the Certicrypt environment (MSR-INRIA joint research center, Imdea Madrid, INRIA Sophia-Antipolis) for formal proofs for computational cryptography [53]. It is also experimented in LABRI as a basis to study formal proofs of probabilistic distributed algorithms.

5.13. The Coccinelle library for term rewriting

Participant: Évelyne Contejean [contact].

Coccinelle is a Coq library for term rewriting. Besides the usual definitions and theorems of term algebras, term rewriting and term ordering, it also models some of the algorithms implemented in the CiME toolbox, such as matching, matching modulo associativity-commutativity, computation of the one-step reducts of a term, RPO comparison between two terms, etc. The RPO algorithm can effectively be run inside Coq, and is used in the Color development (http://color.inria.fr/) as well as for certifying Spike implicit induction theorems in Coq (Sorin Stratulat).

Coccinelle is developed by Évelyne Contejean, available at (http://www.lri.fr/~contejea/Coccinelle), and is distributed under the Cecill-C license.
5. Software

5.1. Tookan

Participants: Graham Steel [correspondant], Romain Bardou.

See also the web page http://secgroup.ext.dsi.unive.it/projects/security-apis/pkcs11-security/tookan/.

Tookan is a security analysis tool for cryptographic devices such as smartcards, security tokens and Hardware Security Modules that support the most widely-used industry standard interface, RSA PKCS#11. Each device implements PKCS#11 in a slightly different way since the standard is quite open, but finding a subset of the standard that results in a secure device, i.e. one where cryptographic keys cannot be revealed in clear, is actually rather tricky. Tookan analyses a device by first reverse engineering the exact implementation of PKCS#11 in use, then building a logical model of this implementation for a model checker, calling a model checker to search for attacks, and in the case where an attack is found, executing it directly on the device. Tookan has been used to find at least a dozen previously unknown flaws in commercially available devices.

The first results using Tookan were published in 2010 [56] and a six-month licence was granted to Boeing to use the tool. In 2011, this transfer activity has continued, principally in combination with a major UK bank. In June, Tookan was used by Steel and Focardi two days of testing on devices belonging to the bank. Following these results, in September, a more significant contract was signed granting the bank 18 months of use of Tookan to test all their in-house equipment. Initial feedback has been very positive.

Tookan is the subject of a CSATT transfer action resulting in the hiring of an engineer, Romain Bardou, who started on September 1st. Early progress in re-implementing key parts of Tookan to improve modularity and overall code quality has been excellent. The next steps for Tookan are still being investigated: the Tookan project is the subject of a ‘qualification’ procedure by IT2 who will evaluate its suitability as the basis for a start-up company. At the same time other options are being considered, such as partnership with an existing SME. A decision is expected in mid-2012.

5.2. Orchids

Participants: Jean Goubault-Larrecq [correspondant], Hedi Benzina, Baptiste Gourdin, Nasr-Eddine Yousfi.

The ORCHIDS real-time intrusion detection system was created in 2003-04 at SECSI. After a few years where research and development around ORCHIDS was relatively quiet, several new things happened, starting from the end of 2010.

First, several companies and institutions expressed interest in ORCHIDS, among which, notably, EADS Cassidian, Thalès, Galois Inc. (USA), the French Direction Générale de l’Armement (DGA).

Second, Baptiste Gourdin was hired as a development engineer (Dec. 2010-Nov. 2011) on an Action de Développement Technologique (ADT). He improved Orchids in several ways. Its user interface benefitted from a complete revamping. New features were implemented, such as conformance with the IODEF and IDMEF standards, connection with vulnerability and network topology databases, the possibility to do forensics that synchronize past events to the state that the above databases were in at the time of the events, among others.

Nasr-Eddine Yousfi has followed up on Baptiste Gourdin, starting from December 2011, on an ITI engineer position allotted by INRIA’s CSATT.

Hedi Benzina implemented a tool on top of ORCHIDS, RuleGen, which allows one to write simple security policies that compile to ORCHIDS rules.

The efforts done in 2011 around ORCHIDS should be seen as the first steps in the creation of an open source consortium, which will be consolidated in the next years.
5.3. AKISS and SubVariant

Participant: Ștefan Ciobăcă.

AKISS (http://www.lsv.ens-cachan.fr/~ciobaca/akiss/) is a tool implementing a procedure for verifying trace equivalence (or equivalently may-testing equivalence) for bounded security processes with no else branches employing cryptographic primitives modeled by an optimally reducing rewrite system.

Trace equivalence can be used to model strong secrecy, vote-privacy and other security properties.

AKISS uses a fully-abstract encoding of symbolic traces into Horn clauses, thereby extending the KISS tool (http://www.lsv.ens-cachan.fr/~ciobaca/kiss/), which can only check static equivalence.

In order to get rid of the equational theory modeling the cryptographic primitives, AKISS employs algorithms for computing strongly complete sets of variants and complete set of unifiers of the SubVariant tool. AKISS is described in an article submitted to ESOP, in Chapter 5 of Ștefan Ciobăcă’s PhD thesis [12].

SubVariant (http://www.lsv.ens-cachan.fr/~ciobaca/subvariant/) is a tool for computing finite strongly complete set of variants modulo a convergent optimally reducing term rewriting system. SubVariant can also compute complete sets of equational unifiers for equational theories implemented by a convergent optimally reducing term rewriting system.

Complete sets of variants and the finite variant property were introduced in [59]. In [33], Ștefan Ciobăcă defines strongly complete sets of variants, which are more natural and more useful. Chapter 3 in Ștefan Ciobăcă’s PhD thesis describes extensively the algorithms behind SubVariant.
TASC Project-Team

5. Software

5.1. CHOCO

Participants: Nicolas Beldiceanu, Alexis De Clerq, Sophie Demassey, Jean-Guillaume Fages, Narendra Jussien [correspondant], Arnaud Letort, Xavier Lorca [correspondant], Arnaud Malapert, Julien Menana, Thierry Petit, Charles Prud’homme [correspondant].

CHOCO is a Java discrete constraints library integrating within a same system explanations, soft constraints and global constraints (90000 lines of source code). This year developments were focussing on the following aspects:

1. Providing a complete solver independent specification of explanation algorithms, data structure for encoding «nogoods» and treatment algorithms. A reference implementation is being made within the new version of our solver CHOCO.
2. Design and development of a dedicated languages to specify the propagation and the search heuristics of constraint solvers.
3. Providing efficient implementation of filtering algorithms for constraints such as tree, increasing_sum, cumulative with resource overload.
4. Providing an implementation of a probabilistic model for alldifferent.


5.2. IBEX

Participants: Gilles Chabert [correspondant], Rémi Douence.

IBEX (Interval-Based EXplorer) is a C++ library for solving nonlinear constraints over real numbers (25000 lines of source code). The main feature of Ibex is its ability to build solver/paver strategies declaratively through the contractor programming paradigm. Ibex includes a parser of the QUIMPER language (QUick Interval Modeling and Programming in a bounded-ERror context) and is currently used in several academic research labs.

G. Chabert and R. Douence (ASCOLA) have contributed in 2011 to the ongoing redesign of the architecture IBEX, the goal being to make it more flexible to cope with specific problems, and more easy to use. The link to the system and documentation is http://choco.emn.fr.

5.3. Global Constraint Catalog

Participants: Nicolas Beldiceanu [correspondant], Sophie Demassey, Mats Carlsson, Helmut Simonis.

The global constraint catalog presents and classifies global constraints and describes different aspects with meta data. It consist of

1. a pdf version that can be downloaded from http://www.emn.fr/z-info/sdemasse/gccat/ (at item last working version) containing 360 constraints, 3000 pages and 700 figures,
2. an on line version accessible from the previous address,
3. meta data describing the constraints (buton PL for each constraint, e.g., alldifferent.pl),
4. an online service (i.e., a constraint seeker) which provides a web interface to search for global constraints, given positive and negative ground examples.
This year developments were focussing on:

1. maintaining the catalogue,
2. deploying an on-line constraint seeker [16] (see http://seeker.mines-nantes.fr/ and http://4c.ucc.ie/~hsimonis/seekerhelp.html for explanation how to use),
3. providing the negation for constraints defined by automata (with and without counter),
4. defining properties of constraints arguments, and
5. providing modelling examples as well as points of interests and common misunderstanding for core constraints.

N. Beldiceanu, S. Demassey, M. Carlsson (SICS, Sweden) and H. Simonis (4C, Ireland) have contributed in 2011. The link to the global constraint catalog is http://www.emn.fr/z-info/sdemasse/gccat/.
TYPICAL Project-Team

5. Software

5.1. Coq

Participants: Bruno Barras [Contact], Jean-Marc Notin, Arnaud Spiwack, Enrico Tassi.

Coq is a major proof system and the primary object and/or tool of our research. Its development is now mainly coordinated by the π² INRIA Paris-Rocquencourt project-team, and some members of the TYPICAL team are active developers of the system.

5.2. Coqfinitegroup

Participants: Cyril Cohen, Assia Mahboubi [Contact].

Coqfinitegroup is the development corresponding to the ongoing effort to formalize the proof of the Feit-Thompson theorem. It is probably the most advanced formal development of group theory today. Its current size is about 80,000 lines of (compact) Coq code. Assia Mahboubi and Cyril Cohen are actively participating to this long term formalization project.

5.3. Dedukti

Participants: Mathieu Boespflug [Contact], Gilles Dowek.

Dedukti is a universal proof checker, based on the $\lambda\pi$-calculus modulo formalism. Mainly developed by Mathieu Boespflug, it is distributed under the GNU licence. The main system includes about 2000 lines of Haskell.

5.4. Ssreflect

Participants: Assia Mahboubi [Contact], Enrico Tassi.

SSReflect is a proof language extension of Coq developed under Georges Gonthier (Microsoft Research). It was originally designed to make the formalization of the Four Color Theorem possible and has been evolving since. It is important to note that it is shipped with redesigned basic proof libraries. Members of the Typical are in charge of the documentation and distribution of this extension.
5. Software

5.1. The veriT solver

**Participants:** Diego Caminha Barbosa de Oliveira, David Déharbe, Pascal Fontaine [correspondant], Bruno Woltzenlogel Paleo.

The veriT solver is an SMT (Satisfiability Modulo Theories) solver developed in cooperation with David Déharbe from the Federal University of Rio Grande do Norte in Natal, Brazil. The solver can handle large quantifier-free formulas containing uninterpreted predicates and functions, and arithmetic on integers and reals. It features a very efficient decision procedure for difference logic, as well as a simplex-based reasoner for full linear arithmetic. It also has some support for user-defined theories, quantifiers, and lambda-expressions. This allows users to easily express properties about concepts involving sets, relations, etc. The prover can produce an explicit proof trace when it is used as a decision procedure for quantifier-free formulas with uninterpreted symbols and arithmetic. To support the development of the tool, a regression platform using INRIA’s grid infrastructure is used; it allows us to extensively test the solver on thousands of benchmarks in a few minutes.

The veriT solver is available as open source under the BSD license, and distributed through the web site [http://www.veriT-solver.org](http://www.veriT-solver.org). It entered for the third time the international competition of SMT solvers SMT-COMP 2011, a joint event with the SMT workshop 2011 and the CAV conference. As in the previous competitions, it performed decently against the other participating SMT solvers. It embeds an original symmetry reduction technique that greatly improved its efficiency on some categories of formulas. This technique was immediately incorporated also in other competing solvers, in particular Z3 (Microsoft) and CVC3 (University of New-York and University of Iowa).

Efforts in 2011 have been focused on the extension of the expressiveness of the tool (with improvements in the handling of quantifiers), and on its efficiency (which was significantly improved at different levels, including a purpose-built SAT solver underlying veriT). A lot of work was also devoted to improve the proof production of the tool, with the definition of a precise proof language. This proof language has been presented to the community as a standard for describing SMT proofs [17]. We are collaborating on this with Laurent Théry and Benjamin Grégoire (Marelle, INRIA Sophia-Antipolis), Laurent Voisin (Systerel), and Frédéric Besson (Celtique, INRIA Rennes).

Future research and implementation efforts will be directed to furthermore extend the accepted language, and increase the efficiency. We target applications where validation of formulas is crucial, such as the validation of TLA\(^+\) and B specifications, and work together with the developers of the respective verification platforms to make veriT even more useful in practice.

The software will be supported by an INRIA ADT, which will start at the beginning of 2012.

5.2. The TLA+ proof system

**Participants:** Stephan Merz, Hernán-Pablo Vanzetto.

TLAPS, the TLA\(^+\) proof system, is a platform for developing and mechanically verifying TLA\(^+\) proofs. It is developed at the Joint MSR-INRIA Centre. The TLA\(^+\) proof language is declarative and based on standard mathematical logic; it supports hierarchical and non-linear proof construction and verification. TLAPS consists of a proof manager that interprets the proof language and generates a collection of proof obligations that are sent to backend verifiers that include theorem provers, proof assistants, SMT solvers, and decision procedures.
TLAPS is publicly available at http://msr-inria.inria.fr/~doligez/tlaps/, it is distributed under a BSD-like license. It handles the non-temporal part of TLA+ with the exception of computing enabledness predicates and can currently be used to prove safety, but not liveness properties. Its backends include a tableau prover for first-order logic, an encoding of TLA+ in the proof assistant Isabelle, as well as an SMT translation and a custom decision procedure for Presburger arithmetic. Our main contribution in 2011 has been the implementation of a new SMT backend that handles formulas including linear arithmetic, elementary set theory, functions, tuples, and records (see section 6.4). Other efforts in 2011 concerned improvements and stabilization of the fingerprinting technique that avoids reproving proof obligations that have remained unchanged since a previous prover run.