RESEARCH CENTER

FIELD
Algorithmics, Programming, Software and Architecture

Activity Report 2012

Section Software

Edition: 2013-04-24

ARIC Team

5. Software

5.1. Overview

AriC software and hardware realizations are accessible from the web page http://www.ens-lyon.fr/LIP/AriC/ware.html. We describe below only those which progressed in 2012.

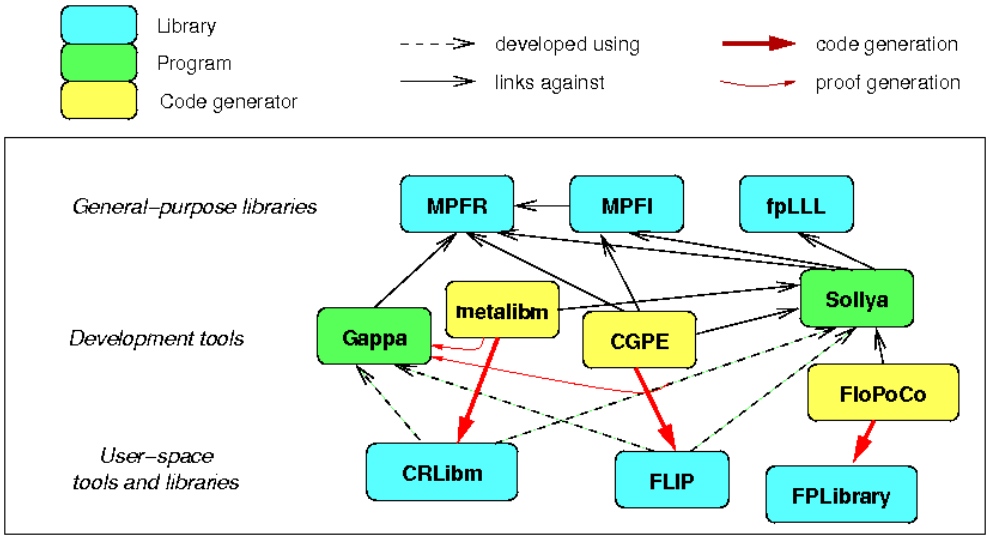

Figure 1. Relationships between some AriC developments.

5.2. FloPoCo

Participants: Florent de Dinechin [correspondant], Matei Istoan.

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 2012, the diverging multiplier implementations in FloPoCo were unified using a common *bit-heap* framework. In addition, several new operators were added.

FloPoCo also now offers state-of-the-art random generators written by David Thomas at Imperial College.

Versions 2.3.1 and 2.4.0 were released in 2012.

Among the known users of FloPoCo are U. Cape Town, U.T. Cluj-Napoca, Imperial College, U. Essex, U. Madrid, U. P. Milano, T.U. Muenchen, T. U. Kaiserslautern, U. Paderborn, CalTech, U. Pernambuco, U. Perpignan, U. Tokyo, Virginia Tech U. and several companies.

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

- Version: 2.3.0 (december 2011)
- APP: IDDN.FR.001.400014.000.S.C.2010.000.20600 (version 2.0.0)
- License: specific, GPL-like.
- Type of human computer interaction: command-line interface, synthesizable VHDL output.
- OS/Middleware: Linux, Windows/Cygwin.
- Required library or software: MPFR, flex, Sollya.
- Programming language: C++.
- Documentation: online and command-line help, API in doxygen format, articles.

5.3. GNU MPFR

Participants: Vincent Lefèvre [correspondant], Paul Zimmermann [Caramel, Inria Nancy - Grand Est].

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 AriC (formerly Arénaire) project-teams. MPFR has been a GNU package since 26 January 2009.

An MPFR-MPC developers meeting took place from 25 to 27 June 2012 in Bordeaux. GNU MPFR 3.1.1 was released on 3 July 2012.

The main changes done in the AriC project-team for the future versions are tcc support, more automation for the releases, new functions to operate on groups of flags, and bug fixes.

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

GNU MPFR is now on the Ohloh community platform for free and open source software: https://www.ohloh.net/p/gnu-mpfr

- 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).

The Perl scripts have been improved to detect various errors from Maple and in particular, restart Maple automatically when the license server is not reachable.

5.5. FLIP: Floating-point Library for Integer Processors

Participants: Claude-Pierre Jeannerod [correspondant], 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 extended the DP2 operator (fused dot product in dimension two) and its specializations, initially designed for rounding to nearest, to directed rounding modes. We have also worked on the implementation of the simultaneous computation of sine and cosine, with proven 1-ulp accuracy and in the same latency as the evaluation of sine alone.

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

5.6. FPLLL: A Lattice Reduction Library

Participants: Xavier Pujol, Damien Stehlé [correspondant].

fplll contains several algorithms on lattices that rely on floating-point computations. This includes implementations of the floating-point LLL reduction algorithm, offering different speed/guarantees ratios. It contains a "wrapper" choosing the estimated best sequence of variants in order to provide a guaranteed output as fast as possible. In the case of the wrapper, the succession of variants is oblivious to the user. It also includes a rigorous floating-point implementation of the Kannan-Fincke-Pohst algorithm that finds a shortest non-zero lattice vector, and the BKZ reduction algorithm.

The fplll library is used or has been adapted to be integrated within several mathematical computation systems such as Magma, Sage and PariGP. It is also used for cryptanalytic purposes, to test the resistance of cryptographic primitives.

Versions 4.0.0 and 4.0.1 were released in 2012, implementing the BKZ reduction algorithm.

URL: http://xpujol.net/fplll/

- ACM: D.2.2 (Software libraries), G.4 (Mathematical software)
- APP: Procedure started
- License: LGPL v2.1
- 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: MPFR and GMP.
- Programming language: C++.
- Documentation: available in html format on **URL:**http://xpujol.net/fplll/fplll-doc.html

5.7. Symbolic-numeric Computations with Linear ODEs

Participant: Marc Mezzarobba.

NumGfun is a Maple package for performing numerical and "analytic" computations with the solutions of linear ordinary differential equations with polynomial coefficients. Its main features include the numerical evaluation of these functions with rigorous error bounds and the computation of symbolic bounds on solutions of certain recurrences. NumGfun is distributed as part of gfun, itself part of the Algolib bundle. It is used by the Dynamic Dictionary of Mathematical Functions to provide its numerical evaluation features. NumGfun 0.6, released in 2012, provides new feature for the numerical solution of so-called regular singular connection problems, and many small improvements.

URL: http://marc.mezzarobba.net/#code-NumGfun

- ACM: D.2.2 (Software libraries), G.4 (Mathematical software)
- APP: cf. gfun
- License: LGPL v2.1
- Type of human computer interaction: Maple library, usable interactively or from Maple code.
- OS/Middleware: any platform supporting Maple.
- Required library or software: Maple, gfun.
- Programming language: Maple
- Documentation: available as Maple help pages and in pdf format.

5.8. SIPE: Small Integer Plus Exponent

Participant: Vincent Lefèvre.

SIPE (Small Integer Plus Exponent) is a mini-library in the form of a C header file, to perform computations in very low precisions with correct rounding to nearest in radix 2. The goal of such a tool is to do proofs of algorithms/properties or computations of tight error bounds in these precisions by exhaustive tests, in order to try to generalize them to higher precisions. The currently supported operations are the addition, subtraction, multiplication, FMA, minimum/maximum/comparison functions (of the signed numbers or in magnitude), and conversions.

A new macro `SIPE_2MUL`, returning the rounded result and the error of a multiplication, has been added.

A test program and scripts to perform timing comparisons with hardware IEEE-754 floating-point and with GNU MPFR are available, together with a discussion on the technical and algorithmic choices behind SIPE and timing results. [39]

- 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 Inria RR-7832.
- 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 ARÉNAIRE project-team (now AriC, 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 Joldeş 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 development in 2012 is the release of version 3.1.1 (the "canard à l'orange" release) in July. With respect to version 3.1.0, this new version improves the reference manual, and fixes a few bugs. Also, a workshop was organized in June in Bordeaux, on the development of GNU MPFR and GNU MPC. In particular, the test coverage of GNU MPFR was improved.

5.3. GNU MPC

Participant: Paul Zimmermann [contact].

GNU 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 GNU MPC library provides correct rounding on both the real part x and the imaginary part y of any result. GNU 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 and Sage computational number theory systems.

A new version, GNU MPC 1.0 (Fagus silvatica), was released in July 2012. Up from this release, GNU MPC is considered to be a mature library. Due to a security issue in automake, we had to release a bug-fix version 1.0.1 in September 2012.

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.

In January 2012, a new version 6.4 was released, followed by 6.4.1 and 6.4.2 in March, and 6.4.3 in June. Apart from bug fixes, and the fact that GMP-ECM is now distributed under the LGPL version 3, those new releases provide a new *-batch* option with faster Stage 1 code, and an improved tuning mechanism.

In February, Paul Leyland found a 43-digit factor using the GPU implementation of Stage 1 written by C. Bouvier, and in August, a new record prime of 79 digits was found by Sam Wagstaff (Purdue University) using GMP-ECM.

5.5. Finite fields

Participants: Pierrick Gaudry, Emmanuel Thomé [contact].

$\texttt{mp}\mathbb{F}_q$ is (yet another) library for computing in finite fields. The purpose of $\texttt{mp}\mathbb{F}_q$ 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*. $\texttt{mp}\mathbb{F}_q$ 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, $\texttt{mp}\mathbb{F}_q$ somehow focuses on prime fields and on fields of characteristic two.

$\texttt{mp}\mathbb{F}_q$'s ability to generate specialized code for desired finite fields differentiates this library from its competitors. The performance achieved is far superior. For example, $\texttt{mp}\mathbb{F}_q$ 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 [1]. $\texttt{mp}\mathbb{F}_q$ 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 [18].

The library's purpose being the *generation* of code rather than its execution, the working core of $\texttt{mp}\mathbb{F}_q$ consists of roughly 18,000 lines of Perl code, which generate most of the `C` code. $\texttt{mp}\mathbb{F}_q$ is distributed at http://mpfq.gforge.inria.fr/.

In 2012, $\texttt{mp}\mathbb{F}_q$ evolved somewhat, in order to do the required code generation needed for evolutions of CADO-NFS, notably in relation with linear algebra over prime fields. A new release is planned soon, once hindrances related to the license of some code fragments are dealt with.

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 processors instruction (SSE, PCLMULQDQ).

The current version of GF2X is 1.1, released in May 2012 under the GNU GPL. Since 2009, GF2X can be used as an auxiliary package for the widespread software library NTL, as of version 5.5.

An LGPL-licensed portion of GF2X is also part of the CADO-NFS software package.

[1] http://www.ecrypt.eu.org/ebats/

5.7. CADO-NFS

Participants: Cyril Bouvier, Jérémie Detrey, Alain Filbois, Pierrick Gaudry, Alexander Kruppa, Emmanuel Thomé [contact], Paul Zimmermann.

CADO-NFS is a program to factor integers using the Number Field Sieve algorithm (NFS), originally developped 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. No new release was made in 2012, but several improvements have been made in the development version, with the help of Alain Filbois (SED engineer) and of Alexander Kruppa, recruited in October for a 2-year engineer contract.

Alain Filbois improved the *purge* program for filtering, by gaining a factor of about 5 in the input-output routines. Together with P. Zimmermann, he also wrote a special-purpose *clique removal* code for huge factorizations requiring out-of-core computing; this code has been used for a new filtering experiment on the relations collected for RSA-768 (not yet finished at the time of writing).

The *Objectif 1024* ADT started in 2012, with the recruitment of Alexander Kruppa as a engineer for 2 years. The four main objectives of this ADT are: (1) be able to use CADO-NFS routinely on clusters of 20 to 100 nodes, including on Amazon EC2; (2) develop precise tools to optimize parameters in the sieving phase; (3) develop more professional test mechanisms; (4) make two major releases of CADO-NFS, and advertize them on potential users.

Overall, CADO-NFS keeps improving its competitivity over alternative code bases. Improvements in CADO-NFS and new results obtained with CADO-NFS are described below.

CASCADE Project-Team

5. Software

5.1. MitMTool

Participants: Patrick Derbez, Jérémy Jean.

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 on 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.

GALAAD Project-Team

5. Software

5.1. Mathemagix, a free computer algebra environment

Participant: Bernard Mourrain.

http://www.mathemagix.org/

algebra, univariate polynomial, multivariate polynomial, matrices, series, fast algorithm, interpreter, compiler, hybrid software.

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, 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 embeding 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: Anaïs Ducoffe, Bernard Mourrain, Meriadeg Perrinel.

http://axel.inria.fr.

computational algebraic geometry, curve, implicit equation, intersection, parameterisation, resolution, surface, singularity, topology

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 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 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).

GEOMETRICA Project-Team

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 Rebufat, Laurent Rineau, Laurent Saboret, Stéphane Tayeb, Jane Tournois, 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, isothetic 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 14 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.

GRACE Team

4. Software

4.1. ECPP

F. Morain has been continually 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, using the fast version that he started developing in 2003. All of the code is written in C, and based on publicly available packages (GMP, mpfr, mpc, mpfrcx).

4.2. 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 most 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 Luca De Feo. This program forms a `gforge` project.

4.3. TIFA

The TIFA library (short for Tools for Integer FActorization), initially developed in 2006, 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.

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

4.4. Quintix

The Quintix library is a Mathemagix package, available at http://www.mathemagix.org/www/main/index.en.html. Quintix is a very efficient library for Galois rings, extensions of Galois rings and root-finding in Galois rings, 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 [23]. The source code is distributed under the General Public License (version 2 or higher).

4.5. finitefieldz

G. Quintin wrote the finitefieldz package which provides arithmetic for finite fields (of any characteristic) and towers of finite fields. He wrote this package with the help of Grégoire Lecerf during the first year of his PhD thesis. The package uses univariate polynomials and multiprecision integers, and also provides univariate polynomial root finding and factorization over finite fields.

4.6. Decoding

Decoding is a standalone C library licensed under the GPLv2. Its primary goal is to implement Guruswami–Sudan list decoding-related algorithms, as efficiently as possible. Its secondary goal is to give an efficient tool for the implementation of decoding algorithms (not necessarily list decoding algorithms) and their benchmarking.

For now (2012/12/13) you can use the library and have a working list decoding algorithm, but there is no unique decoding algorithm (though you can tell decoding to list decode up to half the minimum distance). The library is being further developedm and more algorithms will be added.

The library was presented at the 2012 International Symposium on Symbolic and Algebraic Computation.

LFANT Project-Team

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.
- Version of PARI/GP: 2.5.3
- Version of `gp2c`: 0.0.7pl4
- License: GPL v2+
- Programming language: C

5.2. GNU MPC

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

http://mpc.multiprecision.org/.

GNUMPC 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 GNUMPFR.

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.

2011 has seen the first release of the major version 1.0.

- Version: 1.0.1 *Fagus silvatica*
- License: LGPL v3+
- 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. MPFRCX

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: 0.4.1 *Cassava*
- 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: 0.2 *Blindhühnchen*
- License: GPL v2+
- Programming language: C

5.5. AVIsogenies

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

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

AVISOGENIES (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 (ℓ, ℓ)-isogenies between Jacobian varieties of genus-two hyperelliptic curves over finite fields of characteristic coprime to ℓ; practical runs have used values of ℓ in the hundreds.

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

- Version: 0.6
- 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

5.7. Euclid

Participant: Pierre Lezowski.

http://www.math.u-bordeaux1.fr/~plezowsk/euclid/index.php

EUCLID is a C program to compute the Euclidean minimum of a number field. It uses the PARI library.

- Version: 1.0
- License: GPL v2+
- Programming language: C

5.8. KleinianGroups

Participant: Aurel Page.

http://www.normalesup.org/~page/Recherche/Logiciels/logiciels.html

KLEINIANGROUPS is a Magma package that computes fundamental domains of arithmetic Kleinian groups.

- Version: 1.0
- License: GPL v3+
- Programming language: Magma

POLSYS Project-Team

5. Software

5.1. FGb

Participant: Jean-Charles Faugère [contact].

FGb is a powerful software for computing Groebner bases.It includes the new generation of algorihms 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-polsys.lip6.fr/~jcf/Software/FGb/index.html.

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

5.2. RAGlib

Participant: Mohab Safey El Din [contact].

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

5.3. Epsilon

Participant: Dongming 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)

VEGAS Project-Team

4. Software

4.1. 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], [29], [30], [31] 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 [24] bundled with the GMP multi-precision integer arithmetic [23]. 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 [25].

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.

4.2. 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 [28] 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 ALCIX 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 [27].

4.3. 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 [26] (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 [32]. We are currently developing a model of the *Bivariate Algebraic Kernel* based on our new bivariate polynomial solver [27]. This work is done in collaboration with F. Rouillier at Inria Paris - Rocquencourt and L. Peñaranda at the university of Athens.

4.4. Fast_polynomial: fast polynomial evaluation software

The library *fast_polynomial*[1] provides fast evaluation and composition of polynomials over several types of data. It is interfaced for the computer algebra system *sage*. This software is meant to be a first step toward a certified numerical software to compute the topology of algebraic curves and surfaces. It can also be useful as is and is submitted for integration in the computer algebra system *Sage*.

This software is focused on *fast online computation*, *multivariate evaluation*, *modularity*, and *efficiency*.

Fast online computation. The library is optimized for the evaluation of a polynomial on several point arguments given one after the other. The main motivation is numerical path tracking of algebraic curves, where a given polynomial criterion must be evaluated several thousands of times on different values arising along the path.

Multivariate evaluation. The library provides specialized fast evaluation of multivariate polynomials with several schemes, specialized for different types such as *mpz* big ints, *boost* intervals with hardware precision, *mpfi* intervals with any given precision, etc.

Modularity. The evaluation scheme can be easily changed and adapted to the user needs. Moreover, the code is designed to easily extend the library with specialization over new *C++* objects.

Efficiency. The library uses several tools and methods to provide high efficiency. First, the code uses templates, such that after the compilation of a polynomial for a specific type, the evaluation performance is equivalent to low-level evaluation. Locality is also taken into account: the memory footprint is minimized, such that an evaluation using the classical Hörner scheme will use $O(1)$ temporary objects and divide and conquer schemes will use $O(log(n))$ temporary objects, where n is the degree of the polynomial. Finally, divide and conquer schemes can be evaluated in parallel, using a number of threads provided by the user.

[1] http://trac.sagemath.org/sage_trac/ticket/13358

ALF Project-Team

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.

Microarchitecture temperature model **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.

Microarchitecture temperature modeling

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 [6]. 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.

Address trace compression **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 characterics, 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.

Unpredictable random number generator

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 Volatile 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 [12]. 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.

Performance, hardware counters, analysis tool.

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.
- Any expression can be computed, using the basic arithmetic operators, constants, and counter values.
- A configuration file lets users define their prefered setup, as well as custom expressions.

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

With the ever raising complexity of embedded applications and platforms, the need for efficient and customizable compilation flows is stronger than ever. This need of flexibility is even stronger when it comes to research compiler infrastructures that are necessary to gather quantitative evidence of the performance/energy or cost benefits obtained through the use of reconfigurable platforms. From a compiler point of view, the challenges exposed by these complex reconfigurable platforms are quite significant, since they require the compiler to extract and to expose an important amount of coarse and/or fine grain parallelism, to take complex resource constraints into consideration while providing efficient memory hierarchy and power management.

Because they are geared toward industrial use, production compiler infrastructures do not offer the level of flexibility and productivity that is required for compiler and CAD tool prototyping. To address this issue, we have designed an extensible source-to-source compiler infrastructure that takes advantage of leading edge model-driven object-oriented software engineering principles and technologies.

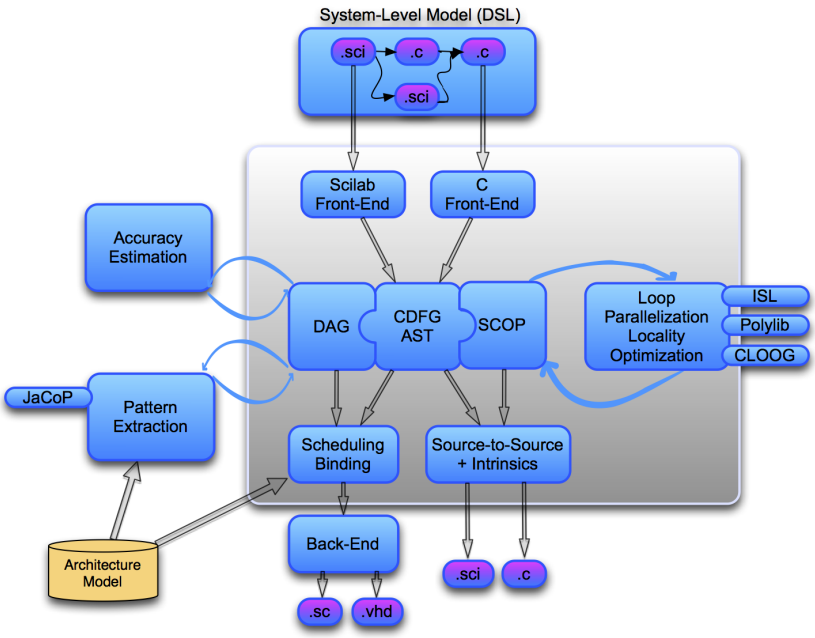

Figure 2. CAIRN*'s general software development framework.*

Figure 2 shows the global framework that is being developed in the group. Our compiler flow mixes several types of intermediate representations. The baseline representation is a simple tree-based model enriched with control flow information. This model is mainly used to support our source-to-source flow, and serves as the backbone for the infrastructure. We use the extensibility of the framework to provide more advanced representations along with their corresponding optimizations and code generation plug-ins. For example,

for our pattern selection and accuracy estimation tools, we use a data dependence graph model in all basic blocks instead of the tree model. Similarly, to enable polyhedral based program transformations and analysis, we introduced a specific representation for affine control loops that we use to derive a Polyhedral Reduced Dependence Graph (PRDG). Our current flow assumes that the application is specified as a system level hierarchy of communicating tasks, where each task is expressed using C (or Scilab in the short future), and where the system level representation and the target platform model are defined using Domain Specific Languages (DSL).

Gecos (Generic Compiler Suite) is the main backbone of CAIRN's flow. It is an open source Eclipse-based flexible compiler infrastructure developed for fast prototyping of complex compiler passes. Gecos is a 100% Java based implementation and is based on modern software engineering practices such as Eclipse plugin or model-driven software engineering with EMF (Eclipse Modeling Framework). As of today, our flow offers the following features:

- An automatic floating-point to fixed-point conversion flow (for HLS and embedded processors). **ID.Fix** is an infrastructure for the automatic transformation of software code aiming at the conversion of floating-point data types into a fixed-point representation. http://idfix.gforge.inria.fr.
- A polyhedral-based loop transformation and parallelization engine (mostly targeted at HLS). http://gecos.gforge.inria.fr. It was used for source-to-source transformations in the context of Nano2012 projects in collaboration with STMicroelectronics.
- A custom instruction extraction flow (for ASIP and dynamically reconfigurable architectures). **Durase** and **UPaK** are developed for the compilation and the synthesis targeting reconfigurable platforms and the automatic synthesis of application specific processor extensions. They use advanced technologies, such as graph matching and graph merging together with constraint programming methods.
- Several back-ends to enable the generation of VHDL for specialized or reconfigurable IPs, and SystemC for simulation purposes (e.g. fixed-point simulations).

5.2. Gecos

Participants: Steven Derrien [corresponding author], Nicolas Simon, Maxime Naullet, Antoine Floc'h, Antoine Morvan, Clément Guy.

Keywords: source-to-source compiler, model-driven software engineering, retargetable compilation.

The Gecos (Generic Compiler Suite) project is a source-to-source compiler infrastructure developed in the CAIRN group since 2004. It was designed to enable fast prototyping of program analysis and transformation and is aims the hardware synthesis and retargetable compilation domains.

Gecos is 100% Java based and takes advantage of modern model driven software engineering practices. It uses the Eclipse Modeling Framework (EMF) as an underlying infrastructure and takes benefits of its features to make it easily extensible. Gecos 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 serves as a backbone infrastructure to projects of the group (project S2S4HSL, ID.FIX). Part of the framework is jointly developed with Colorado State University and since 2012 it is used in the context of the ALMA European project.

Development in Gecos in 2012 have mostly focused on the polyhedral loop transformation engine and its use for hardware synthesis. As a part of the ALMA project, significant efforts are also being made to develop a coarse-grain parallelization engine targeting a distributed memory machine model.

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

Participants: Daniel Menard, Olivier Sentieys [corresponding author], Romuald Rocher, Nicolas Simon.

Keywords: fixed-point arithmetic, source-to-source code transformation, accuracy optimization, dynamic range evaluation

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 two main modules corresponding to the fixed-point conversion (ID.Fix-Conv) and the accuracy evaluation (ID.Fix-Eval)

The different developments carried-out in 2012 allowed us to obtain a fixed-point conversion tool handling functions, conditional structures and repetitive structures having a fixed number of iterations during time. New optimization algorithms have been added. A simulator has been created to verify the results from our analytical approach. For the accuracy evaluation (Acc.Eval), conditional structures and correlation between noise sources have been considered. Some optimizations have been implemented to reduce the computing time and the division operator treatment has been integrated. A tutorial has also been created to install and use this tool.

The development of this tool has been achieved thanks to 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 [corresponding author], François Charot, Antoine Floc'h.

Keywords: compilation for reconfigurable systems, pattern extraction, constraint-based programming.

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* [123]. 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 [corresponding author], François Charot, Antoine Floc'h.

Keywords: compilation for reconfigurable systems, instruction-set extension, pattern extraction, graph covering, constraint-based programming.

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 [corresponding author], Olivier Berder, Arnaud Carer, Steven Derrien.

Keywords: Wireless Sensor Networks, Low Power, Preamble Sampling MAC Protocol, Hardware and Software Platform

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 [86] 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.
- Finally, a last daughter board is dedicated to energy harvesting techniques. Based on the energy management component LTC3108 from Linear Technologies, the board can be configured with several types of stored energy (batteries, micro-batteries, super-capacitors) and several types of energy sources (a small solar panel to recover photovoltaic energy, a piezoelectric sensor for mechanical energy and a Peltier thermal energy sensor).

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 [102], 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 [corresponding author], Laurent Perraudeau.

Keywords: SoC modeling, SystemC simulation model

Figure 3. CAIRN*'s PowWow motherboard with radio and energy-harvesting boards connected*

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.

CAMUS Team

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 [15].

It allows for example to compute the number of solutions of a Presburger formula by eliminating existential 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

Participant: Alain Ketterlin.

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. In particular, it has been used by Andres Charif-Rubial for his PhD work (Université de Versailles Saint-Quentin en Yvelines), and is now included in a released tool called MAQAO (http://www.maqao.org). Our code is also used by Jean-Thomas ACQUAVIVA, at Commissariat à l'Énergie Atomique, for work on compressing instruction traces. These colleagues have slightly modified the code we gave them. We plan to release a stable version incorporating most of their changes in the near future. We also plan to change the license to avoid such drifts in the future.

[8] http://icps.u-strasbg.fr/PolyLib
[9] http://ZPolyTrans.gforge.inria.fr

5.4. Binary files decompiler

Participant: Alain Ketterlin.

Our research on efficient memory profiling has led 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 [45]. 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.

5.5. Parwiz: a dynamic dependency analyser

Participant: Alain Ketterlin.

We have developed 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.

5.6. VMAD software and LLVM

Participants: Alexandra Jimborean, Philippe Clauss, Jean-François Dollinger, Aravind Sukumaran-Rajam, Juan Manuel Martinez Caamaño.

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 .

As the final result of Alexandra Jimborean's PhD thesis, the VMAD framework now handles speculative parallelization of loop nests by applying dynamically polyhedral code transformations. It is currently extended to handle even more advanced code transformations as tiling in particular, and also to handle codes whose memory behavior is not fully linear.

Alexandra Jimborean (PhD student), Matthieu Herrmann (former Master student), Luis Mastrangelo (former Master student), Juan Manuel Martinez Caamaño (Master student), Jean-François Dollinger (PhD student), Aravind Sukumaran-Rajam (PhD student) and Philippe Clauss are the main contributors of this software. It is not yet distributed.

5.7. Polyhedral prover

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

polyhedral transformations, verified compiler

We are currently developing a formal proof of program transformations based on the polyhedral model. We use the CompCert verified compiler [51] as a framework. This tool is written in the specification language of Coq. It is connected to the activity described in section 6.5 .

COMPSYS Project-Team

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, Suif 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 (ILP-based) methods, 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. It 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

Participant: 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 [17], 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. More details on the underlying theory are available in previous reports. 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:

- A 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.
- An extended language of pragmas to feed the source code with compilation directives (a schedule, for example).
- A symbolic layer on polyhedral libraries Polylib (set operations on polyhedra) and Piplib (parameterized ILP). This feature simplifies drastically the developer task.
- Some dependence analysis (polyhedral dependence graph, array dataflow analysis), array region analysis, array liveness analysis.
- A C and VHDL code generation based on the ideas of P. Boulet and P. Feautrier [17].

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 [19] 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 [Compsys/Zettice].

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. IceBuilder

Participants: Christophe Alias, Alexandru Plesco [Compsys/Zettice].

IceBuilder is the HLS tool to be transferred in the start-up Zettice. It is a compiler, whose input is a C program annotated with pragmas, and whose output is an equivalent hardware description as synthesizable VHDL. Also, IceBuilder produces a non-synthesizable SystemC description for debugging purpose. As for any compiler, IceBuilder consists into two steps: (i) a front-end, which generates an intermediate representation from the C program, and (ii) a back-end, which translates the intermediate representation into hardware. The intermediate representation of IceBuilder is a data-aware process network (DPN) (see Section 6.3). The front-end does most of the high-level optimizations (communication pipelining, buffer sizing, datapath pipeline scheduling), which are explicitly represented in the DPN. The front-end is implemented as a separate tool, Dcc, so as to be reused with different targets, for instance GPGPUs. Then, the back-end generates the hardware implementation of the DPN. It produces and connects the required buffers, multiplexors, demultiplexors, synchronization channels, finite-state machines, and datapaths.

IceBuilder represents more than 3000 lines of C++ code.

5.9. Dcc

Participants: Christophe Alias, Alexandru Plesco [Compsys/Zettice].

Dcc is the front-end of the IceBuilder tool. Dcc takes as input a C program annotated with pragmas and produces an optimized data-aware process network (DPN). To do so, Dcc reuses most of the analysis implemented in PoCo (dataflow analysis and control generation), Chuba (communication pipelining), Cl@k and Bee (buffer sizing). Dcc and DPNs are very critical parts of IceBuilder and will require a patent before any publication.

Dcc represents more than 2500 lines of C++ code.

5.10. 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.

5.11. RanK

Participants: Christophe Alias, Alain Darte, Paul Feautrier, Laure Gonnord [Compsys/LIFL].

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 5.10), representing the control structure of the program to be analyzed. 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++.

5.12. SToP

Participants: Christophe Alias, Guillaume Andrieu [LIFL], Laure Gonnord [Compsys/LIFL].

SToP (Scalable Termination of Programs) is the implementation of the modular termination technique presented in Section 6.7 . It takes as input a large irregular C program and conservatively checks its termination. To do so, SToP generates a set of small programs whose termination implies the termination of the whole input program. Then, the termination of each small program is checked thanks to RanK. In case of success, SToP infers a ranking (schedule) for the whole program. This schedule can be used in a subsequent analysis to optimize the program.

SToP represents more than 2000 lines of C++.

5.13. 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 (OBDD). It uses PIP for testing the feasibility – or unfeasibility – of a conjunction of affine inequalities.

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 (quasi-affine syntax tree) syntax.

5.14. LAO Developments in Aggressive Compilation

Participants: Benoit 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. This continued with developments made during the master internships and PhD theses of Florent Bouchez, Benoit Boissinot, and Quentin Colombet, and post-doctoral works of Sebastian Hack and Florian Brandner. 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 second ILP-based solution. 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.

5.15. LAO Developments in JIT Compilation

Participants: Benoit Boissinot, Florian Brandner, Quentin Colombet, 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. Recent efforts 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.

5.16. 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. Beside 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 used 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. Open64/Path64 emitter for TireX and its LAO back-end are available at https://compilation.ens-lyon.fr/. MinIR was presented at WIR'11 [28].

AOSTE Project-Team

5. Software

5.1. TimeSquare

Participants: Charles André, Nicolas Chleq, Julien Deantoni, 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: 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 represents 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 deadline and latency 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 additionaly 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 commun 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. It can be downloaded from the site http://pauillac.inria.fr/~ddr/sas-dameid/.

CONVECS Team

5. Software

5.1. The CADP Toolbox

Participants: Hubert Garavel [correspondent], Frédéric Lang, Radu Mateescu, Wendelin Serwe.

We maintain and enhance CADP (*Construction and Analysis of Distributed Processes* – formerly known as *CAESAR/ALDEBARAN Development Package*) [4], a toolbox for protocols and distributed systems engineering (see http://cadp.inria.fr). In this toolbox, we develop and maintain the following tools:

- CAESAR.ADT [41] 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.
- CAESAR [48], [47] is a compiler that translates LOTOS processes into either C code (for rapid prototyping and testing purposes) or finite graphs (for verification purposes). 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/CAESAR [42] 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/CAESAR plays a central role in CADP by connecting language-oriented tools with model-oriented tools. OPEN/CAESAR 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/CAESAR 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 enable 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 the graph layout produced by BCG_DRAW to be modified interactively,
  - 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_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 evaluation and diagnostic generation fixed point algorithms for usual temporal logics (such as HML [50], CTL [37], ACTL [38], etc.) to be defined in XTL.
- PBG (*Partitioned BCG Graph*) is a file format implementing the theoretical concept of *Partitioned LTS* [46] and providing a unified access to a graph partitioned in fragments distributed over a set of remote machines, possibly located in different countries. The PBG format is supported by several tools, such as:
  - PBG_CP, PBG_MV, and PBG_RM, which facilitate standard operations (copying, moving, and removing) on PBG files, maintaining consistency during these operations,
  - PBG_MERGE (formerly known as BCG_MERGE), which transforms a distributed graph into a monolithic one represented in BCG format,
  - PBG_INFO, which displays various statistical information about a distributed graph.
- The connection between explicit models (such as BCG graphs) and implicit models (explored on the fly) is ensured by OPEN/CAESAR-compliant compilers, e.g.:
  - BCG_OPEN, for models represented as BCG graphs,
  - CAESAR.OPEN, for models expressed as LOTOS descriptions,
  - EXP.OPEN, for models expressed as communicating automata,
  - FSP.OPEN, for models expressed as FSP [55] descriptions,
  - LNT.OPEN, for models expressed as LNT descriptions, and
  - SEQ.OPEN, for models represented as sets of execution traces.

The CADP toolbox also includes TGV (*Test Generation based on Verification*), which has been developed by the VERIMAG laboratory (Grenoble) and the VERTECS project team at Inria Rennes – Bretagne-Atlantique.

The CADP tools are well-integrated and can be accessed easily using either the EUCALYPTUS graphical interface or the SVL [43] 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 [correspondent], Frédéric Lang.

We develop a compiler named TRAIAN for translating LOTOS NT descriptions into C programs, which will be used for simulation, rapid prototyping, verification, and testing.

The current version of TRAIAN, which handles LOTOS NT types and functions only, has useful applications in compiler construction [44], being used in all recent compilers developed by the CONVECS team.

The TRAIAN compiler can be freely downloaded from the CONVECS Web site (see http://convecs.inria.fr/software/traian).

5.3. The PIC2LNT Translator

Participants: Radu Mateescu, Gwen Salaün [correspondent].

We develop a translator named PIC2LNT from an applied π-calculus (see Section 6.1) to LNT, which enables the analysis of concurrent value-passing mobile systems using CADP.

PIC2LNT is developed by using the SYNTAX tool (developed at Inria Paris-Rocquencourt) for lexical and syntactic analysis together with LOTOS NT for semantical aspects, in particular the definition, construction, and traversal of abstract trees.

The PIC2LNT translator can be freely downloaded from the CONVECS Web site (see http://convecs.inria.fr/software/pic2lnt).

DART Project-Team

5. Software

5.1. 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 [35] 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 [37]. 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 softwre evaluation: A-2, SO-4, SM-2, EM-1, SDL-2, DA-4, CD-4, MS-4, TPM4
- Version: 2.1.0

ESPRESSO Project-Team

5. Software

5.1. The Polychrony toolset and its hypertext source documentation

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 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/SSME platform, a front-end to the Signal toolbox in the Eclipse environment. The SME/SSME 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 differents 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 Inria GForge: 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 SSME platform, including the executable of the Signal toolbox.

The Polychrony toolset currently runs on Linux, MacOS and Windows systems.

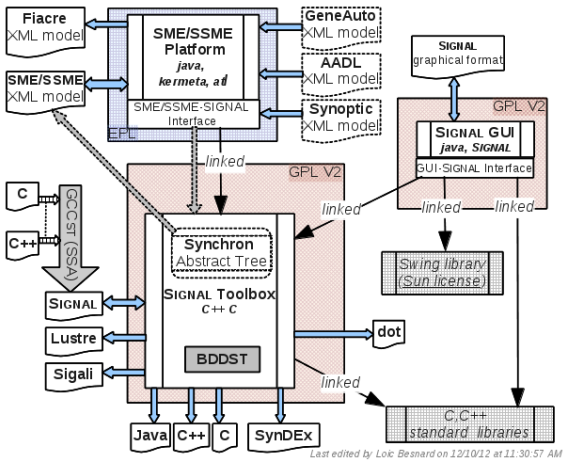

Figure 7. The Polychrony toolset high-level architecture

The Geensoft company, now part of Dassault Systèmes, supplies a commercial implementation of Polychrony, called RT-Builder, used for industrial scale projects (see http://www.geensoft.com).

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 7 .

5.2. The Eclipse interface

Participants: Loïc Besnard, Yue Ma, Huafeng Yu.

Meta-modeling, Eclipse, Ecore, Signal, Model transformation

We have developed a meta-model and interactive editor of Polychrony in Eclipse. Signal-Meta is the meta-model 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 toolchain. 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 toolchain.

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

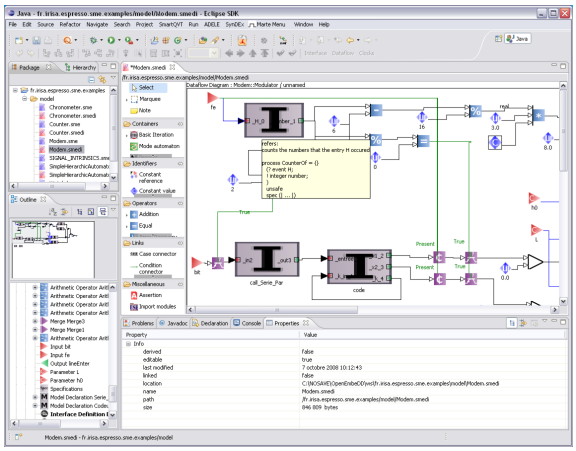

Figure 8. Eclipse SME Environment.

Polychrony (see figure 8) 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 data-flow) part, and one to model all explicit clock relations and dependences. The SME environment is available through the ESPRESSO update site [23], in the current OpenEmbeDD distribution [22], or in the TopCased distribution [25]. Note that a new meta-model of Signal, called SSME (Syntactic Signal-Meta under Eclipse), closer to the Signal abstract syntax, has been defined and integrated in the Polychrony toolset.

5.3. Integrated Modular Avionics design using Polychrony

Participants: Loïc Besnard, Thierry Gautier, Paul Le Guernic, Jean-Pierre Talpin.

The Apex interface, defined in the ARINC standard [26], 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 [27]). 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 [5] 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.

MUTANT Project-Team

5. Software

5.1. Antescofo

Participants: Arshia Cont, Jean-Louis Giavitto, Florent Jacquemard, José Echeveste.

Antescofo is a modular polyphonic Score Following system as well as a Synchronous Programming language for musical composition. The module allows for automatic recognition of music score position and tempo from a realtime audio Stream coming from performer(s), making it possible to synchronize an instrumental performance with computer realized elements. The synchronous language within Antescofo allows flexible writing of time and interaction in computer music.

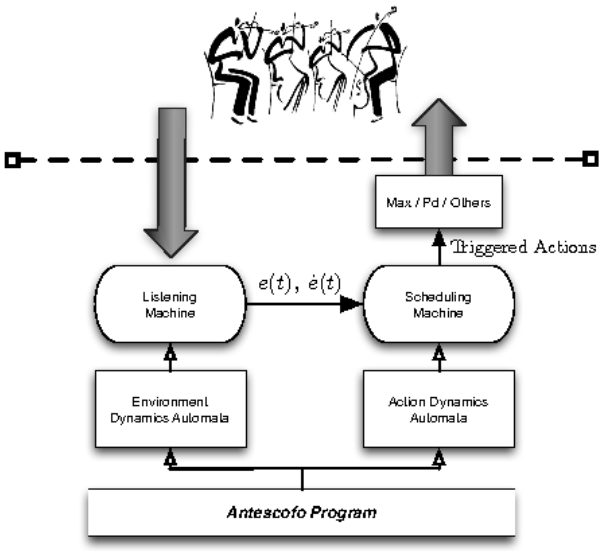

Figure 2. General scheme of Antescofo virtual machine

Antescofo is developed as modules for *Max* and *PureData* real-time programming environments.

5.2. Antescofo Visual Editor

Participants: Thomas Coffy [ADT], Arshia Cont, José Echeveste.

The Antescofo programming language can be extended to visual programing to better integrate existing scores and to allow users to construct complex and embedded temporal structures that are not easily integrated into text. This project has started in October 2012 thanks to Inria ADT Support. The foundations of a visual editor is set and the goal is to release a standalone editor for Antescofo programs in 2013.

PARKAS Project-Team

5. Software

5.1. Lucid Synchrone

Participant: Marc Pouzet [contact].

Synchronous languages, type and clock inference, causality analysis, compilation

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.

It is distributed under binary form, at URL http://www.di.ens.fr/~pouzet/lucid-synchrone/.

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, originaly introduced in Lucid Synchrone are now integrated in the new SCADE 6 compiler developped 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.

Programming language, synchronous reactive programming, concurrent systems, dedicated type-systems.

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, embedded in an ML language (OCaml).

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://rml.lri.fr. 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 Pierre and Marie Curie University and for the simulation of sensor networks at France Telecom and Verimag (CNRS, Grenoble).

In 2012, a new automatic build system for ReactiveML program based on ocamlbuild has been implemented. A new static analysis which checks that programs cooperate has been developed. A full ReactiveML toplevel compiled into JavaScript has been made available at http://rml.lri.fr/tryrml. The ReactiveML distribution has also been cleaned up.

5.3. Heptagon

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

Synchronous languages, compilation, optimizing compilation, parallel code generation, behavioral synthesis.

Heptagon is an experimental language for the implementation of embedded real-time reactive systems. It is developed inside the Synchronics large-scale initiative, in collaboration with Inria Rhones-Alpes. It is essentially a subset of Lucid Synchrone, without type inference, type polymorphism and higher-order. It is thus a Lustre-like language extended with hierchical automata in a form very close to SCADE 6. The intention for making this new language and compiler is to develop new aggressive optimization techniques for sequential C code and compilation methods for generating parallel code for different platforms. This explains much of the simplifications we have made in order to ease the development of compilation techniques.

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: an n-synchronous data-flow programming language

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

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.

The language compiler and associated tools are available in a binary form at http://www.lri.fr/~mandel/lucy-n.

In 2012, a first version of the code generator has been distributed. The typing algorithms has been improved.

5.5. ML-Sundials

Participants: Timothy Bourke, Marc Pouzet [contact].

ML-Sundials library provides an Ocaml interface to the Sundials numerical suite [7] (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 developping.

5.6. GCC

Participants: Albert Cohen [contact], Tobias Grosser, Antoniu Pop, Feng Li, Riyadh Baghdadi, Nhat Minh Le.

Compilation, optimizing compilation, parallel data-flow programming automatic parallelization, polyhedral compilation. http://gcc.gnu.org

Licence: GPLv3+ and LGPLv3+

[7] https://computation.llnl.gov/casc/sundials/main.html

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.

Tobias Grosser is the maintainer of the Graphite optimization pass of GCC.

5.7. isl

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

Presburger arithmetic, integer linear programming, polyhedral library, automatic parallelization, polyhedral compilation. http://freshmeat.net/projects/isl

Licence: MIT

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.

5.8. ppcg

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

Presburger arithmetic, integer linear programming, polyhedral library, automatic parallelization, polyhedral compilation. http://freshmeat.net/projects/ppcg

Licence: LGPLv2.1+

More tools are being developed, based on isl. PPCG is our source-to-source research tool for automatic parallelization in the polyhedral model. It serves as a test bet for many algorithms and heuristics published by our group, and is currently the best automatic parallelizer for CUDA (on the Polybench suite).

5.9. Ott: tool support for the working semanticist

Participant: Francesco Zappa Nardelli [contact].

Languages, semantics, tool support, theorem prouvers.

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). It is widely used in the scientific community.

In 2012 we implemented several bug-fixes and we kept the theorem prouver backends up-to date with the prover evolution. We have also been working toward a closer integration with the Lem tool.

The currently relased version is 0.21.2.

5.10. Lem: a tool for lightweight executable semantics

Participant: Francesco Zappa Nardelli [contact].

Languages, semantics, tool support, theorem prouvers.

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/.

In collaboration with Scott Owens (U. Kent, UK) and Peter Sewell (U. Cambridge, UK).

5.11. Cmmtest: a tool for hunting concurrency compiler bugs

Participants: Francesco Zappa Nardelli [contact], Robin Morisset, Pankaj Pawan.

Languages, concurrency, memory models, C11/C++11, compiler, bugs.

The cmmtest tool performs random testing of C and C++ compilers against the C11/C++11 memory model. A test case is any well-defined, sequential C program; for each test case, cmmtest:

1. compiles the program using the compiler and compiler optimisations that are being tested;
2. runs the compiled program in an instrumented execution environment that logs all memory accesses to global variables and synchronisations;
3. compares the recorded trace with a reference trace for the same program, checking if the recorded trace can be obtained from the reference trace by valid eliminations, reorderings and introductions.

Although not yet publicly distributed, cmmtest already identified several mistaken write introductions and other unexpected behaviours in the latest release of the gcc compiler. These have been promptly fixed by the gcc developers.

POP ART Project-Team

5. Software

5.1. NBac

Participant: Bertrand Jeannet.

NBAC (Numerical and Boolean Automaton Checker) [15] 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 [63]. 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 verifying and debugging LUSTRE programs [65] [52] [36]. It is connected to the LUSTRE toolset [16]. 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 has been used by the test generator STG [17] [42] [66] for selecting test cases.

It has recently been superseded by REAVER (see Section 5.2).

5.2. ReaVer

Participant: Peter Schrammel.

REAVER (REActive VERifier [18]) is a tool framework for the safety verification of discrete and hybrid systems specified by logico-numerical data-flow languages, like LUSTRE, LUCIDSYNCHRONE or ZELUS. It provides time-unbounded analysis based on abstract interpretation techniques. In many aspects it is the successor of NBAC (see Section 5.1).

It features partitioning techniques and several logico-numerical analysis methods based on Kleene iteration with widening and descending iterations, abstract acceleration, max-strategy iteration, and relational abstractions; logico-numerical product and power domains (based on the APRON and BddApron domain libraries) with convex polyhedra, octagons, intervals, and template polyhedra; and frontends for the hybrid NBAC format, LUSTRE via lus2nbac, and ZELUS/LUCIDSYNCHRONE. Compared to NBAC, it is connected to higher-level, more recent synchronous and hybrid languages, and provides much more options regarding analysis techniques.

It has been used for several experimental comparisons published in papers and it integrates all the methods developped by Peter Schrammel in its PhD.

5.3. Implementations of Synchronous Programs

Participant: Alain Girault.

[15] http://pop-art.inrialpes.fr/people/bjeannet/nbac/
[16] http://www-verimag.imag.fr/The-Lustre-Toolbox.html
[17] http://www.irisa.fr/prive/ployette/stg-doc/stg-web.html
[18] http://members.ktvam.at/schrammel/research/reaver

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[19]. Our first scheduling heuristic produces static multiprocessor schedules tolerant to a specified number of processor and communication link failures [55]. The basic principles upon which we rely to make the schedules fault tolerant is, on the one hand, the active replication of the operations [56], 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) [57]. 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] [33][17], [11]. Our results on fault tolerance are summarized in a web page [20].

5.4. Apron and BddApron Libraries

Participant: Bertrand Jeannet.

5.4.1. Principles

The APRON library [21] is dedicated to the static analysis of the numerical variables of a program by abstract interpretation [43]. 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 provides:

- 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 [53], which is the underlying abstract domain of the tool FLUCTUAT [51] 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 [30] in the context of the ANR ASOPT project.

The BDDAPRON library [22] 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 the manipulation of 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.

[19] http://www-rocq.inria.fr/syndex
[20] http://pop-art.inrialpes.fr/~girault/Projets/FT
[21] http://apron.cri.ensmp.fr/library/
[22] http://pop-art.inrialpes.fr/~bjeannet/bjeannet-forge/bddapron/index.html

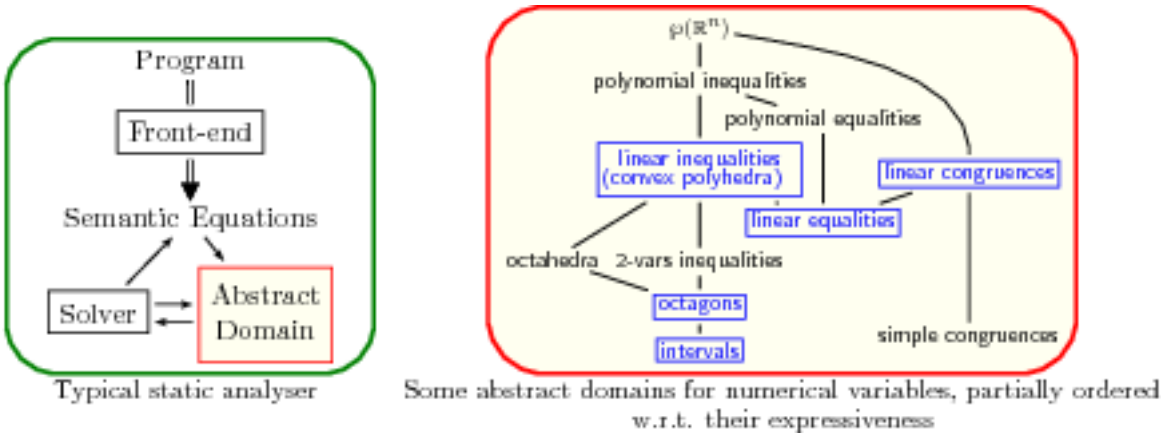

Figure 1. Typical static analyser and examples of abstract domains

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

Participant: 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. Cosyma

Participants: Gregor Goessler [contact person], Sebti Mouelhi.

We have developed COSYMA, a tool for automatic controller synthesis for incrementally stable switched systems based on multi-scale discrete abstractions (see 6.2.1). The tool accepts a description of a switched system represented by a set of differential equations and the sampling parameters used to define an approximation of the state-space on which discrete abstractions are computed. The tool generates a controller — if it exists — for the system that enforces a given safety or time-bounded reachability specification.

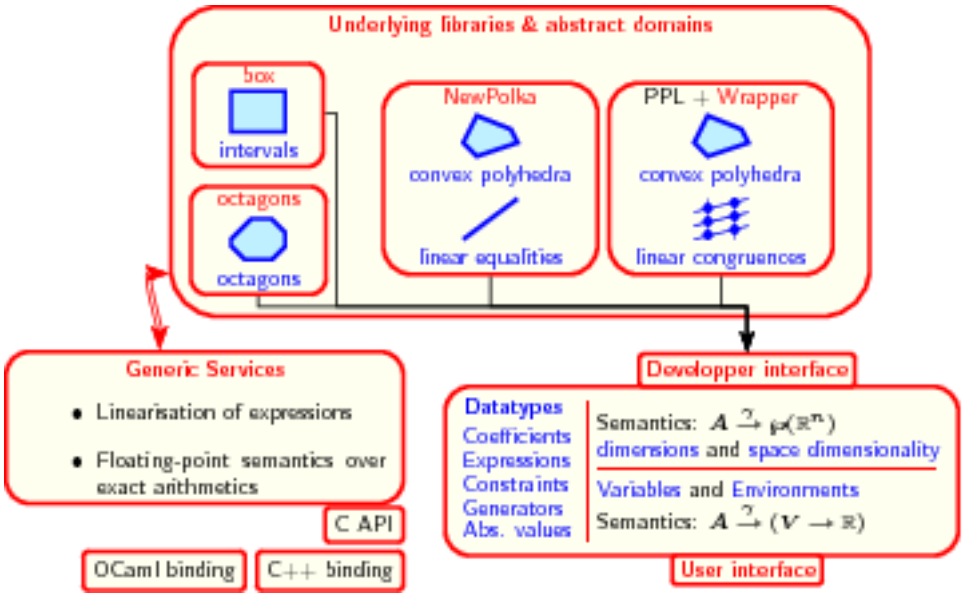

Figure 2. Organisation of the APRON *library*

5.5.3. 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 SIGALI[23] and Mode Automata [24]. It is the result of a collaboration with Eric Rutten from the SARDES team.

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.

5.5.4. Rapture

Participant: Bertrand Jeannet.

RAPTURE[25] [64] [46] 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.

[23] http://www.irisa.fr/vertecs/Logiciels/sigali.html
[24] http://www-verimag.imag.fr
[25] http://pop-art.inrialpes.fr/people/bjeannet/rapture/rapture.html

5.5.5. The Interproc family of static analyzers

Participant: Bertrand Jeannet [contact person].

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 §8.1.2.1).

- FIXPOINT[26]: 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 post-fixpoint computation with widening and descending iterations in a sound way [59].
- INTERPROC[27]: 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 [28]. It is used as the experimental platform of the ASOPT ANR project.
- CONCURINTERPROC extends INTERPROC with concurrency, for the analysis of multithreaded programs interacting via shared global variables. It is also deployed through a web-interface [29].
- PINTERPROC extends INTERPROC with pointers to local variables. It is also deployed through a web-interface [30].

5.5.6. Heptagon/BZR

Participant: Gwenaël Delaval.

HEPTAGON is a dataflow synchronous language, inspired from LUCIDSYNCHRONE[31]. 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 SYNCHRONICSINRIA large-scale action.

HEPTAGON has been used to built BZR [32], which is an extension of the former with contracts constructs. These contracts allow to express dynamic temporal properties on the inputs and outputs of HEPTAGON node. These properties are then enforced, within the compilation of a BZR program, by discrete controller synthesis, using the SIGALI tool [33]. The synthesized controller is itself generated in HEPTAGON, allowing its analysis and compilation towards different target languages (C, JAVA, VHDL).

[26] http://http://pop-art.inrialpes.fr/people/bjeannet/bjeannet-forge/fixpoint
[27] http://pop-art.inrialpes.fr/people/bjeannet/bjeannet-forge/interproc
[28] http://pop-art.inrialpes.fr/interproc/interprocweb.cgi
[29] http://pop-art.inrialpes.fr/interproc/concurinterprocweb.cgi
[30] http://pop-art.inrialpes.fr/interproc/pinterprocweb.cgi
[31] http://www.di.ens.fr/~pouzet/lucid-synchrone/
[32] http://bzr.inria.fr
[33] http://www.irisa.fr/vertecs/Logiciels/sigali.html

S4 Project-Team

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 [8]. 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/licences/Licence_CeCILL-C_V1-en.html).

5.2. Synet: A General Petri-Net Sythesis 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 [33]. It can synthesize Petri-nets from automata or regular expressions 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 [35]. 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 [32]. Synthesis is constrained to produce so-called distributables nets [34], 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 it 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. ANR Open-PEOPLE platform

Participants: Fabrice Vergnaud, Jérôme Vatrinet, 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 ergonomy 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 is a key part of our platform, and the quality of its design has 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 had gone on.

We had continued the work to solidify our development platform supporting our work and that of our partners. We had produced a finer grained definition of the software platform functionalities, and a more precise definition of the tools integration protocol. We had worked towards the corresponding implementation documents, adding two new delivrables about the architecture of the software platform and the ergonomics of the software platform. For the latter, we had extensively interviewed user about ergonomics and designed several GUI mockups. We had progressed on the implementation of the software platform, especially with respect to the internet portal to remote-control the hardware platform. We had 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 had 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 had been completed in Fall 2011.

We had 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 had 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 had realized an ergonomic study for the consumption laws editors, with mockups and user interviews and validation. We had worked on the implementation of the editors for the consumption laws, which had required learning new environments and development tools (related to the EMF framework and the AADL, QUDV and MathML models). As a consequence, we had completed the implementation of the GUI and engine to create units and quantities. We had finalized the architecture needed to integrate external modules in the platform.

In 2012, this work went on. Basically, 2012 was the year of the concrete Open-PEOPLE platform, where all our efforts finally came to maturity. We thus completed global GUI of the Open-PEOPLE Software Platform. We performed the integration of various external tools and modules and the . We provided several improvements to the Remote Control Module providing access to the Hardware Platform. We finalized the implementation of consumption laws editors. We implemented the export and import functionality of Open-PEOPLE models. We created a new community-based website to allow sharing of Open-PEOPLE models.

We overall progressed as forecast in an iterative development and release schedule.

Version 1.0 (2012-04-06) was the first embodiment and public release of the Open-PEOPLE platform.

Version 1.1 (2012-06-12) added a default environment with pre-set Units, Quantities and AADL Property associations, asynchronous file uploads and downloads in the Remote Control Module, and better handling of big files (file size limit is now 4GB), and several bug fixes.

Version 1.3 (2012-09-27) changed internal mechanism of QUDV serialization (quantities, units and property associations), added version information to QUDV and Weaving meta-models, added internal builders to automatically generates QUDV and Weaving configuration files, added support of OSATE 2, improved UI reactivity (especially during file transfers), added progress bars for the remote control, and several bug fixes.

Version 1.4 (2012-10-25) added the Adele Graphical Editor, new OSATE 2 Snapshot, and several bugfixes.

Version 2.0 (2012-12-13) added RDALTE, AADL2SystemC, and a Standard environment with models and model sharing implemented (including a community sharing website), a new snapshot of OSATE2.

5.2. VITRAIL

Participants: Frédéric Diss, 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.

In 2012, we continued working on the analysis of software, gathering statistics about polymorphism in Java programs, aiming at comparing various type analyses make statically (CHA, RTA, VTA) and the dynamic trace provided by (a) real execution(s). This work is going on and has not been published yet.

We also developed a public website for the VITRAIL project, which is going live these days.

VERTECS Project-Team

5. Software

5.1. 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 deposited 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, devoted to the control of infinite systems modeled by STS.

5.2. 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 [27] [23]. 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.

Sigali is also integrated as part of the compiler of the language BZR.

ABSTRACTION Project-Team

5. Software

5.1. The Apron Numerical Abstract Domain Library

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

Convex polyhedra, Intervals, Linear equalities, Numerical abstract domain, Octagons.

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 InriaGForge. 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).

In 2012, APRON has been used in several researches conducted within or in collaboration with the Abstraction project-team: the design of a sufficient-condition generator [23] and the design of a constraint solver based on abstract domains [25].

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.

Absence of runtime error, Abstract interpretation, Static analysis, Verifier.

ASTRÉE is a static analyzer for sequential programs based on abstract interpretation [41], [32], [42], [34].

The ASTRÉE static analyzer [31], [46][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, correspondent], Radhia Cousot, Jérôme Feret, Antoine Miné, Xavier Rival.

Absence of runtime error, Abstract interpretation, Data races, Interference, Memory model, Parallel software, Static analysis, Verifier.

ASTRÉEA is a static analyzer prototype for parallel software based on abstract interpretation [43], [44], [36]. It started with support from THÉSÉE ANR project (2006–2010) and is continuing within the ASTRÉEA project (2012–2015).

The ASTRÉEA prototype www.astreea.ens.fr is a fork of the ASTRÉE static analyzer (see 5.2) that adds support for analyzing parallel embedded C software.

ASTRÉEA 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. ASTRÉEA 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 aeronautic software. Additionally, ASTRÉEA 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). ASTRÉEA checks for the same run-time errors as ASTRÉE, with the addition of data-races.

Compared to ASTRÉE, ASTRÉEA 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.

ASTRÉEA 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 a few tens of hours on a 2.9 GHz 64-bit intel server using one core and generates around 1,200 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. The development of AstréeA continues within the scope of the ASTRÉEA ANR project (Section 8.1.1.2).

5.4. The OpenKappa modeling plateform

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

Causal traces, Model reduction, Rule-based modelling, Simulation, Static analysis.

OPENKAPPA is a collection of tools to build, debug and run models of biological pathways. It contains a compiler for the Kappa Language [52], a static analyzer [51] (for debugging models), a simulator [50], a compression tool for causal traces [49],[20], and a model reduction tool [4], [48], [53].

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 pluggin 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 ETH Zürich, the UNAM-Genomics Mexico team. It is used as pedagocical 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].

Abstract interpretation, Certified compilation, Static analysis, Translation validation, Verifier.

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].

Arbitrary precision integers, Arithmetic, OCaml.

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

4. Software

4.1. 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://www.derric-lang.org

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: [8][32], [14]

In 2012 Derric was validated on a large body of image files taken from wikipedia, and a new approach to software optimization via model transformation was developed for optimizing Derric code. We released Derric 1.0 in 2012.

4.2. Basic Voting Theory

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

Characterization: A1, SO-3, SM-1, EM-1, SDL-2, OC-DA-3-CD-3-MS-3-TPM-3.

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

Objective: Demonstrate the basic concepts of voting theory.

Users: Students and researchers interested in voting theory.

Impact: This is a demonstrator and a tool for teaching.

Competition: None.

Engineering: Haskell program.

4.3. Rascal

Participants: Paul Klint, Jurgen Vinju [correspondent], Tijs van der Storm, Jeroen van den Bos, Mark Hills, Bert Lisser, Atze van der Ploeg, Vadim Zaytsev, Anastasia Izmaylova, Michael Steindorfer, Ali Afroozeh.

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

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: [23], [22], [11], [21], [22]

4.3.1. Novelties

- Statically typed access to external data-sources [21]. This includes access to CVS files, spreadheets, databases, etc.
- Significant improvements to online documentation and inter-active tutor environment.
- Full transparent support for Unicode codepoints.
- Added language-supported quickcheck-style random testing facility (by Wietse Venema, intern), including bridge to JUnit testing framework and IDE support.
- Revived access libraries to CVS, SVN and Git VCSs.
- Added support for JSON export and import, towards Rascal webservices.
- Totally re-implemented and extended debugging interface.
- Priority and associativity mechanism for context-free grammars was completed, such that it can not be used to accidentally remove sentences from a language anymore.
- Reimplementation of the except disambiguation filter with much higher efficiency.
- Improved module import times.
- Reimplemented URI encoding/decoding mechanism for correctness and portability.
- Added semi-automated exam generation and grading feature to the Rascal tutor environment.
- Experimented with strategies for error recovery in context-free general top-down parser.
- Added MissGrant and SuperAwesomeFighter language workbench demonstrations.
- Structured re-design of menus and menu options in the IDE

- Added bindings to Apache statistics libraries
- Created Rascalopedia, a glossary of concepts and terms that are relevant for metaprogrammers. The descriptions are aiming at under-graduate students.
- Two previously designed programmable transformation languages for grammars in a broad sense: the unidirectional XBGF and the bidirectional ΞBGF — have been reimplemented as libraries in Rascal.
- Improved general stability and efficiency.

4.4. IDE Meta-tooling Platform

Participants: Jurgen Vinju [correspondent], Michael Steindorfer.

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`. Currently we are moving the project to Github to explore more different ways of collaboration.

Publications: [3]

4.5. 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, and Alex Aloh, 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.

4.6. Turing language

Participants: Davy Landman [correspondent], Tijs van der Storm, Jeroen van den Bos, Vadim Zaytsev, Paul Klint.

Characterization: A3, SO-2, SM-1, EM-1, SDL-5, DA-3-CD-3-MS-3-TPM-3

WWW: http://www.legoturingmachine.org

Objective: This software is used to program the Lego Turing Machine which was built as a piece for the Turing Centennial exposition at CWI. The software features a full fledged Eclipse based IDE for a small programming language which is compiled to Turing machine instructions that run on a Lego machine.

Users: People interested in learning about computation and programming languages.

Impact: the Lego Turing Machine and it's software have reached more than 3 million people via the internet (slashdot, vimeo, youtube) and all Dutch national newspapers.

Competition: none.

Engineering: the hardware is Lego and the software is fully generated from Rascal syntax definitions and IDE construction functions.

4.7. Lua AiR

Participant: Riemer van Rozen [correspondent].

Characterization: A2-up, SO-4, SM-2-up, EM-3-up, SDL-4, DA-3-CD-3-MS-3-TPM-3

WWW: https://github.com/cwi-swat/Lua_AiR

Objective: This system provides IDE integrated support for static analysis of Lua code. Lua is a scripting language used in game development.

Users: Game programmers and game designers

Impact: Lua AiR is currently a research prototype designed to experiment with and validate the static analysis of a highly dynamic scripting language.

Competition: none.

Engineering: LuA AiR is fully implemented in Rascal.

CARTE Project-Team

5. Software

5.1. Morphus/MMDEX

MMDEX is a virus detector based on morphological analysis. It is composed of our own disassembler tool, on a graph transformer and a specific tree-automaton implementation. The tool is used in the EU-Fiware project and by some other partners (e.g. DAVFI project).
Written in C, 20k lines.
APP License, IDDN.FR.001.300033.000.R.P.2009.000.10000, 2009.

5.2. TraceSurfer

TraceSurfer is a self-modifying code analyzer coming with an IDA add-on. It works as a wave-builder. In the analysis of self-modifying programs, one basic task is indeed to separate parts of the code which are self-modifying into successive layers, called waves. TraceSurfer extracts waves from traces of program executions. Doing so drastically simplifies program verification.
Written in C, 5k lines.
Private licence.
http://code.google.com/p/tartetatintools/

5.3. CROCUS

CROCUS is a program interpretation synthetizer. Given a first order program (possibly written in OCAML), it outputs a quasi-interpretation based on max, addition and product. It is based on a random algorithm. The interpretation is actually a certificate for the program's complexity. Users are non academics (some artists).
Written in Java, 5k lines.
Private licence.

CASSIS Project-Team

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 [1] 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 [79]) 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 [68] for solving unification problems in disjoint unions of arbitrary theories.

CL-AtSe has been successfully used [67] 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 [82] for tool details and precise benchmarks).

Recently, Cl-Atse has been enhanced in various ways. As an official back-end for the Avantssar European Project, the tool's development followed the project's requirements for semantic and functionalities. In particular, the tool now fully supports the Aslan semantic, including support for Horn Clauses (for intruder-independent deductions, like e.g. management of credentials), improved support for LTL-based security properties, objects management w.r.t. a set semantic (instead of multiset by default), or smarter behavior in presence of ACM communication channels (default and preferred channel mode for Cl-Atse is CCM). While unofficial in Avantssar, the tracing option to target some specific traces during analysis has also been renewed w.r.t. the new modeling of transitions within the Aslan syntax. Also, tool support and bug corrections for all Avantssar's tools is now processed through a bugzilla server (see https://regis.scienze.univr.it/bugzilla/bugzilla-4.0.4/), and online analysis and orchestration are available on our team server (https://cassis.loria.fr). Then again, Cl-Atse now supports negative constraints on the intruder's knowledge. This support is correct and complete without algebraic operators (like Xor and Exp.), and implements in practice the assumptions and

[1] http://www.avispa-project.org

methods from [32]. This important improvement to the analysis algorithm in Cl-Atse allows us to find much more adequate orchestrations, and thus to reduce the orchestrator's processing times in a large scale. It was also used to model e.g. separation of duties.

5.2. Testing Tools

Participants: Fabrice Bouquet, Frédéric Dadeau, Philippe Paquelier, Kalou Cabrera.

5.2.1. Hydra

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 transferringthe existing work on test generation from B abstract machines, JML, and statecharts using constraint solving techniques.

5.2.2. jMuHLPSL

jMuHLPSL [9] is a mutant generator tool that takes as input a verified HLPSL protocol, and computes mutants of this protocol by applying systematic mutation operators on its contents. The mutated protocol then has to be analyzed by a dedicated protocol analysis tool (here, the AVISPA tool-set). Three verdicts may then arise. The protocol can still be *safe*, after the mutation, this means that the protocol is not sensitive to the realistic "fault" represented by the considered mutation. This information can be used to inform the protocol designers of the robustness of the protocol w.r.t. potential implementation choices, etc. The protocol can also become *incoherent*, meaning that the mutation introduced a functional failure that prevents the protocol from being executed entirely (one of the participants remains blocked in a given non-final state). The protocol can finally become *unsafe* when the mutation introduces a security flaw that can be exploited by an attacker. In this case, the AVISPA tool-set is able to compute an attack-trace, that represents a test case for the implementation of the protocol. If the attack can be replayed entirely, then the protocol is not safe. If the attack can not be replayed then the implementation does not contain the error introduced in the original protocol.

The tool is written in Java, and it is freely available at: http://disc.univ-fcomte.fr/home/~fdadeau/tools/jMuHLPSL.jar.

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 [2]. 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). This application aims to make a collaborative calendar where users can simultaneously modify events (or appointements) 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). We have also 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.

[2] http://www.sun.com/software/jxta/

CELTIQUE Project-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. Jacal

Participants: Frédéric Besson [correspondant], Thomas Jensen, David Pichardie, Delphine Demange, Vincent Monfort, Pierre Vittet.

Static program analysis, Javacard, Certification, AFSCM

`Jacal` is a JAvaCard AnaLyseur developed on top of the SAWJA4.2 platform. This proprietary software verifies automatically that Javacard programs conform with the security guidelines issued by the AFSCM (Association Française du Sans Contact Mobile). `Jacal` is based on the theory of abstract interpretation and combines several object-oriented and numeric analyses to automatically infer sophisticated invariants about the program behaviour. The result of the analysis is thereafter harvest to check that it is sufficient to ensure the desired security properties.

4.4. 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. The checker is now part of the Timbuk distribution. Timbuk distribution now also provide a CounterExample Guided Abstraction Refinement (CEGAR) tool for tree automata completion. The CEGAR part is based on the Buddy BDD library.

- Version: 3.1
- Programming language: Ocaml

COMETE Project-Team

5. Software

5.1. A model checker for the probabilistic asynchronous π-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 π-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 π-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 π-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/.

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 [39] and in the file README file width instructions at the URL http://www.lix.polytechnique.fr/comete/software/README-anonmodels.html.

The software can be download at 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 [40] and in the file README file width instructions at the URL http://www.lix.polytechnique.fr/comete/software/README-corners.html.

The software can be download at 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 π-calculus

Participants: Peng Wu [correspondant], Catuscia Palamidessi.

MMCsp is a compiler from a simple probabilistic π-calculus to PRISM. models. It is built on XSB, 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 2005-2007, in the context of the collaboration between the teams Comète and PRISM under the Inria/ARC Project ProNoBis. It is based on the papers [44] and [42].

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

CONTRAINTES Project-Team

5. Software

5.1. BIOCHAM, biochemical abstract machine

Participants: François Fages, Steven Gay, Sylvain Soliman.

The Biochemical Abstract Machine BIOCHAM [18] is a modeling environment for systems biology distributed as open-source since 2003. Current version is v3.4, released in October. 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 correspnding to three abstraction levels:

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

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 [5] 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 computation, or left aside, like unambiguous ODE to Petri net conversion, since a more general heuristic conversion has been developed for BIOCHAM [8], [19].

5.3. STSE, spatio-temporal simulation environment

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. YeastImageToolkit

Participant: Szymon Stoma.

YeastImageToolkit is an extention of YeastTracker software started originally by Jannis Uhlendorf. It allows following single cells in movies and quantifying fluorescent images based on this tracking as well as creating cell lineages. The software is currently under development and is designed to be a CellProfiler plugin facilitating yeast cell tracking, lineage and fluorescent signal quantification. Project webpage is: http://yeast-segtrack.weebly.com/.

5.5. SBMC, systems biology model-checker

Participant: Szymon Stoma.

Systems Biology Model Checker (SBMC) is a webservice allowing to verify biological models (e.g. signaling pathways stored in SBML files) against their specifications given in Signal Temporal Logics (STL). This project aims at providing to a large audience the methods described in [21] and used to analyse extrensic apoptosis pathway. Project webpage is: SBMC.

5.6. FO-CTL(R_{lin}), first-order computation tree logic over the reals

Participants: François Fages, Thierry Martinez.

FO-CTL(R_{lin}) is a solver for full First-Order Computation Tree Logic with linear arithmetic over the reals in constrained transition systems (CTS). CTS are transition systems where both states and transitions are described with constraints. FO-CTL(R_{lin}) generalizes the implementation done in Biocham of LTL(R_{lin}) for linear traces to branching Kripke structure.

5.7. Rules2CP

Participants: François Fages, Raphaël Martin, Thierry Martinez.

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.

5.8. SiLCC, linear concurrent constraint programming

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.

5.9. EMoP, existential modules for Prolog

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.

5.10. CHRat, CHR with ask and tell

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.

5.11. CLPGUI, constraint logic programming graphical user interface

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 developped 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.

DEDUCTEAM Team

4. Software

4.1. Dedukti

Dedukti is a proof-checker for the $\lambda\Pi$-calculus modulo. As it can be parametrized by an arbitrary set of rewrite rules, defining an equivalence relation, this calculus can express many different theories. Dedukti has been created for this purpose: to allow the interoperability of different theories.

Dedukti is designed to be versatile: it must be efficient on proofs that contain many computations—such as proofs by reflection—as well as proofs that do not contain any–such as proofs coming from HOL. These constraints has led us to adopt a Just-In-Time compilation architecture. And instead of designing our own JIT compiler, we have chosen to reuse the cutting-edge LuaJIT compiler. This technological choice, namely devolving the type-checking to Lua, makes Dedukti a proof-checker generator.

This has allowed the introduction of many optimizations: a normalization by evaluation strategy, a higher-order abstract syntax representation of terms and a context-free, bidirectional type-checking algorithm [22].

Dedukti has been developed by Mathieu Boespflug, Olivier Hermant, Quentin Carbonneaux, and Ronan Saillard.

4.2. CoqInE and HOLiDe

Dedukti comes with two companion tools: HOLiDe, an embedding of HOL proofs through the OpenTheory format [51], and CoqInE, an embedding of Coq proofs. Almost all the standard library of HOL and a significant part of that of Coq are checked by Dedukti.

CoqInE now supports the following features of Coq: the raw Calculus of Constructions, inductive types, and fixpoint definitions. It is now able to translate more than 80% of the standard library of Coq [21]. Ongoing work focuses on modules and functors, and on universes.

CoqInE has been developed by Mathieu Boespflug, Guillaume Burel, and Ali Assaf.

HOLiDe supports all the features of HOL, including polymorphism, constant definitions, and type definitions. It is able to translate all of the OpenTheory standard theory library.

HOLiDe has been developed by Ali Assaf.

4.3. iProver Modulo

iProver Modulo is an extension of the automated theorem prover iProver originally developed by Konstantin Korovin at the University of Manchester. It implements Ordered polarized resolution modulo, a refinement of the Resolution method based on Deduction modulo. It takes as input a proposition in predicate logic and a clausal rewriting system defining the theory in which the formula has to be proved. Normalization with respect to the term rewriting rules is performed very efficiently through translation into OCaml code, compilation and dynamic linking. Experiments have shown that Ordered polarized resolution modulo dramatically improves proof search compared to using raw axioms. iProver modulo is also able to produce proofs that can be checked by Dedukti, therefore improving confidence. iProver modulo is written in OCaml, it consists of 1,200 lines of code added to the original iProver.

It is developed by Guillaume Burel.

These four systems are available on the website of the team.

FORMES Team

5. Software

5.1. aCiNO

Participants: Fei He [correspondant], Min Zhou.

aCiNO is an SMT (Satisfiability Modulo Theory) solver based on a Nelson-Oppen [62] 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

Participants: Frédéric Blanqui [correspondant], Kim-Quyen Ly.

CoLoR is a Coq [42] library on rewriting theory and termination of more than 72,000 lines of code [4]. 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.

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

5.3. CoqMT

Participants: Qian Wang [correspondant], Jean-Pierre Jouannaud.

The proof-assistant Coq is based on a complex type theory, which resulted from various extensions of the Calculus of Constructions studied independently from each other. With the collaboration of Bruno Barras, we decided to address the challenge of proving the real type theory underlying Coq, and even, indeed, of its recent extension CoqMT developed in FORMES by Pierre-Yves Strub. To this end, we have studied formally the theory CoqMTU, which extends the pure Calculus of Constructions by inductive types, a predicative hierarchy of universes, and a decidable theory T for some first-order inductive types [1]. Recently, we were able to announce the complete certification of CoqMTU in Coq augmented with appropriate intuitionistic set-theoretic axioms in order to fight Gödel's incompleteness theorem, a work which has not been published yet. As a consequence, Coq and CoqMTU are the first proof assistants which consistency (relative to intuitionistic set theory IZF augmented with the afore-mentioned axioms) is formally entirely proved (in Coq). While previous formal proofs for Coq and other proof assistants all assumed strong normalization, the present one *proves* strong normalization thanks to the new notion of *strongly-normalizing* model introduced by Bruno Barras. While consistency is done already, decidability of type-checking remains to be done. This is a straightforward consequence for Coq, but a non-trivial task for CoqMTU because of the interaction between inductive types and the first-order theory T. It should however be announced around the turn of the year. We consider this work as a major scientific achievement of the team.

5.4. EDOLA

Participants: Hehua Zhang [correspondant], Ming Gu, Hui Kong.

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

EDOLA [72] is an integrated tool for domain-specific modeling and verification of PLC applications [70]. 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+ [56] 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 [56] and a first-order theorem prover SPASS [71].

5.5. HOT

Participant: Frédéric Blanqui [correspondant].

HOT is an automated termination prover for higher-order rewrite systems based on the notion of computability closure and size annotation [13]. It won the 2012 competition in the category "higher-order rewriting union beta". The sources are not public.

5.6. Moca

Participant: Frédéric Blanqui [correspondant].

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

Moca is a construction functions generator for OCaml [57] 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/.

5.7. Rainbow

Participants: Frédéric Blanqui [correspondant], Kim-Quyen Ly.

Rainbow is a tool for verifying the correctness of termination certificates expressed in the CPF XML format as used in the termination competition. Termination certificates are currently translated and checked in Coq by using the CoLoR library. But a new standalone version is under development using Coq extraction mechanism.

Rainbow is distributed under the CeCILL license on http://color.inria.fr/rainbow.html. See the website for more information.

5.8. 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 [52], [25]. 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:

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

5.9. 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 [22] (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 [24] but this is still experimental (work done by Frédéric Tuong in 2011).

Finally, we started to prove that the C code for simulating ARM instructions in Simlight is correct with respect to the Coq model.

GALLIUM Project-Team

5. Software

5.1. OCaml

Participants: Damien Doligez [correspondant], Xavier Clerc [team SED], Alain Frisch [LexiFi], Jacques Garrigue [Nagoya University], Thomas Gazagnaire [OCamlPro], Fabrice Le Fessant [Inria Saclay and OCamlPro], Xavier Leroy, Luc Maranget [EPI Moscova].

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], Jacques-Henri Jourdan, Valentin Robert.

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 `long long` and `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).

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

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 retargeted to output scripts for different frameworks (for example, Isabelle).

Web site: http://zenon-prover.org/

MARELLE Project-Team

4. Software

4.1. Tralics

Participant: José Grimm [correspondant].

Tralics is a Latex-to-XML translator available at http://www-sop.inria.fr/marelle/tralics. Version 2.15 has been released this year. Some features have been added, and some bugs corrected.

4.2. 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

4.3. Certicrypt and Easycrypt

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 [correspondant].

`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 and 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 [correspondant], 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, i.e. Petri nets with read arcs [42],[48]. 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 and produce a smaller unfolding. Cunf has recently been extended with a verification component that takes advantage of these features [70]. More details can be found at http://www.lsv.ens-cachan.fr/~rodrigue/tools/cunf/. Moreover, Cunf has been integrated into the CosyVerif environment (see section 5.1.4).

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

Participants: Hilal Djafri, Benoît Barbot [correspondant].

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; recently, it has been equipped with functionalities for rare event analysis. 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/`

5.1.4. *COSYVERIF*

Participants: Serge Haddad [correspondant ?], Benoît Barbot.

CosyVerif is a software environment whose goal is the formal specification and verification of dynamic systems.

It has been designed in order to:

- support different formalisms with the ability to easily create new ones, - provide a graphical interface for every formalism, - include verification tools called via the interface as a web service, - offer the possibility for a developer to integrate his/her own tool, also allowing it to interact with the other tools.

This environment consists of two software tools: Coloane, the graphical interface, and Alligator, an integration framework based web services. It is enlarged with the existing verification tools developed in our laboratories (founding members or partners). Why ?

The development of Cosyverif has been decided and it is supported by three partners of the Parisian verification group, MeFoSyLoMa. This group is composed of seven teams. and the founding members of are LIP6, LIPN and LSV. First, these members aim at sharing their tools, comparing and supporting industrial case studies and finally making them long-lasting. Second, they also want to promote the practice of formal verification in industry and thus they intend to ease the task of integration of new formalisms and tools.

It is managed by a steering committee consisting of researchers and engineers. It decides strategic orientations as well as technical choices. Current Tools

Two formalisms are supported: automata and Petri nets, both with extensions. Most of the tools are related to Petri nets. Some of them perform structural analyses like invariant computations. while other tools perform behavioural analyses: symbolic reachability graph building, unfolding, stochastic simulations, etc. Finally some of them transform high-level nets into low-level ones. All the developed software are open source and free software tools. Alligator is published under the GNU Affero General Public License (AGPL) version 3 ; Coloane is published under the Eclipse Public License (EPL) version 1 .

Three engineers have worked or are currently working on COSYVERIF:

- Francis Hulin-Hubard, part-time (CNRS) in 2012;
- Clément Desmoulins , full-time (ANR), 6 months; and
- Alban Linard, full-time Inria engineer, for 2 years.

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, Christophe Calvès, Horatiu Cirstea, Pierre-Etienne Moreau [correspondant], Claudia Tavares.

Since 2002, we have developed a new system called *Tom* [33], presented in [17], [18]. 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 experiments on the efficient compilation of rule-based systems [30]. 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 algorithms and rewriting rules for term-graphs.

Tom is documented, maintained, and available at http://tom.loria.fr as well as at http://gforge.inria.fr/projects/tom.

PARSIFAL Project-Team

5. Software

5.1. Abella

Participants: Kaustuv Chaudhuri [correspondant], Matteo Cimini, Dale Miller.

Abella is an interactive theorem prover based on the two-level logic logic approach. It consists of a sophisticated reasoning logic that supports induction, co-induction, and generic reasoning, and a specification logic that is based on logic programming. Abella was initially designed to reason about simple second-order Lambda Prolog programs, which is sufficient for the computational specifications.

During 2012, as part of the RAPT Associated Team, Chaudhuri and Yuting Wang (intern from Univ. Minnesota) have been working on extending the expressive power of both levels of the Abella system. The following modifications have been made.

- We have extended the specification logic to support the full Lambda Prolog, which can be used to provide succinct higher-order specifications that tend to be unnatural and difficult to reason about with only second-order Lambda Prolog programs.
- We have extended the type system of Abella from simple types to parametrically polymorphic types. This is a significant improvement in the user-friendliness of the system as a lot of code does not have to be manually monomorphised and duplicated any more.
- We have experimented with extending the type system of Abella even further to higher-order predicate quantification. The theoretical basis of this work is a part of ongoing research, although we already have a number of examples of practical benefits of this extension.
- Finally, several improvements have been made to Abella's proof language to make the proofs more robust and reusable. We intend to make a more drastic change to the proof language in the future that will make proofs more declarative and high level.

The core development of Abella has also been centralized, with a single canonical repository and a new webpage: http://abella-prover.org. These resources are managed jointly by members of Parsifal and our colleagues at the University of Minnesota.

The next version of Abella, version 2.0, is in beta testing with expected release early in 2013.

5.2. Bedwyr

Participants: Quentin Heath, Dale Miller [correspondant].

During 2012, Quentin Heath has made the following important improvements to the Bedwyr model checking system.

- The concrete syntaxs for Bedwyr and Abella have been unified. Now, both systems can load the definitions and theorems developed in the other system. Eventually, we expect to have our model checker (Bedwyr) and interactive theorem prover (Abella) share theories and proofs.
- The documentation, distribution, and testing of Bedwyr were all improved, greatly increasing the usability of this system.
- The underlying support for logic has also been increased. In particular, the Bedwyr system contains a *tabling* mechanism which is capable of remembering past successful proofs (it can even support a finite failure as a successful proof of a negation). The most recent version of Bedwyr allows one to actually program the table in rather sophisticated ways. For example, simple lemmas can be loaded into the table and these lemmas can be used to greatly extend the range of what is tabled (remembered). We are currently examining different trade-offs between different styles of reasoning in the table (backchaining vs forwardchaining).

The work of Heath is being done in the context of the BATT ADJ project funded by Inria.

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

5.3. Psyche

Participants: Mahfuza Farooque, Stéphane Graham-Lengrand [correspondant].

Psyche (*Proof-Search factorY for Collaborative HEuristics*) is a modular programme for universal proof-search in classical logic. The motivation is twofold:

On the one hand, prove some mathematics of the broadest range while making the most of problem-specific techniques; On the other hand, gain high confidence about the correctness of the proofs produced without having to rely on a proof-checker.

The architecture is that of an interaction between a trusted universal kernel and smart plugins that are meant be efficient at solving certain kinds of problems:

The kernel contains the mechanisms for exploring the proof-search space in a sound and complete way, taking into account branching and backtracking. The output of Psyche comes from the (trusted) kernel and is therefore correct by construction. The plugins then drive the kernel by specifying how the branches of the search space should be explored, depending on the kind of problem that is being treated. The quality of the plugin is then measured by how fast it drives the kernel towards the final answer.

Version 1.0 of Psyche (released 4/9/2012) handles classical propositional logic, and its proof-search mechanism is simply the incremental construction of proof-trees in the polarised and focussed sequent calculus. The mechanism is driven by a plugin that emulates the behaviour of a SAT-solver (DPLL), with non-trivial features such as the eager application of the Unit Propagation rule, conflict analysis, backjumping and clause learning.

Psyche's input for that kind of SAT-problem is a file given in the standard DIMACS format.

See also the web page http://www.lix.polytechnique.fr/~lengrand/PSI/index.php?page=Psyche/index.

PI.R2 Project-Team

5. Software

5.1. http://coq.inria.frCOQ

Participants: Bruno Barras [TypiCal team, Saclay], Yves Bertot [Marelle team, Sophia], Pierre Boutillier, Xavier Clerc [SED team], Pierre Courtieu [CNAM], Maxime Dénès [Marelle team, Sophia], Julien Forest [CNAM], Stéphane Glondu [CARAMEL team, Nancy Grand Est], Benjamin Grégoire [Marelle team, Sophia], Vincent Gross [Consultant at NBS Systems], Hugo Herbelin [correspondant], Pierre Letouzey, Assia Mahboubi [TypiCal team, Saclay], Julien Narboux [University of Strasbourg], Jean-Marc Notin [TypiCal team, Saclay], Christine Paulin [Proval team, Saclay], Pierre-Marie Pédrot, Loïc Pottier [Marelle team, Sophia], Matthias Puech, Yann Régis-Gianas, François Ripault, Matthieu Sozeau, Arnaud Spiwack, Pierre-Yves Strub [Formes team, Beijing], Enrico Tassi [TypiCal team, Saclay], Benjamin Werner [TypiCal team, Saclay].

5.1.1. Version 8.4

Version 8.4 was released in August 2012. It introduced a new proof engine designed and implemented by Arnaud Spiwack and a new extensive modular library of arithmetic contributed by Pierre Letouzey. It also included an extension of the underlying logic with "η-conversion" by Hugo Herbelin and "commutative-cuts compliant guard condition" by Pierre Boutillier, 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 Boutillier 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 finalised and extended 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.

In Fall 2012, Pierre Letouzey also revised the event infrastructure of CoqIDE, from a thread-based model to pure GTK event-loop. This way, CoqIDE is more reactive, less subject to deadlocks (especially under Windows), and the source code is more idiomatic and easier to understand. Interestingly, this work takes advantage of deeper notions such as C.P.S. (continuation passing style).

Pierre Boutillier and Pierre-Marie Pédrot built an abstract communication interface between Coq and CoqIDE based on XML syntax. They also refined the ability to customise CoqIDE. Pierre Boutillier made CoqIDE rely on Gtksourceview.

5.1.3. Proof engine

Arnaud Spiwack has proposed an extension of the expressiveness of tactics based on his previous work for a new proof engine. It allows for more atomic tactics, has a primitive support for backtracking, and allows for tactics which manipulate several goals.

5.1.4. Evaluation algorithms

Pierre Boutillier has proposed a new unfolding algorithm for global constants so that the definition of these ones are unfolded only if it triggers extra reductions. This helps users to keep goals concise during interactive proofs.

5.1.5. Type classes, internal representation

Matthieu Sozeau is adapting the type-classes mechanism to benefit from the new tactic engine and avoid reimplementing a whole proof-search algorithm with backtracking on top of the tactic language. This will bring high benefits in terms of efficiency and ease of use to the users. Forward proof-search for type classes was stabilised and is now used in libraries for better control on the search space, notably in the MathClasses library developed in Nijmegen [69].

An important shortcoming of type classes is the verbosity of the representation of projections from a class, as was illustrated in François Garillot's PhD thesis [48]. Matthieu Sozeau has developed a branch of Coq supporting an efficient representation of these projections based on the idea of bidirectional type checking which is now under stabilisation. This support will also enhance the performance of the assistant on developments using regular parameterised records and dependent sums like the HoTT library on homotopy type theory and the Forcing plugin developed by Sozeau *et al* [32].

5.1.6. Universes

While visiting the Institute for Advanced Study, Matthieu Sozeau implemented a new system of universe polymorphism that makes it possible to develop highly generic theories in the Coq system. Based on ideas from Harper and Pollack's [53] design of polymorphism as an elaboration in the Lego theorem prover, he developed an original algorithm for type inference of universes and implemented it in Coq. Its first application is inside the Homotopy Type Theory (HoTT) research program, as the formalisation of HoTT requires a high level of polymorphism that was not available before. Many other theories will benefit from this, including Sozeau's work on Forcing, B. Barras' (Typical) work on models of type theory or in the Math Classes library mentioned before. It also opens up possibilities to formalise category-theoretic notions without being limited by the universe system, a long-standing barrier in the Coq proof assistant.

5.1.7. The Equations plugin

Matthieu Sozeau continued the maintenance of the Equations plugin and developed a new Forcing plugin for Coq (see below).

5.1.8. Tools

Pierre-Marie Pédrot has written a program using the internal representation of libraries to compact Coq object files. It is based on well-known automata algorithms, representing memory as transition systems. The idea underlying this program is generalisable to any OCaml data structure, provided some conditions on its use are satisfied, and was formalised in a paper that was accepted at JFLA 2013.

5.1.9. Internal architecture of the Coq software

Pierre Letouzey continued a large reorganisation of the internal components of Coq, since these components are currently too much interdependent. This work brought better isolation between some of the Coq components and explicit interfaces between them. This allowed to simplify the compilation of Coq, since it is now easier to build the OCaml syntax extension used when compiling many advanced parts of Coq. Moreover, this clearer architecture should also help new contributors when they discover and interact with this large and complex code-base.

Pierre-Marie Pédrot also made some reorganisations of the code. This includes a clean generic library superseding the one of OCaml, pushing the CAMLP4/5 dependent parts out of the lower strata, as well as benefiting from the OCaml module system to get more uniformity in the naming of usual data structures.

Pierre Letouzey proposed a nicer backtracking infrastructure to Coq, used when the user wants to cancel some recent commands and go back before them. This new infrastructure unifies and improves what was used earlier by ProofGeneral and CoqIDE, the two main user interfaces for Coq.

Pierre Letouzey also dedicated many efforts to improve the support of the Windows platform by Coq.

5.1.10. Efficiency

Pierre-Marie Pédrot has been trying to optimise various parts of the Coq system, including the new tactical system designed by Arnaud Spiwack. Some neat tricks on garbage collection permitted to reach a substantial time improvement in compilation of object files. Various architectural modifications were also made in the process, like trying to get rid of the generic comparison in the code base.

Pierre Letouzey investigated an alternative implementation of the code dealing with Coq universes. These universes are a critical part of Coq: they have direct consequences on Coq safety, and handling them is time-consuming (between 10% to 20% depending on the Coq usage). This alternative implementation looks promising, but still requires some more work and stress-tests before being integrated in mainstream Coq.

5.1.11. Documentation generation

François Ripault and Yann Régis-Gianas developed a new version of coqdoc, the documentation generator of Coq. This new implementation is based on the interaction protocol with the Coq system and should be more robust with respect to the evolution of Coq.

5.1.12. General maintenance

Pierre Letouzey has been the main maintainer of Coq with extra contributions from Hugo Herbelin, Pierre Boutillier, Matthieu Sozeau, Pierre-Marie Pédrot, ...

5.1.13. Development Action

An “Action de Développement Technologique” about Coq started September 2011 and continued this year. It gathers the πr^2 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.

Yann Régis-Gianas set up an “osqa” server for Frequently Asked Questions.

The ADT Coq supported the internship of François Ripault.

Hugo Herbelin formalised a type-theoretic construction of semi-simplicial sets answering a problem raised early this year by Steve Awodey, Peter LeFanu Lumsdaine and others, in relation with the homotopy models of type theory.

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 [19].

5.3. Other software developments

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

PROSECCO Project-Team

5. Software

5.1. ProVerif

Participants: Bruno Blanchet [correspondant], Xavier Allamigeon [April–July 2004], Vincent Cheval [Sept. 2011–], Ben Smyth [Sept. 2009–Feb. 2010].

PROVERIF (proverif.inria.fr) is an automatic security protocol verifier in the symbolic 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, specified 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.

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 is widely used by the research community on the verification of security protocols (see http://proverif.inria.fr/proverif-users.html for references).

PROVERIF is freely available on the web, at proverif.inria.fr, under the GPL license.

5.2. CryptoVerif

Participants: Bruno Blanchet [correspondant], David Cadé [Sept. 2009–].

CRYPTOVERIF(cryptoverif.inria.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 and correspondences, which include in particular authentication. It 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, and Diffie-Hellman key agreements.

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 has been used in particular for a study of Kerberos in the computational model, and as a back-end for verifying implementations of protocols in F# and C.

CRYPTOVERIF is freely available on the web, at cryptoverif.inria.fr, under the CeCILL license.

5.3. Tookan

Participants: Graham Steel [correspondant], Romain Bardou.

See also the web page http://tookan.gforge.inria.fr/.

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 [47] and a six-month licence was granted to Boeing to use the tool. In 2011, a contract was signed with a major UK bank. Tookan is now the subject of a CSATT transfer action resulting in the hiring of an engineer, Romain Bardou, who started on September 1st, 2011. During 2012 Bardou and Steel implemented a new version of Tookan that is intended to form the technological basis for a spin-off company to be created in 2013.

5.4. miTLS

Participants: Alfredo Pironti [correspondant], Karthikeyan Bhargavan, Cedric Fournet [Microsoft Research], Pierre-Yves Strub [IMDEA], Markulf Kohlweiss [Microsoft Research].

miTLS is a verified reference implementation of the TLS security protocol in F#, a dialect of OCaml for the .NET platform. It supports SSL version 3.0 and TLS versions 1.0-1.2 and interoperates with mainstream web browsers and servers. miTLS has been verified for functional correctness and cryptographic security using the refinement typechecker F7.

A paper describing the miTLS library is under review, and the software is being prepared for imminent release in January 2013.

5.5. WebSpi

Participants: Karthikeyan Bhargavan [correspondant], Sergio Maffeis [Imperial College London], Chetan Bansal [BITS Pilani-Goa], Antoine Delignat-Lavaud.

WebSpi is a library that aims to make it easy to develop models of web security mechanisms and protocols and verify them using ProVerif. It captures common modeling idioms (such as principals and dynamic compromise) and defines a customizable attacker model using a set of flags. It defines an attacker API that is designed to make it easy to extract concrete attacks from ProVerif counterexamples.

WebSpi has been used to analyze social sign-on and social sharing services offered by prominent social networks, such as Facebook, Twitter, and Google, on the basis of new open standards such as the OAuth 2.0 authorization protocol.

WebSpi has also been used to investigate the security of a number of cryptographi web applications, including password managers, cloud storage providers, an e-voting website and a conference management system.

WebSpi is under development and released as an open source library at http://prosecco.inria.fr/webspi/

5.6. Defensive JavaScript

Participants: Antoine Delignat-Lavaud [correspondant], Karthikeyan Bhargavan, Sergio Maffeis [Imperial College London].

Defensive JavaScript (DJS) is a subset of the JavaScript language that guarantees the behaviour of trusted scripts when loaded in an untrusted web page. Code in this subset runs independently of the rest of the JavaScript environment. When propertly wrapped, DJS code can run safely on untrusted pages and keep secrets such as decryption keys. DJS is especially useful to write security APIs that can be loaded in untrusted pages, for instance an OAuth library such as the one used by "Login with Facebook". It is also useful to write secure host-proof web applications, and more generally for cryptography that happens on the browser.

The DJS type checker and various libraries written in DJS are available from http://www.defensivejs.com.

SECSI Project-Team

5. Software

5.1. Tookan

Participants: Graham Steel [correspondant], Romain Bardou.

See also the web page http://tookan.gforge.inria.fr/.

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 [48] and a six-month licence was granted to Boeing to use the tool. In 2011, a contract was signed with a major UK bank. Tookan is now the subject of a CSATT transfer action resulting in the hiring of an engineer, Romain Bardou, who started on September 1st, 2011. During 2012 Bardou and Steel implemented a new version of Tookan that is intended to form the technological basis for a spin-off company to be created in 2013. As a result of the transfer of Graham Steel and Romain Bardou to team Prosecco, this project is being continued in that team.

5.2. Orchids

Participants: Jean Goubault-Larrecq [correspondant], Hedi Benzina, 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.

Nasr-Eddine Yousfi followed up on Baptiste Gourdin, starting from December 2011, on an ITI engineer position allotted by Inria's CSATT. He mostly explored ways of writing security meta-policies for confidentiality of sensitive dat.

Orchids will be the core of a contract between Inria and DGA, to be signed in December 2012, for three years.

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], Thierry Petit, Charles Prud'homme [correspondant], Remi Douence.

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. Since September 2011, we are working on a new version of the CHOCO solver. This implies a total refactoring of the source code in order to make it simpler to use and maintain. We introduce a new propagation engine framework that directly handle state-of-the-art techniques, such as *advisors*, *propagator groups*, *activity-based search* and *explanations*, to ensure a good level of efficiency, and plug a MiniZinc modeling language parser. An alpha release will be available by the beginning of 2013.
2. In the context of the new version of the CHOCO solver we design an *adaptive propagation engine* to enhance performance as well as a *solver independent language to write strategies* for controlling the new adaptive propagation engine. The adaptive propagation engine can both deal with variable-oriented propagation engines and constraint-oriented propagation engines. It is usually accepted that there is no best approach in general and modern constraint solvers therefore implement only one.
3. New scalable global constraints were provides both in the context of *graph constraints* (with also graph variables) and in the context of *scheduling constraints*. These constraints respectively allow to handle sparse graphs with up to 10000 vertices, and resource scheduling problems with up to one million tasks.
4. A new global constraint called *focus* for concentrating high cost values motivated by several concrete examples, such as resource constrained scheduling problems with machine rentals, was introduced.
5. The work on providing probability-based constraints to get light propagation filtering algorithm has been pursued. A particular focus has been put on calculating the probabilistic indicator for the bound-consistency propagator of an *alldifferent* constraint.

N. Beldiceanu, A. De Clerq, S. Demassey, J.-G. Fages, N. Jussien, A. Letort, X. Lorca, T. Petit, C. Prud'Homme and R. Douence have contributed in 2012. The link to the system and documentation is http://choco.emn.fr.

5.2. IBEX

Participants: Ignacio Araya, Anthony Baire, Gilles Chabert [correspondant], Rémi Douence, Bertrand Neveu, Gilles Trombettoni.

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.

Continuing last year work on the redesign of the architecture of IBEX, the IBEX library has been entirely refactored from scratch to provide a more clean and easy-to-use interface as well as a more powerful engine.The development started in late 2011, the kernel has been completed in mid-2012 and almost all the functionalities of IBEX integrated in the new architecture. Global optimization and system solving front-end algorithms have been tested on more than 500 benchmarks. Installation scripts for a deployment on multiple platforms have also been done by an engineer of Inria (Anthony Baire). A first web site has been activated, with an on-line installation documentation, a programming tutorial (still under writing), and an API. An alpha release is now available for download. A first training course on IBEX 2.0 has been organized at ENSTA Bretagne, Brest, the 17-18th December with about 25 participants. Similar training courses will also occur in 2013.

An explorative study aimed at showing that the explicit representation of search trees can play a distinguished role in the field of numerical constraints was done this year. The idea was also to define a new high-level language to handle explicit search trees, in the fashion of quadtrees (that one can intersects, etc.). We have developed a prototype in Haskell to validate the approach and have illustrated it over different examples.

5.3. CHOCO-IBEX

Participants: Gilles Chabert [correspondant], Charles Prud'homme [correspondant].

Work has been done to provide an interface for connecting the CHOCO and the IBEX libraries in order to handle problems where we both have continuous and discrete variables. This interface allows to filter continuous domains from CHOCO with the IBEX engine as well as to check for unsatisfiability or entailment. It also manages reification variables. This interface has been tested on a toy problem and seems to work as expected. Some glue code (on both sides) is still missing to handle reificiation and should be integrated in a short term. The interface should be ready for usage with the next version of CHOCO (3.0).

5.4. Global Constraint Catalog

Participants: Nicolas Beldiceanu [correspondant], 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 406 constraints, 3397 pages and 758 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. making the *core global constraints* (10 constraints) more accessible to a wider audience:
   - for this purpose examples with their corresponding pictures have been systematically provided for showing all solutions for an example of each core global constraint.
   - in addition a set of about 30 exercises with their corrections have been done for half of the core global constraints.
3. a redesign of all the 758 figures of the catalog has been undertaken in autumn 2012 using TikZ (in December 2012 312 figures were redesigned).
4. adding constraints related to sequences that we found relevant for learning constraints from electricity production curves.

N. Beldiceanu, M. Carlsson (SICS, Sweden) and H. Simonis (4C, Ireland) have contributed in 2012. The link to the global constraint catalog is http://www.emn.fr/z-info/sdemasse/gccat/.

TOCCATA Team

5. Software

5.1. The CiME rewrite toolbox

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

Keywords: Equational reasoning, Rewriting, Termination, Confluence, Completion

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/MTT/) 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], François Bobot, Jean-Christophe Filliâtre, Guillaume Melquiond, Andrei Paskevich.

Keywords: Deductive verification, Java programming language, Java modeling language, Java Card, ANSI C programming language.

Criteria for Software Self-Assessment [5]: A-3, SO-4, SM-3, EM-2, SDL-5-down, OC-4.

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 [86], at Universidade do Minho in Portugal [49], at Moscow State University, Russia (http://journal.ub.tu-berlin.de/eceasst/article/view/255).

5.3. The Why3 system

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

Keywords: Deductive verification

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

[5] self-evaluation following the guidelines (http://www.inria.fr/content/download/11783/409665/version/4/file/SoftwareCriteria-V2-CE.pdf) of the Software Working Group of Inria Evaluation Committee(http://www.inria.fr/institut/organisation/instances/commission-d-evaluation)

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 used for teaching (Master Parisien de Recherche en Informatique).

Why3 is used by other academic research groups, e.g. within the CertiCrypt/EasyCrypt project (http://easycrypt.gforge.inria.fr/) for certifying cryptographic programs.

5.4. The Alt-Ergo theorem prover

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

Keywords: Automated theorem proving, Combination of decision procedures, Satisfiability modulo theories

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 [103] 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 *Why3* 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. The Cubicle model checker modulo theories

Participants: Sylvain Conchon [contact], Alain Mebsout.

Partners: A. Goel, S. Krstić (Intel Strategic Cad Labs in Hillsboro, OR, USA), F. Zaïdi (LRI, Université Paris-sud)

Keywords: Satisfiability modulo theories, model checking, array-based systems

Cubicle is an open source model checker for verifying safety properties of array-based systems. This is a syntactically restricted class of parametrized transition systems with states represented as arrays indexed by an arbitrary number of processes. Cache coherence protocols and mutual exclusion algorithms are typical examples of such systems.

Cubicle model-checks by a symbolic backward reachability analysis on infinite sets of states represented by specific simple formulas, called cubes. Cubicle is based on ideas introduced by MCMT (http://users.mat.unimi.it/users/ghilardi/mcmt/) from which, in addition to revealing the implementation details, it differs in a more friendly input language and a concurrent architecture. Cubicle is written in OCaml. Its SMT solver is a tightly integrated, lightweight and enhanced version of *Alt-Ergo*; and its parallel implementation relies on the Functory library.

5.6. Bibtex2html

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

Keywords: Bibliography, Bibtex format, HTML, World Wide Web.

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

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.

5.7. OCamlgraph

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

Keywords: Graph, Library, *OCaml*.

OCamlgraph is a graph library for *OCaml*. 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.

5.8. Mlpost

Participants: Jean-Christophe Filliâtre [contact], François Bobot.

Keywords: Library, *OCaml*.

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 *OCaml*. 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].

5.9. Functory

Participant: Jean-Christophe Filliâtre [contact].

Keywords: Library, *OCaml*.

Functory is a distributed computing library for *OCaml*. 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 [84] and at TFP 2011 [83].

5.10. The Pff library

Participant: Sylvie Boldo [contact].

Keywords: Interactive theorem proving, floating-point arithmetic. Criteria for Software Self-Assessment: A-2, SO-3, SM-3, EM-3, SDL-5, OC-4.

The Pff library for the *Coq* proof assistant is a formalization of floating-point arithmetic with high-level definitions and high-level properties [58].

It is distributed as open source, under a LGPL license, at http://lipforge.ens-lyon.fr/www/pff/, and is packaged in Debian and Ubuntu as "coq-float".

It was initiated by M. Daumas, L. Rideau and L. Théry in 2001, and then developed and maintained by S. Boldo since 2004. It is now only maintained by S. Boldo. The development has ended as this library is now subsumed by the Flocq library (see below).

5.11. The Flocq library

Participants: Sylvie Boldo [contact], Guillaume Melquiond.

Keywords: Interactive theorem proving, floating-point arithmetic.

Criteria for Software Self-Assessment: A-2, SO-3, SM-3, EM-3, SDL-4, OC-4.

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 [5]. 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.

5.12. The Gappa tool

Participant: Guillaume Melquiond [contact].

Keywords: Automated theorem proving, floating-point arithmetic, fixed-point arithmetic.

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

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 [79] [109].

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.

5.13. The Interval package for Coq

Participant: Guillaume Melquiond [contact].

Keywords: Interactive theorem proving, interval arithmetic, floating-point arithmetic.

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

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 library 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.

5.14. The Alea library for randomized algorithms

Participants: Christine Paulin-Mohring [contact], Pierre Courtieu.

Keywords: Interactive theorem proving, randomized algorithms, probability

Criteria for Software Self-Assessment: A-2, SO-3, SM-2, EM-3, SDL-4, OC-4.

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 [54]. It is also experimented in LABRI as a basis to study formal proofs of probabilistic distributed algorithms.

5.15. The Coccinelle library for term rewriting

Participant: Évelyne Contejean [contact].

Keywords: Interactive theorem proving, Coq, rewriting, termination certificate 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 a 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 developement (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.

5.16. The Coquelicot library for real analysis

Participants: Sylvie Boldo [contact], Catherine Lelay, Guillaume Melquiond.

Keywords: Interactive theorem proving, real analysis

Criteria for Software Self-Assessment: A-2, SO-4, SM-2, EM-3, SDL-1, OC-4.

Coquelicot is a *Coq* library dedicated to real analysis: differentiation, integration, and so on. It is a conservative extension of the standard library of *Coq*, but with a strong focus on usability.

Coquelicot is available at http://coquelicot.saclay.inria.fr/.

5.17. CFML

Participant: Arthur Charguéraud [contact].

Keywords: Program verification, Interactive theorem proving, *OCaml*

Criteria for Software Self-Assessment: A-2, SO-4, SM-2, EM-3, SDL-1, OC-4. The *CFML* tool supports the verification of *OCaml* programs through interactive *Coq* proofs. The tool is made of two parts: on the one hand, a characteristic formula generator implemented as a *OCaml* program that parses *OCaml* code and produces Coq formulae; and, on the other hand, a Coq library that provides notation and tactics for manipulating characteristic formulae interactively in Coq.

CFML is distributed under the LGPL license, and is available at http://arthur.chargueraud.org/softs/cfml/. The tool has been initially developed by A. Charguéraud in 2010, and has been maintained and improved since by the author.

TYPICAL Project-Team

4. Software

4.1. Coq

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

Coq is a major proof system an the primary object and / or tool of our research. Its development is now mainly coordinated by the πr^2 Inria Paris-Rocquencourt project-team, and some members of the TypiCal team are active developers of the system.

4.2. Coqfinitgroup

Participants: Cyril Cohen, Assia Mahboubi [Contact], Enrico Tassi.

Coqfinitegroup is the development corresponding to the full formalization of 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 have been actively participating to this long term formalization project.

4.3. 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. Enrico Tassi has worked on an extended language of patterns for term selection now included in the distribution of this extension. Members of the Typical are in charge of the documentation and distribution of this extension.

VERIDIS Project-Team

5. Software

5.1. The veriT solver

Participants: Rodrigo Castaño, David Déharbe, Pablo Federico Dobal, Pascal Fontaine [correspondent].

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.

Efforts in 2012 have been focused on efficiency, with various improvements and the redesign of the core solver. A preliminary prototype integrating Redlog for handling non-linear arithmetic showed encouraging results. Short term future works include improving the design, adding full support for non-linear arithmetic, and increasing 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. In 2012, we presented at ABZ [16] a plugin for Rodin using SMT solvers (and notably veriT) to discharge B proof obligations: on a large repository of industrial and academic cases, this SMT-based plugin decreased by 75% the number of proof obligations requiring human interactions, compared to the original B prover. See also section 8.1 for our work within the DeCert project.

For helping development within and around veriT, Pablo Federico Dobal has been hired for two years starting September 2012 as a young engineer supported by the Inria ADT program.

5.2. The TLA+ proof system

Participants: Stephan Merz [correspondent], 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 publically 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$^+$ 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, and a backend for interfacing with SMT solvers. The SMT backend has been improved significantly in 2012 and is now considered by users as the most useful backend prover for system verification. Version 1.0 of TLAPS was released in January 2012, followed by version 1.1 in November, and the system was presented at the conference FM 2012 [15].