RESEARCH CENTER

FIELD
Algorithmics, Programming, Software and Architecture

Activity Report 2013

Section Software

Edition: 2014-03-20

ARIC Project-Team

5. Software and Platforms

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 2013.

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 [32]. FloPoCo is a generator of operators written in C++ and outputting synthesizable VHDL automatically pipelined to an arbitrary frequency.

2013 saw more work on the *bit-heap* framework [28], [18]. In addition, several new operators were added, in particular for fixed-point sine, cosine [21] and arctangent.

Version 2.5.0 was released in 2013.

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. Tohoku, U. Tokyo, Virginia Tech U. and several companies.

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

- Version: 2.5.0 (June 2013)
- APP: IDDN.FR.001.400014.000.S.C.2010.000.20600 (version 2.0.0)
- License: pending, should be 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.

GNU MPFR 3.1.2 was released on 13 March 2013.

The main work done in the AriC project-team:

- Bug fixes and improved portability.
- Complete revision of the behavior on special values (signed zeros and infinities) and consistency with standards (IEEE 754-2008, ISO C, POSIX) checked. Thanks to this work, several problems in MPFR and the POSIX specification have been detected and the MPFR manual has been completed: https://sympa.inria.fr/sympa/arc/mpfr/2013-12/msg00001.html

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 and some minor bugs have been fixed.

5.5. FPLLL: A Lattice Reduction Library

Participant: 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 4.0.4 was released in 2013, fixing a number of user-interface bugs.

URL: http://perso.ens-lyon.fr/damien.stehle/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://perso.ens-lyon.fr/damien.stehle/fplll/fplll-doc.html

5.6. Sipe

Participant: Vincent Lefèvre.

Sipe is a mini-library in the form of a C header file, to perform radix-2 floating-point computations in very low precisions with correct rounding, either to nearest or toward zero. 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 addition, subtraction, multiplication (possibly with the error term), fused multiply-add/subtract (FMA/FMS), and miscellaneous comparisons and conversions. Sipe provides two implementations of these operations, with the same API and the same behavior: one based on integer arithmetic, and a new one based on floating-point arithmetic; see [25], [39].

New in 2013:

- the floating-point implementation;
- rounding toward zero (only with the integer implementation).
- 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: comment at the beginning of the code and Research report Inria RR-7832.
- URL: https://www.vinc17.net/research/sipe/

CARAMEL Project-Team

5. Software and Platforms

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 2013 is the release of version 3.1.2 (the "canard à l'orange" release) in March. This version fixes a few bugs from previous version.

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.

No new version of GNU MPC was released in 2013, which confirms the status of mature library.

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 February 2013, a new version 6.4.4 was released. Apart from bug fixes, this new release provides some improvements (better integration of the GPU code, of the new *-batch* option, ...).

In September 2013, a new record prime of 83 digits was found by R. Propper using GMP-ECM.

5.5. Finite Fields

Participants: Pierrick Gaudry, Emmanuel Thomé [contact], Luc Sanselme.

mp$\mathbb{F}_q$ is (yet another) library for computing in finite fields. The purpose of 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*. mp$\mathbb{F}_q$ can adapt to finite fields of any characteristic and any extension degree. However, one of the targets being the use in cryptology, mp$\mathbb{F}_q$ somehow focuses on prime fields and on fields of characteristic two.

When it was first written in 2007, mp$\mathbb{F}_q$ established 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. A stream of academic works followed the idea behind mp$\mathbb{F}_q$ and improved over such timings, notably by Scott, Aranha, Longa, Bos, Hisil, Costello.

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

In 2013, version 1.1 of mp$\mathbb{F}_q$ has been released. This new release includes new assembly code by Luc Sanselme providing optimized arithmetic over fields whose characteristic fits in a number of bits which fit within half-word boundaries.

In 2013, Hamza Jeljeli collaborated with Bastien Vialla from LIRMM, Montpellier to integrate experimental code based on RNS arithmetic (Residue Number System), intending to provide back-end functionality for the linear algebra code in CADO-NFS. This feature set is still experimental.

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.

In 2013, the development version of GF2X has been updated to incorporate detection of Intel Haswell micro-processors, which provide much improved performance for the PCLMULQDQ instruction (this instruction if of utmost importance for GF2X).

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

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, and is referenced from the software page at http://cado-nfs.gforge.inria.fr/. A major new release, CADO-NFS 2.0, was published in November 2013. The client/server framework was completely rewritten to allow the use of CADO-NFS routinely on clusters of 100 to 1000 nodes.

More and more people use CADO-NFS to perform medium to large factorizations. Also in 2013 some researchers in the field wrote some papers where they study the implementation and default parameters of CADO-NFS. This is very useful feedback from the scientific community.

5.8. Belenios

Participants: Pierrick Gaudry, Stéphane Glondu [contact].

In collaboration with the CASSIS team, we develop an open-source private and verifiable electronic voting protocol, named BELENIOS. Our system is an evolution of an existing system, Helios, developed by Ben Adida, and used e.g. by UCL and the IACR association in real elections. The main differences with Helios are the following ones:

- In Helios, the ballot box publishes the encrypted ballots together with their corresponding voters. This raises a privacy issue in the sense that whether someone voted or not shall not necessarily be publicized on the web. Publishing this information is in particylar forbidden by CNIL's recommendation. BELENIOS no longer publishes voters' identities, still guaranteeing correctness of the tally.
- Helios is verifiable except that one has to trust that the ballot box will not add ballots. The addition of ballots is particularly hard to detect as soon as the list of voters is not public. We have therefore introduced an additional authority that provides cretentials that the ballot box can verify but bot forge.

This new version has been implemented by Stéphane Glondu [1]. and has been tested in July 2013 in a mock election in the teams CASSIS and CARAMEL.

[1] http://belenios.gforge.inria.fr/

CASCADE Project-Team (section vide)

CRYPT Team (section vide)

GEOMETRICA Project-Team

5. Software and Platforms

5.1. CGAL, the Computational Geometry Algorithms Library

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

With the collaboration of Pierre Alliez, 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, Clément Jamin, 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).

The transfer and diffusion of CGAL in industry is achieved through the company GEOMETRY FACTORY (http://www.geometryfactory.com). GEOMETRY FACTORY is a *Born of Inria* company, founded by Andreas Fabri in January 2003. The goal of this company is to pursue the development of the library and to offer services in connection with CGAL (maintenance, support, teaching, advice). GEOMETRY FACTORY is a link between the researchers from the computational geometry community and the industrial users.

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 number 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 mesh generation and related packages. Two 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 12 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 Project-Team

5. Software and Platforms

5.1. ECPP

François Morain has been continually improving his primality proving algorithm, ECPP, originally developed in the early 1990s. Proving the primality of a 512-bit number requires less than a second on an average PC. Morain's personal record is around 25000 decimal digits, using the FASTECPP variant that he started developing in 2003. The code is written in C, and based on publicly available packages (GMP, MPFR, MPC, MPFRCX).

5.2. SEA

Together with E. Schost and L. De Feo, François Morain has developed a new implementation of the SEA algorithm for computing the cardinality of elliptic curves over large prime and binary finite fields. This program is a `gforge` project, based on the NTL library. The large prime case is relevant to cryptographic needs; the binary case, while not useful in contemporary cryptography, is a good testbed for De Feo's FAAST package.

5.3. TIFA

TIFA (Tools for Integer FActorization), initially developed in 2006, has been continuously improved during the last few years. TIFA includes a base library written in C99 using the GMP library, stand-alone factorization programs, and a basic benchmarking framework. Available online at http://www.lix.polytechnique.fr/Labo/Jerome.Milan/tifa/tifa.xhtml, TIFA is distributed under the Lesser General Public License (version 2.1 or later).

5.4. Quintix

Guillaume Quintin's Quintix library implements efficient arithmetic in Galois rings and their unramified extensions, the root-finding algorithms presented in [7], basic functions for the manipulation of Reed–Solomon codes, and the complete Sudan list-decoding algorithm. Part of the Mathemagix computer algebra system (http://www.mathemagix.org/), the source code is distributed under the General Public License (version 2 or higher).

5.5. finitefieldz

Within the Mathemagix CAS (http://www.mathemagix.org/), Guillaume Quintin wrote the finitefieldz package, which provides arithmetic for finite fields and towers of finite fields, as well as univariate polynomial root finding and factorization over finite fields.

5.6. 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 and benchmarking of general decoding algorithms. As of 2012/12/13, DECODING provides a working list decoding algorithm, but there is no unique decoding algorithm (though this can be emulated by list-decoding up to half the minimum distance). The library is being still under development, and more algorithms will be added. DECODING was presented at the 2012 International Symposium on Symbolic and Algebraic Computation.

LFANT Project-Team

5. Software and Platforms

5.1. Pari/Gp

Participants: Karim Belabas [correspondent], 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: 2.5.5
- Version of `gp2c`: 0.0.8
- License: GPL v2+
- Programming language: C

5.2. GNU MPC

Participants: Andreas Enge [correspondent], Mickaël Gastineau [CNRS], Philippe Théveny [INRIA project-team ARIC], 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 front ends for constant folding, the evaluation of constant mathematical expressions during the compilation of a program. Since 2011, it is an official GNU project.

2012 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: 0.4.2 *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 [8].

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

5.5. AVIsogenies

Participants: Damien Robert [correspondent], 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. APIP

Participant: Jérôme Milan.

http://www.lix.polytechnique.fr/~milanj/apip/apip.xhtml

APIP, Another Pairing Implementation in PARI, is a library for computing standard and optimised variants of most cryptographic pairings.

The following pairings are available: Weil, Tate, ate and twisted ate, optimised versions (à la Vercauteren–Hess) of ate and twisted ate for selected curve families.

The following methods to compute the Miller part are implemented: standard Miller double-and-add method, standard Miller using a non-adjacent form, Boxall et al. version, Boxall et al. version using a non-adjacent form.

The final exponentiation part can be computed using one of the following variants: naive exponentiation, interleaved method, Avanzi-Mihailescu's method, Kato et al.'s method, Scott et al.'s method.

- Version: 2012-10-17
- License: GPL v2+
- Programming language: C with libpari

5.7. CMH

Participants: Andreas Enge, Emmanuel Thomé [INRIA project-team CARAMEL].

http://cmh.gforge.inria.fr/

CMH computes Igusa class polynomials, parameterising two-dimensional abelian varieties (or, equivalently, Jacobians of hyperelliptic curves of genus 2) with given complex multiplication.

- Version: development snapshot
- License: GPL v3+
- Programming language: C

5.8. Cubic

Participant: Karim Belabas.

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

CUBIC is a stand-alone 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

5.9. Euclid

Participant: Pierre Lezowski.

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

Euclid is a program to compute the Euclidean minimum of a number field. It is the practical implementation of the algorithm described in [41]. Some corresponding tables built with the algorithm are also available. Euclid is a stand-alone program depending on the PARI library.

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

5.10. 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 and Platforms

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 solving over the reals polynomial systems and computing sample 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)

Specfun Team

5. Software and Platforms

5.1. Mgfun

(1994–): Maple package for symbolic summation, integration, and other closure properties of multivariate special functions.

Now distributed as part of Algolib, a collection of packages for combinatorics and manipulations of special functions, available at http://algo.inria.fr/libraries/.

5.2. DDMF

(2007–): Web site consisting of interactive tables of mathematical formulas on elementary and special functions. The formulas are automatically generated by OCaml and computer-algebra routines. Users can ask for more terms of the expansions, more digits of the numerical values, proofs of some of the formulas, etc. See http://ddmf.msr-inria.inria.fr/.

5.3. DynaMoW

(2007–): Programming tool for controlling the generation of mathematical websites that embed dynamical mathematical contents generated by computer-algebra calculations. Written in OCaml. See http://ddmf.msr-inria.inria.fr/DynaMoW/.

5.4. Ring

(2004–): Coq normalization tool and decision procedure for expressions in commutative ring theories. Written in Coq and OCaml. Integrated in the standard distribution of the Coq proof assistant since 2005.

5.5. SSReflect

(2006–): Extension of the language of the Coq system. Originally written by G. Gonthier for his formal proof of the Four-Color Theorem. A. Mahboubi and E. Tassi participate to its development, maintenance, distribution, user support and have written its user manual. See http://www.msr-inria.fr/projects/mathematical-components/.

5.6. Coqfinitgroup

(2006–): Coq libraries that cover the mechanization of the proof of the Odd Order Theorem. Stable libraries are distributed with the SSReflect extension. A. Mahboubi is one of the main contributors to the code and its documentation. E. Tassi contributed to the design of core data structures and to parts of the formalization. A formal proof was completed in September 2012, and the content of the libraries, under continued improvements in view of potential reuse, is available online at http://www.msr-inria.fr/projects/mathematical-components/.

VEGAS Project-Team

4. Software and Platforms

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 [7], [10] 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 [29] bundled with the GMP multi-precision integer arithmetic [28]. 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 [30].

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 [4] 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.

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 [32] (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 [34]. We are currently developing a model of the *Bivariate Algebraic Kernel* based a new bivariate polynomial solver.

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* and its algorithms are documented[2]. 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
[2] http://arxiv.org/abs/1307.5655

ALF Project-Team

5. Software and Platforms

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, **tiptop**, a user-level Linux utility that collects data from hardware performance counters for running tasks, and **Padrone**, a platform for dynamic binary analysis and optimization.

5.2. ATMI

Participant: Pierre Michaud.

Microarchitecture temperature model

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.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

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 [9]. 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/index.php?option=com_content&view=article&id=5/havege&catid=3/projects&Itemid=3&lang=fr 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.

Tiptop version 2.2 was released in March 2013.

For more information, please contact Erven Rohou and/or visit http://tiptop.gforge.inria.fr.

5.7. Padrone

Participants: Erven Rohou, Emmanuel Riou.

Performance, profiling, dynamic optimization

Status: Ongoing development, early prototype.

Padrone is new platform for dynamic binary analysis and optimization. It provides an API to help clients design and develop analysis and optimization tools for binary executables. Padrone attaches to running applications, only needing the executable binary in memory. No source code or debug information is needed. No application restart is needed either. This is specially interesting for legacy or commercial applications, but also in the context of cloud deployment, where actual hardware is unknown, and other applications competing for hardware resources can vary. The profiling overhead is minimum.

Padrone is written in C.

For more information, please contact Erven Rohou.

ATEAMS Project-Team

4. Software and Platforms

4.1. MicroMachinations

Participant: Riemer Van Rozen [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:

Objective: To create an integrated, live environment for modeling and evolving game economies. This will allow game designers to experiment with different strategies to realize game mechanics. The environment integrates with the SPIN model checker to prove properties (reachability, liveness). A runtime system for executing game economies allows MicroMachinations models to be embedded in actual games.

Users: Game designers

Impact: One of the important problems in game software development is the distance between game design and implementation in software. MicroMachinations has the potential to bridge this gap by providing live design tools that directly modify or create the desired software behaviors.

Competition: None.

Engineering: The front-end of MicroMachinations is built using the Rascal language workbench, including visualization, model checking, debugging and standard IDE features. The runtime library is implemented in C++ and will be evaluated in the context of industrial game design.

Publications: [28]

4.1.1. Novelties

- Development on MMLib was started which allows the execution of game economies directly within games.

4.2. Derric

Participants: Tijs Van Der Storm, Jeroen Van Den Bos [correspondent].

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

WWW: http://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: [35], [34][14], [16], [15]

4.2.1. Novelties

- Construction of a 1TB benchmark based on Wikipedia images.
- The Derric DSL for digital forensics now features Trinity, a runtime IDE to debug file format descriptions [35].

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, Ashim Shahi.

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: [7], [6], [8], [5], [6]

4.3.1. Novelties

- A new language-parametric model to represent software projects, called M3 [38].
- Performance improvements of the Rascal interpreter throughout.
- Initial version of a compiler for Rascal, based on new language construct guarded coroutines.
- Origin tracking for values and expressions of type string.
- A library for accessing and analyzing Excel and Word documents.
- Improvements to the Rascal IDE: better output handling, hyper linked source code locations in the console, dedicated project explorer view.
- Content completion for DSLs implemented in Rascal.
- Significant improvements to the Rascal static type checker.
- Experiments with improved GLL parsing (Iguana).
- Several new example DSL implementations to illustrate Rascal as a language workbench: Marvol, a DSL for controlling NAO robots, and two implementations of a DSL for questionnaires (DemoQLes and QL-R-Kemi).

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: https://github.com/impulse-org/

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: [2]

4.4.1. Novelties

- The IMP program database (PDB) was completely redesigned.

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 Loh, 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 a completely self-hosted system in 7000 lines of code.

Publications: [12], [17], [11]

4.5.1. Novelties

- A compiler for a dedicated Ensō language, which targets JavaScript.
- Added a demo based on the LWC'13 questionnaire language assignment.

4.6. LiveQL

Participant: Tijs Van Der Storm [correspondent].

Characterization: A1, SO-3, SM-1, EM-2, SDL-4, OC-DA-4-CD-4-MS-4-TPM-4

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

Objective: Experimenting with live programming concepts and techniques in the context of domain specific languages (DSLs).

Users: End-user programmers.

Impact: LiveQL is an experiment in making a DSL "live", i.e. any change to the DSL program is immediately reflected in the running program. This has the potential to widen the audience of DSL users to include end-user programmers.

Competition: The end-goal is to provide live end-user programming environments with domain-specific checking and optimization. The most similar tools are spreadsheet applications. However, these are still quite general.

Engineering: LiveQL is built in Java, using the ANTLR parser generator.

Publications: [36]

4.7. QL-R-Kemi

Participant: Tijs Van Der Storm [correspondent].

Characterization: A1, SO-3, SM-1, EM-2, SDL-4, OC-DA-4-CD-4-MS-4-TPM-4

WWW: https://github.com/cwi-swat/QL-R-Kemi

Objective: Demonstrate the language workbench features of the Rascal meta programming language and environment. Investigate domain specific language application in the domain of questionnaires and surveys.

Users: Students, scientists.

Impact: Questionnaires are common in social science, tax administration and statistics. Discovering the right abstractions for describing questionnaires has the potential to significantly improve the practice of constructing questionnaire software.

Competition: Traditional survey tools are often wizard-based, lack computational capabilities and lack a formal foundation. The same language is built in a number of different language workbenches which served as a benchmark to compare such tools [24].

Engineering: Uses all features of the Rascal language workbench.

Publications: [24]

CAIRN Project-Team

5. Software and Platforms

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 Morvan.

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 for 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. 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.

Developments in Gecos in 2013 have focused on polyhedral loop transformations and efficient SIMD code generation for fixed point arithmetic data-types as a part of the ALMA project. Significant efforts were also been put to provide a coarse-grain parallelization engine targeting the data-flow actor model in the context of the COMPA ANR project. An article describing the design choice and the main features of the framework was presented at the international workshop on Source Code Analysis and Manipulation in september 2013 [46].

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

Participants: 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 2013 allowed us to obtain a fixed-point conversion tool handling functions, conditional structures and repetitive structures having a fixed number of iterations during time. The frontend has been modified to reduce limitations due to syntax of C langage. A new data type (sc_fixed) is now able to be generated from the back-end. In the context of the DEFIS ANR project, the ID.Fix tool has been reorganized to be integrated in the DEFIS toolflow.

In 2013, ID.Fix has been demonstrated during University Booth at IEEE/ACM DATE and IEEE/ACM DAC. See http://www.youtube.com/watch?v=nKYA4hezplQ

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

Participants: Christophe Wolinski [corresponding author], François Charot, Antoine Floc'H [former member].

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 [85] 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 [101], 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.

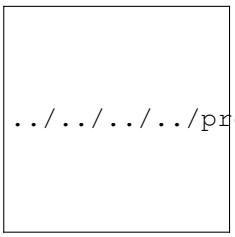

Figure 3. CAIRN*'s PowWow motherboard with radio and energy-harvesting boards connected*

5.7. Ziggie: a Platform for Wireless Body Sensor Networks

Participants: Olivier Sentieys, Olivier Berder, Arnaud Carer, Antoine Courtay [corresponding author], Robin Bonamy.

Keywords: Wireless Body Sensor Networks, Low Power, Gesture Recognition, Localization, Hardware and Software Platform

The Zyggie sensor node has been developed in the team to create an autonomous Wireless Body Sensor Network (WBSN) with the capabilities of monitoring body movements. The Zyggie platform is part of the BoWI project funded by CominLabs. Zyggie is composed of:

- An ATMEGA128RFA1 microcontroller,
- An MPU9150 Inertial Measurement Unit (IMU),
- An RF AS193 switch with two antennas,
- An LSP331AP barometer,
- A DC/DC voltage regulator with a battery charge controller,
- A wireless inductive battery charge controller and
- Some switches and control LEDs.

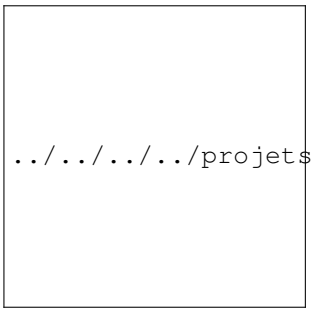

Figure 4. CAIRN*'s Ziggie platform for WBSN*

The IMU is composed of a 3-axis accelerometer, a 3-axis gyrometer and a 3-axis magnetometer. The IMU is communicating its data to the embedded microcontroller via an I2C protocol. We also developed our own MAC protocol for synchronization and data exchanges between nodes. The Zyggie platform is used in many PhD works for evaluating data fusion algorithms (RSSI + IMU data) (Zhongwei Zheng, UR1 and Alexis Aulery, UBS/UR1), low power computing algorithms (Alexis Aulery, UBS/UR1), wireless protocols (Viet Hoa Nguyen, UR1) and body channel characterization (Rizwan Masood, TB).

5.8. SoCLib: Open Platform for Virtual Prototyping of Multi-Processors System on Chip

Participants: François Charot [corresponding author], Laurent Perraudeau [external collaborator].

Keywords: SoC modeling, SystemC simulation model

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 and Platforms

5.1. PolyLib

Participant: Vincent Loechner.

PolyLib [1] 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 (Parkas Projet, Inria Paris - Rocquencourt).

5.2. ZPolyTrans

Participant: Vincent Loechner.

ZPolyTrans [2] 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 [29].

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

Participants: Alain Ketterlin, Philippe Clauss.

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.

[1] http://icps.u-strasbg.fr/PolyLib
[2] http://ZPolyTrans.gforge.inria.fr

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.

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 [27]. 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. APOLLO software and LLVM

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

We are developing a new framework called APOLLO (Automatic speculative POLyhedral Loop Optimizer) whose main concepts are based on our previous framework VMAD. However, several important implementation issues are now handled differently in order to improve the performance and usability of the framework, and also to open its evolution to new interesting perspectives. Thus VMAD played the role of a prototype which is now being re-written as a sustainable tool named APOLLO. As VMAD, APOLLO is dedicated to automatic, dynamic and speculative parallelization of loop nests that cannot be handled efficiently at compile-time. It is composed of a static part consisting of specific passes in the LLVM compiler suite, plus a modified Clang frontend, and a dynamic part consisting of a runtime system. It is described in more details in subsection 6.1 .

Aravind Sukumaran-Rajam (PhD student), Juan Manuel Martinez Caamaño (PhD student), Jean-François Dollinger (PhD student), Willy Wolff (Master student) and Philippe Clauss are the main contributors of APOLLO. It will soon be distributed.

5.7. IBB source-to-source compiler

Participants: Imen Fassi, Philippe Clauss.

We have developed a multifor-compiler called IBB for Iterate-But-Better. IBB translates any C program containing multifor-loops into an equivalent C program in which all multifor-loops are replaced with equivalent for-loops. The resulting source code can then be compiled using any C compiler to produce executable code. IBB will soon be distributed.

5.8. Polyhedral prover

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

We are currently developing a formal proof of program transformations based on the polyhedral model. We use the CompCert verified compiler [28] as a framework. This tool is written in the specification language of Coq.

5.9. CLooG

Participant: Cédric Bastoul.

CLooG [3] is a free software and library to generate code (or an abstract syntax tree of a code) for scanning Z-polyhedra. That is, it finds a code (e.g. in C, FORTRAN...) that reaches each integral point of one or more parameterized polyhedra. CLooG has been originally written to solve the code generation problem for optimizing compilers based on the polyhedral model. Nevertheless it is used now in various area e.g. to build control automata for high-level synthesis or to find the best polynomial approximation of a function. CLooG may help in any situation where scanning polyhedra matters. While the user has full control on generated code quality, CLooG is designed to avoid control overhead and to produce a very effective code. CLooG is widely used (including by GCC and LLVM compilers), disseminated (it is installed by default by the main Linux distributions) and considered as the state of the art in polyhedral code generation.

5.10. OpenScop

Participant: Cédric Bastoul.

OpenScop [4] is an open specification that defines a file format and a set of data structures to represent a static control part (SCoP for short), i.e., a program part that can be represented in the polyhedral model. The goal of OpenScop is to provide a common interface to the different polyhedral compilation tools in order to simplify their interaction. To help the tool developers to adopt this specification, OpenScop comes with an example library (under 3-clause BSD license) that provides an implementation of the most important functionalities necessary to work with OpenScop.

5.11. Clan

Participant: Cédric Bastoul.

Clan [5] is a free software and library which translates some particular parts of high level programs written in C, C++, C# or Java into a polyhedral representation called OpenScop. This representation may be manipulated by other tools to, e.g., achieve complex analyses or program restructurations (for optimization, parallelization or any other kind of manipulation). It has been created to avoid tedious and error-prone input file writing for polyhedral tools (such as CLooG, LeTSeE, Candl etc.). Using Clan, the user has to deal with source codes based on C grammar only (as C, C++, C# or Java). Clan is notably the frontend of the two major high-level compilers Pluto and PoCC.

5.12. Candl

Participant: Cédric Bastoul.

Candl [6] is a free software and a library devoted to data dependences computation. It has been developed to be a basic bloc of our optimizing compilation tool chain in the polyhedral model. From a polyhedral representation of a static control part of a program, it is able to compute exactly the set of statement instances in dependence relation. Hence, its output is useful to build program transformations respecting the original program semantics. This tool has been designed to be robust and precise. It implements some usual techniques for data dependence removal, as array privatization or array expansion, offers simplified abstractions like dependence vectors and performs violation dependence analysis. Candl is notably the dependence analyzer of the two major high-level compilers Pluto and PoCC.

5.13. Clay

Participant: Cédric Bastoul.

[3] http://www.cloog.org
[4] http://icps.u-strasbg.fr/~bastoul/development/openscop
[5] http://icps.u-strasbg.fr/~bastoul/development/clan
[6] http://icps.u-strasbg.fr/~bastoul/development/candl

Clay [7] is a free software and library devoted to semi-automatic optimization using the polyhedral model. It can input a high-level program or its polyhedral representation and transform it according to a transformation script. Classic loop transformations primitives are provided. Clay is able to check for the legality of the complete sequence of transformation and to suggest corrections to the user if the original semantics is not preserved. Clay is still experimental at this report redaction time but is already used during advanced compilation labs at Paris-Sud University and is one of the foundations of our ongoing work on simplifying code manipulation by programmers.

[7] http://icps.u-strasbg.fr/~bastoul/development/clay

COMPSYS Project-Team

5. Software and Platforms

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 mainly concern three activities: a) the development of research tools, in general available on demand, linked to polyhedra and loop/array transformations, b) the development of tools linked to the start-up Zettice, in general not available, c) the development of algorithms within the back-end compilers of STMicroelectronics and/or Kalray.

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. Some of these tools can be used through a web interface on the Compsys tool demonstrator web page http://compsys-tools.ens-lyon.fr/.

The other activity concerns the developments within the compilers of industrial partners such as STMicroelectronics and Kalray. These are not stand-alone tools, which could be used externally, but algorithms and data structures implemented inside the LAO back-end compiler or other compiler branches, year after year, with the help of STMicroelectronics or Kalray colleagues. They are also completed by important efforts for integration and evaluation within the complete compiler toolchains. They concern exact (ILP-based) methods, algorithms for aggressive optimizations, techniques for just-in-time compilation, code representations, and for improving the design of the compiler.

More recently, an important development activity has been started in the context of the Zettice start-up project (see Section 7.3). An important effort of applied research and software development has been achieved since, which results, in particular, in two major software developments: Dcc (DPN C Compiler) and IceGEN. These tools are outlined in Sections 5.8 and 5.9 .

5.2. Pip

Participants: Cédric Bastoul [professor, Strasbourg University and Inria/CAMUS], 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 [16], 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, see Section 5.2). 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 [16].

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 https://bitbucket.org/mbelaoucha/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 [17] 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 has been transferred to the (incubated) start-up Zettice, initiated by Alexandru Plesco.
- Bee has been 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 (see Section 5.5). 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. The underlying theory and corresponding experiments are described in [4].

Chuba has been implemented by Christophe Alias, using the compiler infrastructure PoCo (see Section 5.5). 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. Dcc

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

Dcc (DPN C Compiler) is the *front-end* of the HLS tool transferred to the start-up Zettice (see Section 7.3). Dcc takes as input a C program annotated with pragmas and produces an optimized data-aware process network (DPN). A DPN is a regular process network that makes explicit the I/O transfers and the synchronizations. Dcc features throughput optimization, communication vectorization, and automatic parallelization. Furthermore, Dcc applies analysis to build the DPN circuitry: multiplexing, channels sizing and allocation, FSM generation. To do so, Dcc uses extensively the analysis implemented in PoCo (Section 5.5), in particular dataflow analysis and control generation, and Bee (Section 5.6 for buffer sizing. The DPN specific analysis of Dcc is currently under patent deposit.

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

5.9. IceGEN

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

IceGEN (Integrated Circuit Generator) is the *back-end* of the HLS tool transferred to the start-up Zettice (see Section 7.3). IceGEN takes as input the DPN produced by Dcc (see Section 5.8) and generates:

- a SystemC description relevant for fast and accurate circuit simulation.
- a VHDL description of the circuit, which can be mapped efficiently to an FPGA.

IceGEN makes an extensive use of the pipelined arithmetic operators of the tool FloPoCo [18] developed by Florent De Dinechin, formerly from Inria ARIC team.

IceGEN represents more than 6000 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 (see Section 5.3), 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, see Section 5.11), 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 (see Section 5.12) to prove program termination.

5.11. Aspic

Participant: Laure Gonnord.

Aspic is an invariant generator for general counter automata. Used with C2fsm (see Section 5.10), it can be used to derivate invariant for numerical C programs, and also prove safety. It is also part of the WTC toolsuite (see http://compsys-tools.ens-lyon.fr/wtc/index.html), a set of examples to demonstrate the capability of the RanK tool (see Section 5.12) for evaluating worse-case time complexity (number of transitions when executing an automaton).

Aspic implements the theoretical results of Laure Gonnord's PhD thesis on acceleration techniques and has been maintained since 2007.

5.12. RanK

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

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 theory underlying RanK was presented at SAS'10 [14].

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 (see Section 5.11), 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 (see the WTC benchmark suite http://compsys-tools.ens-lyon.fr/wtc/index.html). It uses the libraries Piplib (Section 5.2) and Polylib.

RanK has been implemented by Christophe Alias, using the compiler infrastructure PoCo (Section 5.5). It represents more than 3000 lines of C++. The tool has been presented at the CSTVA'13 workshop [11].

5.13. SToP

Participants: Christophe Alias, Guillaume Andrieu [University of Lille], Laure Gonnord [Compsys].

SToP (Scalable Termination of Programs) is the implementation of the modular termination technique presented at the TAPAS'12 workshop [15]. 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 (see Section 5.12). 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++.

5.14. 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), see details in [22]. It uses PIP (see Section 5.2) 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.

5.15. 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 past 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 activities related to 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. See details in the "new results" presented in previous Compsys activity reports.

5.16. 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. See details in the "new results" presented in previous Compsys activity reports.

5.17. 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 (a.k.a., 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. MinIR was presented at WIR'11 [29].

CONTRAINTES Project-Team

5. Software and Platforms

5.1. BIOCHAM, biochemical abstract machine

Participants: François Fages, François-Marie Floch, Steven Gay, Sylvain Soliman.

The Biochemical Abstract Machine BIOCHAM 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;
- automatic model reduction and comparison, through the use of subgraph epimorphisms [8];
- 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 Prolog framework dedicated to the analysis of Petri nets. It was originally built for the computation of invariants using GNU Prolog's CLP(FD) solver but has been further extended to allow import/export of various Petri nets formats. In 2013 it was ported to SWI Prolog, in order to use its more general FD solver for the satisfaction of the min-plus constraints coming from the tropical equilibration problem [13].

5.3. STSE (Spatio-Temporal Simulation Environment)

Participant: Szymon Stoma.

The overall goal of this software platform is to gather 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

Participants: Szymon Stoma, Grégory Batt, Pascal Hersen, Artémis Llamosi.

Yeast Image Toolkit (YIT) is a set of tools facilitating segmentation and tracking of yeast cells in brightfield images. Toolkit consists of novel segmentation and tracking algorithm (CellStar), benchmark images and software allowing allowing to asses performance of the tracking. The software is currently under development and is designed to be a CellProfiler plugin. Project webpage is: http://yeast-image-toolkit.biosim.eu/.

5.5. 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.

5.6. 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.

5.7. 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.

5.8. 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.

5.9. 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.

5.10. 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.

DREAMPAL Team

5. Software and Platforms

5.1. Software and Platforms

Download page : https://gforge.inria.fr/frs/?group_id=3646

5.1.1. HoMade

HoMade V4 is available and was used by 140 students this year on a Xilinx Nexys3 board. The Xilinx Virtex6 and Virtex7 are also supporting this new release. All the design is in VHDL except some ISE schematic specifications.

The main novelty of this release concerns :

- three stage pipe-lining of the HoMade core,
- new execution stack to improve frequency,
- instruction memory loading via UART port,
- MSPMD support
- reflexive features: Write-In-Program-Memory (WIM) instruction
- development of new IPs.

A test with 56 HoMade Slaves on a ring topology was running on a Virtex6 on a parallel matrix-vector multiplication example.

A low-level stack-based assembler supports binary generation from a Forth-like post-fixed syntax. It is written in Forth and can automatically generate binary code for the UART Port loading. This assembler will be merged with the JHomade software.

5.1.2. JHomade

JHomade is a software suite including a compiler for the HoMade processor. It allows us to compile HiHope programs (or HoMade assembly) and load the binary on the FPGA board. Its first release was in december 2013.

5.1.3. $\mathbb{K}$-based language-independent symbolic execution and verification tool

The results in [9], [14] were implemented in the $\mathbb{K}$ framework and are distributed with it. The implementation allows users to symbolically execute programs in arbitrary languages defined in $\mathbb{K}$, with the only restriction that data cannot become code (and reciprocally). It also allows users to formally verify programs against specifications written in Reachability Logic, a specification formalism that can be seen as a language-independent Hoare logic. These language-independent tools will be specialized to the languages of interest in the project (HiHope, HoMade assembly and machine code).

INDES Project-Team

5. Software and Platforms

5.1. Introduction

Most INDES software packages, even the older stable ones that are not described in the following sections are freely available on the Web. In particular, some are available directly from the Inria Web site:

http://www.inria.fr/valorisation/logiciels/langages.fr.html

Most other software packages can be downloaded from the INDES Web site:

http://www-sop.inria.fr/teams/indes

5.2. Functional programming

Participants: Cyprien Nicolas, Bernard Serpette, Manuel Serrano [correspondant].

5.2.1. The Bigloo compiler

The programming environment for the Bigloo compiler [7] is available on the Inria Web site at the following URL: http://www-sop.inria.fr/teams/indes/fp/Bigloo. The distribution contains an optimizing compiler that delivers native code, JVM bytecode, and .NET CLR bytecode. It contains a debugger, a profiler, and various Bigloo development tools. The distribution also contains several user libraries that enable the implementation of realistic applications.

BIGLOO was initially designed for implementing compact stand-alone applications under Unix. Nowadays, it runs harmoniously under Linux and MacOSX. The effort initiated in 2002 for porting it to Microsoft Windows is pursued by external contributors. In addition to the native back-ends, the BIGLOO JVM back-end has enabled a new set of applications: Web services, Web browser plug-ins, cross platform development, etc. The new BIGLOO .NET CLR back-end that is fully operational since release 2.6e enables a smooth integration of Bigloo programs under the Microsoft .NET environment.

5.3. Language-based Security

Participants: Tamara Rezk [correspondant], José Santos.

5.3.1. IFJS compiler

The IFJS compiler is applied to JavaScript code. The compiler generates JavaScript code instrumented with checks to secure code. The compiler takes into account special features of JavaScript such as implicit type coercions and programs that actively try to bypass the inlined enforcement mechanisms. The compiler guarantees that third-party programs cannot (1) access the compiler internal state by randomizing the names of the resources through which it is accessed and (2) change the behaviour of native functions that are used by the enforcement mechanisms inlined in the compiled code.

The compiler is written in JavaScript and can be found at http://www-sop.inria.fr/indes/ifJS.

5.4. Web programming

Participants: Gérard Berry, Cyprien Nicolas, Manuel Serrano [correspondant].

5.4.1. The HOP web programming environment

HOP is a higher-order language designed for programming interactive web applications such as web agendas, web galleries, music players, etc. It exposes a programming model based on two computation levels. The first one is in charge of executing the logic of an application while the second one is in charge of executing the graphical user interface. HOP separates the logic and the graphical user interface but it packages them together and it supports strong collaboration between the two engines. The two execution flows communicate through function calls and event loops. Both ends can initiate communications.

The HOP programming environment consists in a web *broker* that intuitively combines in a single architecture a web server and a web proxy. The broker embeds a HOP interpreter for executing server-side code and a HOP client-side compiler for generating the code that will get executed by the client.

An important effort is devoted to providing HOP with a realistic and efficient implementation. The HOP implementation is *validated* against web applications that are used on a daily-basis. In particular, we have developed HOP applications for authoring and projecting slides, editing calendars, reading RSS streams, or managing blogs.

HOP has won the software *open source contest* organized by the ACM Multimedia Conference 2007. It is released under the GPL license. It is available at http://hop.inria.fr.

5.5. Old software

5.5.1. Camloo

Camloo is a caml-light to bigloo compiler, which was developed a few years ago to target bigloo 1.6c. New major releases 0.4.x of camloo have been done to support bigloo 3.4 and bigloo 3.5. Camloo make it possible for the user to develop seamlessly a multi-language project, where some files are written in caml-light, in C, and in bigloo. Unlike the previous versions of camloo, 0.4.x versions do not need a modified bigloo compiler to obtain good performance. Currently, the only supported backend for camloo is bigloo/C. We are currently rewriting the runtime of camloo in bigloo to get more portability and to be able to use HOP and camloo together.

5.5.2. Skribe

SKRIBE is a functional programming language designed for authoring documents, such as Web pages or technical reports. It is built on top of the SCHEME programming language. Its concrete syntax is simple and looks familiar to anyone used to markup languages. Authoring a document with SKRIBE is as simple as with HTML or LaTeX. It is even possible to use it without noticing that it is a programming language because of the conciseness of its original syntax: the ratio *tag/text* is smaller than with the other markup systems we have tested.

Executing a SKRIBE program with a SKRIBE evaluator produces a target document. It can be HTML files for Web browsers, a LaTeX file for high-quality printed documents, or a set of *info* pages for on-line documentation.

5.5.3. Scheme2JS

Scm2JS is a Scheme to JavaScript compiler distributed under the GPL license. Even though much effort has been spent on being as close as possible to R5RS, we concentrated mainly on efficiency and interoperability. Usually Scm2JS produces JavaScript code that is comparable (in speed) to hand-written code. In order to achieve this performance, Scm2JS is not completely R5RS compliant. In particular it lacks exact numbers.

Interoperability with existing JavaScript code is ensured by a JavaScript-like dot-notation to access JavaScript objects and by a flexible symbol-resolution implementation.

Scm2JS is used on a daily basis within HOP, where it generates the code which is sent to the clients (web-browsers). Scm2JS can be found at http://www-sop.inria.fr/indes/scheme2js.

5.5.4. The FunLoft language

FunLoft (described in http://www-sop.inria.fr/teams/indes/rp/FunLoft) is a programming language in which the focus is put on safety and multicore.

FunLoft is built on the model of FairThreads which makes concurrent programming simpler than usual preemptive-based techniques by providing a framework with a clear and sound semantics. FunLoft is designed with the following objectives:

- provide a safe language, in which, for example, data-races are impossible.
- control the use of resources (CPU and memory), for example, memory leaks cannot occur in FunLoft programs, which always react in finite time.
- have an efficient implementation which can deal with large numbers of concurrent components.
- benefit from the real parallelism offered by multicore machines.

A first experimental version of the compiler is available on the Reactive Programming site http://www-sop.inria.fr/teams/indes/rp. Several benchmarks are given, including cellular automata and simulation of colliding particles.

5.5.5. CFlow

The prototype compiler "CFlow" takes as input code annotated with information flow security labels for integrity and confidentiality and compiles to F# code that implements cryptography and protocols that satisfy the given security specification.

Cflow has been coded in F#, developed mainly on Linux using mono (as a substitute to .NET), and partially tested under Windows (relying on .NET and Cygwin). The code is distributed under the terms of the CeCILL-B license.

5.5.6. FHE type-checker

We have developed a type checker for programs that feature modern cryptographic primitives such as fully homomorphic encryption. The type checker is thought as an extension of the "CFlow" compiler developed last year on the same project. It is implemented in F#. The code is distributed under the terms of the CeCILL-B license.

5.5.7. Mashic compiler

The Mashic compiler is applied to mashups with untrusted scripts. The compiler generates mashups with sandboxed scripts, secured by the same origin policy of the browsers. The compiler is written in Bigloo and can be found at http://www-sop.inria.fr/indes/mashic/.

PAREO Project-Team

5. Software and Platforms

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.

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].

Since 2002, we have developed a new system called *Tom* [31], 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 [29]. 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.

TASC Project-Team

5. Software and Platforms

5.1. CHOCO

Participants: Nicolas Beldiceanu, Alexis de Clerq, Jean-Guillaume Fages [main developer], Narendra Jussien [correspondant], Arnaud Letort, Xavier Lorca [correspondant], Thierry Petit, Charles Prud'Homme [main developer], 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.
6. A stable version of Choco, tagged 3.1.0, is available since September 2nd 2013. This version won two silver medals in the MiniZinc Challenge 2013. It has been downloaded more than 4000 times between September and December 2013.

The link to the system and documentation is http://choco.emn.fr.

5.2. IBEX

Participants: Ignacio Salas Donoso, 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.

In 2013 the work on IBEX has focussed on the following points.

- Continuing last year work on the redesign of the architecture, the IBEX library has been augmented with new features. First, *affine forms* (with the help of Jordan Ninin) have been introduced in core calculations as an alternative to interval arithmetic. Five different implementations are under testing. A symbolic differentiation module has been developed and used for applying first-order conditions in global optimization (an interval variant of Khun-Tucker conditions). We have started a redesign of the global optimizer to integrate properly first-order conditions and exact equality constraints (not relaxed to inequalities). This work will be pursued in early 2014.
- In deterministic continuous constrained global optimization, upper bounding the objective function generally resorts to local minimization at several nodes of the branch and bound. We have proposed an alternative approach when the constraints are inequalities and the feasible space has a non-null volume. First, we extract an inner region, i.e., an entirely feasible convex polyhedron or box in which all points satisfy the constraints. Second, we select a point inside the extracted inner region and update the upper bound with its cost. We have implemented this principle with two inner region extraction algorithms, once being based on the algorithm published in CP'10 by G. Chabert & N. Beldiceanu for sweeping with continuous domains. The corresponding paper is *Upper Bounding in Inner Regions for Global Optimization under Inequality Constraints* that has been accepted for publication in JOGO (Journal of Global Optimization).
- The packaging of IBEX has also been considerably enhanced with the integration of a 3rd interval library (Filib++) and a 2nd LP solver (CPlex) and by making the library compatible with 64-bits platforms. The writing of documentation and tutorials has continued. A document of specifications has also been written for an automatic benchmarking tool.

5.3. CHOCO-IBEX

Participants: Gilles Chabert [correspondant], Jean-Guillaume Fages [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. The "Choco-Ibex" interface, initially designed for filtering only, has been augmented with *inflators*, a generic service on which the hybrid *geost* sweep algorithm is based. This gives a basis for a possible implementation of a future hybrid packing solver (objects with curved shapes), the target application of the NetWMS2 project. The interface is available in Choco-3.1.0.

5.4. Artificial Intelligence Using Randomness

Participant: Florian Richoux [correspondant].

AIUR (Artificial Intelligence Using Randomness) is an AI for $StarCraft: BroodWar^{tm}$.

The main idea is to be unpredictable by making some stochastic choices. The AI starts a game with a "mood" randomly picked up among 5 moods, dictating some behaviors (aggressive, fast expand, macro-game, ...). In addition, some other choices (productions, timing attacks, early aggressions, ...) are also taken under random conditions.

Learning is an essential part of AIUR. For this, it uses persistent I/O files system to record which moods are efficient against a given opponent, in order to modify the probability distribution for the mood selection.

AIUR is an open source program under GNU GPL V3 licence, written in C++ (18.000 lines of code). Source and documentations are available at http://code.google.com/p/aiurproject/. AIUR finished 3rd to $StarCraft^{tm}$ AI competitions organized at the conferences AIIDE 2013 and CIG 2013.

5.5. 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 423 constraints, 3936 pages and 900 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 55 exercises with their corrections have been done for half of the core global constraints.
3. a redesign of all the 900 figures of the catalog has been undertaken in autumn 2012 using TikZ (in December 2013 750 figures were redesigned).
4. adding counting information related to the number of solutions of a constraint (integer sequences and visualization).
5. adding constraints related to sequences that we found relevant for learning constraints from electricity production curves.

ESPRESSO Project-Team

5. Software and Platforms

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 dataflow 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).

In 2013, to be able to use the Signal GUI both as a specific tool and as a graphical view under Eclipse, the code of the Signal GUI has been restructured in three parts: a common part used by both tools (28 classes), a specific part for the Signal GUI (2 classes), a specific part for Eclipse (2 classes). Such a structuration facilitates the maintenance of the products.

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 building of the Signal toolbox is managed using the CMake utility (http://www.cmake.org). It is used to ensure the portability on the different operating systems (CMake generates native makefiles). For the same reason, in 2013, we have integrated the management of the tests of the Signal batch compiler under CMake (using CTest).

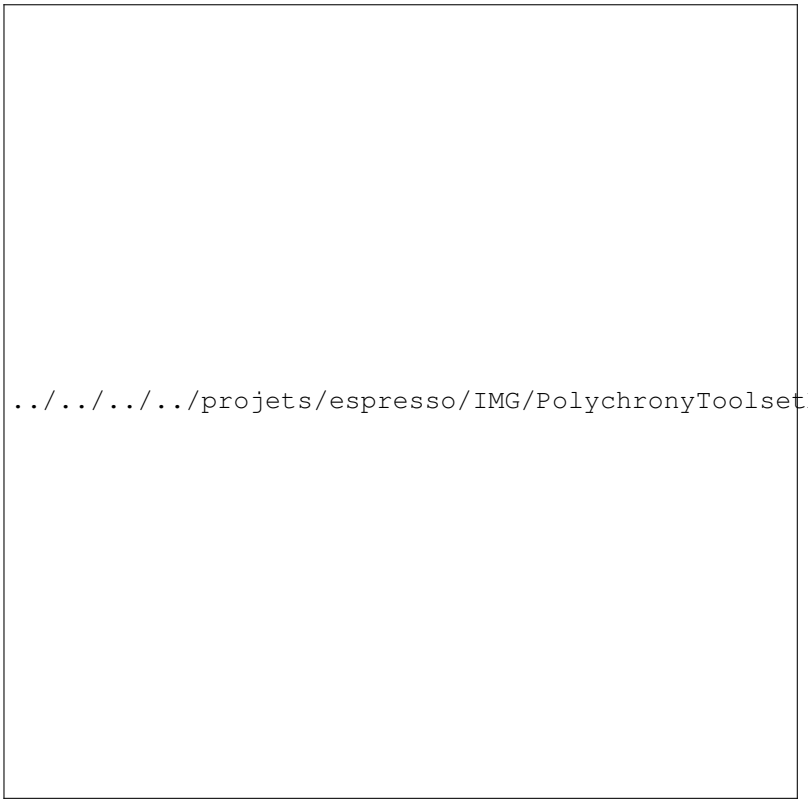

Figure 7. The Polychrony toolset high-level architecture

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

- The Polychrony toolset public web site: http://www.irisa.fr/espresso/Polychrony/index.php. 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.

In 2012, during the OPEES project, we have integrated Polychrony on the Polarsys Experimental Eclipse platform, a new industry collaboration to build open source tools for safety-critical software development. In 2013, the integration to the actual Eclipse Polarsys platform (http://www.polarsys.org) has been started. The proposal project is available at http://www.eclipse.org/proposals/polarsys.polychrony. The creation of a new Polarsys project (as defined in the Eclipse Development Process) is decomposed into several steps (before the first release): project proposal, community review, trademark review, creation review, provisioning, initial contribution. For trademark reasons, Polychrony is called POP as Polarsys Project (POP, a polychronous modeling environment on Polarsys). At the end of 2013, we are at the "provisioning" step. It will be provisioned as soon as the employer consent (Inria, CNRS) form will be signed.

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

Dassault Systèmes, supplies a commercial implementation of Polychrony, called RT-Builder, used for industrial scale projects.

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

Participant: Loïc Besnard.

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 [33]: 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 Polychrony called SME (Signal-Meta under Eclipse). To highlight the different parts of the modeling in Signal, we split the modeling of a Signal process in three diagrams: one to model the interface of the process, one to

model the computation (or dataflow) part, and one to model all explicit clock relations and dependences. The SME environment is available through the ESPRESSO update site [24]. 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.

It should be noted that the Eclipse Foudation does not host code under GPL license. So, the Signal toolbox useful to compile Signal code from Eclipse is hosted on our web server. For this reason, the building of the Signal toolbox, previously managed under Eclipse, has now been exported. The interface of the Signal toolbox for Eclipse is now managed using the CMake tool like the Signal toolbox and the Signal GUI.

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 [25], 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 [26]). 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.

5.4. Safety-Critical Java Level 1 Code generation from Dataflow Graph Specifications

Participants: Adnan Bouakaz, Jean-Pierre Talpin.

We have proposed a dataflow design model [19] of SCJ/L1 applications [43] in which handlers (periodic and aperiodic actors) communicate only through lock-free channels. Hence, each mission is modeled as a dataflow graph. The presented dataflow design model comes with a development tool integrated in the Eclipse IDE for easing the development of SCJ/L1 applications and enforcing the restrictions imposed by the design model. It consists of a GMF editor where applications are designed graphically and timing and buffering parameters can be synthesized. Indeed, abstract affine scheduling is first applied on the dataflow subgraph, that consists only of periodic actors, to compute timeless scheduling constraints (e.g. relation between the speeds of two actors) and buffering parameters. Then, symbolic fixed-priority schedulability analysis (i.e., synthesis of timing and scheduling parameters of actors) considers both periodic and aperiodic actors.

Through a model-to-text transformation, using Acceleo, the SCJ code for missions, interfaces of handlers, and the mission sequencer is automatically generated in addition to the annotations needed by the memory checker. Channels are implemented as cyclic arrays or cyclical asynchronous buffers; and a fixed amount of memory is hence reused to store the infinite streams of tokens. The user must provide the SCJ code of all the `handleAsyncEvent()` methods. We have integrated the SCJ memory checker [52] in our tool so that potential dangling pointers can be highlighted at compile-time. To enhance functional determinism, we would like to develop an ownership type system to ensure that actors are strongly isolated and communicate only through buffers.

S4 Project-Team

5. Software and Platforms

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 [21]. 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 [23]. 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 [20]. Synthesis is constrained to produce so-called distributables nets [22], 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 Team

5. Software and Platforms

5.1. ANR Open-PEOPLE platform

Participants: Anis Koubaa, 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.

In 2013, work in TRIO on this platform was minimal, because of the lack of development resources.

We performed software updates on the servers, ensuring the continuity of access (which is especially important since other partners of the former ANR Open-PEOPLE project still use this platform and actively develop on it). We fixed a few bugs on the platform.

In late 2013, we started working, in the context of Anis Koubaa's student project, on adding a new functionality to help develop energy combustion models, namely the automatic extraction of mathematical laws, derived from the measurements (cloud of points) coming from the experimental hardware platform.

5.2. VITRAIL

Participants: Pierre Caserta, Romarik Jodin, Olivier Zendra.

The aim of the VITRAIL operation is to provide tools for the advanced and immersive visualization of programs. Some of this work has been done with the University of Montréal, the University of Montpellier and to a lesser extent the 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.

In 2013, we were able to restart developments on the VITRAIL platform.

We first explored a Linux port, toward which we progressed but were stopped by the fact we relied on Ogre3D, a library that calls some Windows specific APIs. We have identified those, and we believe that an OpenGL port would be a sensible path to OS portability.

We also ported our VITRAIL Vizualizer software to a new type of display, namely 3 interactive whiteboards placed so as to form a 3-side box.

Finally, we also successfully implemented a first prototype of interaction through a head-tracking system relying on two cameras. First experiments gave promising results.

AOSTE Project-Team

5. Software and Platforms

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 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/download.php), 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. Lopht

Participants: Thomas Carle, Manel Djemal, Zhen Zhang, Dumitru Potop Butucaru [correspondant].

The Lopht (Logical to Physical Time Compiler) has been designed as an implementation of the AAA methodology. Lopht is similar to SynDEx by relying on off-line allocation and scheduling techniques to allow real-time implementation of dataflow synchronous specifications onto multiprocessor systems. But it has two significant originality points: a stronger focus on efficiency (but without compromising correctness), and a focus on novel target architectures (many-core chips and time-triggered embedded systems).

Improved efficiency is attained through the use of classical and novel data structures and optimization algorithms pertaining to 3 fields: synchronous language compilation, classical compiler theory, and real-time scheduling. A finer representation of execution conditions allows us to make a better use of double resource reservation and thus improve latency and throughput. The use of software pipelining allows the improvement of computation throughput. The use of post-scheduling optimisations allows a reduction in the number of preemptions. The focus on novel architectures means that architecture descriptions need to define novel communication media such as the networks-on-chips (NoCs), and that real-time characteristics must include those specific to a time-triggered execution model, such as the Major Time Frame (MTF).

Significant contributions to the Lopht tool have been brought by T. Carle (the extensions concerning time-triggered platforms), M. Djemal (the extensions concerning many-core platforms), and Zhen Zhang under the supervision of D. Potop Butucaru. The tool has been used and extended during the PARSEC project. It is currently used in the IRT SystemX/FSF project, in the collaboration with Astrium Space Transportation (Airbus Defence and Space), and in the collaboration with Kalray SA. It has been developed in OCaml.

5.5. 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 Project-Team

5. Software and Platforms

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 [1]. 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 [47], [46] 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 on-the-fly analysis tools have been developed within the OPEN/CAESAR environment, among which:
  - BISIMULATOR, which checks bisimulation equivalences and preorders,
  - CUNCTATOR, which performs 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 MCL 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,

[1] http://cadp.inria.fr

  - 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_CMP, which compares two graphs,
  - 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 [51], CTL [36], ACTL [37], etc.) to be defined in XTL.

- PBG (*Partitioned BCG Graph*) is a file format implementing the theoretical concept of *Partitioned LTS* [45] 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 [57] 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, Wendelin Serwe.

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 CONVECS.

The TRAIAN compiler can be freely downloaded from the CONVECS Web site [2].

5.3. The PIC2LNT Translator

Participants: Radu Mateescu, Gwen Salaün [correspondent].

We develop a translator named PIC2LNT from an applied π-calculus (see § 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 [3].

5.4. The PMC Partial Model Checker

Participants: Radu Mateescu, Frédéric Lang.

We develop a tool named PMC (*Partial Model Checker*, see § 6.4), which performs the compositional model checking of dataless MCL formulas on networks of communicating automata described in the EXP language.

PMC can be freely downloaded from the CONVECS Web site [4].

[2] http://convecs.inria.fr/software/traian
[3] http://convecs.inria.fr/software/pic2lnt
[4] http://convecs.inria.fr/software/pmc

Hycomes Team

4. Software and Platforms

4.1. 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 [5], [4]. 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).

4.2. Flipflop: A Test and Flip Net Synthesis Tool for Maintenance and Surgical Process Mining

Participant: Benoît Caillaud.

http://tinyurl.com/oql6f3y

Flipflop is a Test and Flip net synthesis tool implementing a linear algebraic polynomial time algorithm. Computations are done in the $\mathbb{Z}/2\mathbb{Z}$ ring. Test and Flip nets extend Elementary Net Systems by allowing test to zero, test to one and flip arcs. The effect of flip arcs is to complement the marking of the place. While the net synthesis problem has been proved to be NP hard for Elementary Net Systems, thanks to flip arcs, the synthesis of Test and Flip nets can be done in polynomial time. Test and flip nets have the required expressivity to give concise and accurate representations of surgical processes (models of types of surgical operations). Test and Flip nets can express causality and conflict relations. The tool takes as input either standard XES log files (a standard XML file format for process mining tools) or a specific XML file format for surgical applications. The output is a Test and Flip net, solution of the following synthesis problem: Given a finite input language (log file), compute a net, which language is the least language in the class of Test and Flip net languages, containing the input language.

This software has been designed in the context of the S3PM project (see Section 6.1).

MUTANT Project-Team

5. Software and Platforms

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.

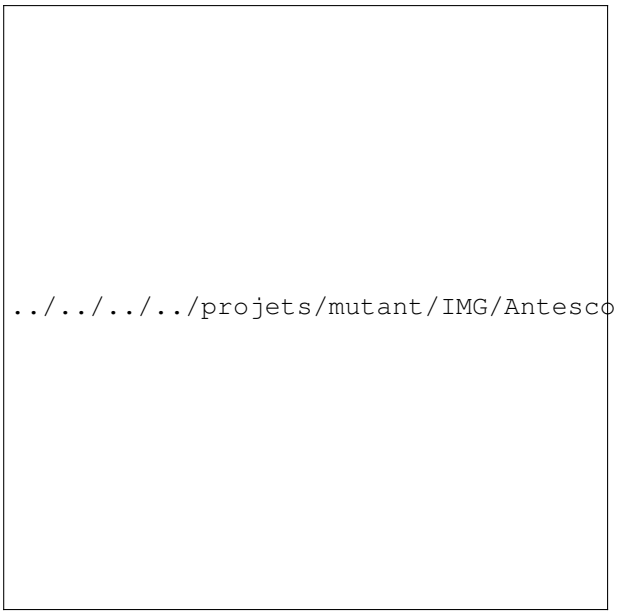

Figure 5. General scheme of Antescofo virtual machine

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

A complete new version of Antescofo has been released on November 2013 on Ircam Forumnet. This version is the result of one year of intensive effort by MuTant team members and associate artists.

This release include major improvements on the reactive language: richer set of synchronization strategies and control structures, dynamic continuous actions, high order function and processes, historicized variables and dynamic expressions everywhere (delays or periods as variables or expressions).

The new internal architecture unifies completely the handling of external (musical) events and the handling of internal (logical) events in a framework able to manage multiple time frames (relative, absolute or computed).

The new version targets the Max and PureData (Pd) environments on Mac, but also on Linux (Pd version) and offers also a standalone offline version. The standalone version is used to simulate a performance in Ascograph.

An important enhancement has been made by proposing a richer set of synchronization strategies between the event recognized by the listening machine and the action to be performed by the reactive engines. Theses new strategies include *anticipative* strategies that exhibits a smoother musical behavior. These strategies are now tested in various musical situations like accompaniments and in the creation of new pieces.

Some new results including a behavioral semantics of the static kernel of the Antescofo reactive engine [17] and tools for formal verification and conformance testing of the system [24], [35] are presented below.

5.2. Ascograph: 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 is held since October 2012 thanks to Inria ADT Support.

AscoGraph, the new Antescofo graphical score editor has been released this year. It provides a autonomous Integrated Development Environment (IDE) for the authoring of Antescofo scores. Antescofo listening machine, when going forward in the score during recognition, uses the message passing paradigm to perform tasks such as automatic accompaniment, spatialization, etc. The Antescofo score is a text file containing notes (chord, notes, trills, ...) to follow, synchronization strategies on how to trigger actions, and electronic actions (the reactive language). This editor shares the same score parsing routines with Antescofo core, so the validity of the score is checked on saving while editing in AscoGraph, with proper parsing errors handling. Graphically, the application is divided in two parts (see Figure 3). On the left side, a graphical representation of the score, using a timeline with tracks view. On the right side, a text editor with syntax coloring of the score is displayed. Both views can be edited and are synchronized on saving. Special objects such as "curves", are graphically editable: they are used to provide high-level variable automation facilities like breakpoints functions (BPF) with more than 30 interpolations possible types between points, graphically editable.

One other really important feature is the score import from MusicXML or MIDI files, which make the complete workflow of the composition of a musical piece much easier than before.

AscoGraph is strongly connected with Antescofo core object (using OSC over UDP): when a score is edited and modified it is automatically reloaded in Antescofo, and on the other hand, when Antescofo follows a score (during a concert or rehearsal) both graphical and textual view of the score will scroll and show the current position of Antescofo.

AscoGraph is released under Open-Source MIT license and has been released publicly along with new Antescofo architecture during IRCAM Forum 2013.

PARKAS Project-Team

5. Software and Platforms

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). As of 2013, 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. The Heptagon language and compiler are a direct descendent of it. The new language Zélus for hybrid systems modeling borrows many features originaly introduced in Lucid Synchrone.

5.2. ReactiveML

Participants: Guillaume Baudart, 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://reactiveml.org. 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). A new application to mixed music programming has been developed.

In 2013, a new web site has been developed. New programming constructs have been added. The runtime system has been cleanup. Moreover, a new implementation based on the PhD of Cédric Pasteur has also been provided http://reactiveml.org/these_pasteur.

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, a parallel code generator with Futures, support for correct and efficient in-place array computations. 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, C++, java and VHDL code.

Transfer activities based on our experience in Heptagon are taking place through the "Fiabilité and Sûreté de Fonctionnement" project at IRT SystemX, led by Alstom Transport, since 2013.

Heptagon is jointly developed with Gwenael Delaval and Alain Girault from the Inria POP ART team (Grenoble). Gwenael Delaval is developing the controller synthesis tool BZR (http://bzr.inria.fr/) above Heptagon. Both software are distributed under a GPL licence.

5.4. Lucy-n: an n-synchronous data-flow programming language

Participants: Albert Cohen, 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 2013, a complete re-implemtantion has been started. This new version will take into account the new features developed during the PhD of Adrien Guatto. Parallel code generation for this new version also involves compilation and runtime system research in collaboration with Nhat Minh Lê and Robin Morisset.

5.5. ML-Sundials

Participants: Timothy Bourke, Jun Inoue, Marc Pouzet [contact].

The ML-Sundials bindings allow the use of the state-of-the-art Sundials numerical simulation library from OCaml programs (like, for instance, the Zélus runtime). The Sundials packages includes three main components: CVODE, IDA, and KINSOL.

This year we redesigned and reimplemented the interface to CVODE to fix a problem with memory leaks between OCaml and C heaps. We have submitted an APP request for this code. The CVODE component is an important part of our work on the Zélus programming language.

We also developed a new interface for the IDA component, which we have started to use in our experiments with DAEs (Modelica).

We plan to develop an interface for the remaining KINSOL component over the next three months and then to release the entire library under an open-source license.

5.6. Zélus

Participants: Timothy Bourke, Marc Pouzet [contact].

Zélus is a new programming language for hybrid system modeling. It is based on a synchronous language but extends it with Ordinary Differential Equations (ODEs) to model continuous-time behabiors. It allows for combining arbitrarily data-flow equations, hierarchical automata and ODEs. The language keeps all the fundamental features of synchronous languages: the compiler statically ensure the absence of deadlocks and critical races; it is able to generate statically scheduled code running in bounded time and space and a type-system is used to distinguish discrete and logical-time signals from continuous-time ones. The ability to combines those features with ODEs made the language usable both for programming discrete controllers and their physical environment.

The Zélus implementation has two main parts: a compiler that transforms Zélus programs into OCaml programs and a runtime library that orchestrates compiled programs and numeric solvers. The runtime can use the Sundials numeric solver, or custom implementations of well-known algorithms for numerically approximating continuous dynamics.

This year we reimplemented several basic numeric solver algorithms after a careful analysis of the Simulink versions together with the binding to SUNDIALS CVODE. This was necessary to enable detailed comparsions between our tool and Simulink (the de facto industrial standard in this domain). We also improved the algorithm for zero-crossing detection, simplified and streamlined the back-end interface.

We developed several new examples to aid in the development, debugging, and dissemination of our work together with various talks and demonstrations. These included a simple backhoe model (which served as a introducing example in the HSCC paper [12]), an adaptive control example from Astrom and Wittenmark's text, and a model of Zeno behaviour based on a zig-zagging object (presented at Synchron).

Zélus has been released officially in 2013 with several complete documented examples on http://zelus.di.ens.fr. An important software development has been done in the compiler internals during year 2013: a new *causality analysis* has been designed and implemented and a new back-end to generate efficient sequential code for both the discrete step and the continuous step.

5.7. GCC

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

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

Licence: GPLv3+ and LGPLv3+

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

PARKAS contributes to the polyhedral compilation framework, also known as Graphite. We also distribute an experimental branch for a stream-programming extension of OpenMP called OpenStream (used in numerous research activites and grants). This effort borrows key design elements to synchronous data-flow languages.

Tobias Grosser is one of main contributors of the Graphite optimization pass of GCC.

5.8. 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 includes an ILP solver based on generalized basis reduction, and a new polyhedral code generator. isl also supports affine transformations for polyhedral compilation, and increasingly abstract representations to model source and intermediate code in a polyhedral framework.

isl has become the de-facto standard for every recent polyhedral compilation project. Thanks to a license change from LGPL to MIT, its adoption is also picking up in industry.

5.9. 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: MIT

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 bed for many compilation algorithms and heuristics published by our group, and is currently the best automatic parallelizer for CUDA and OpenCL (on the Polybench suite).

5.10. Tool support for the working semanticist

Participant: Francesco Zappa Nardelli [contact].

Languages, semantics, tool support, theorem prouvers.

We are working on tools to support large scale semantic definitions, for programming languages and architecture specifications. For that we develop two complementary tools, Ott and Lem.

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.

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, Coq, and Isabelle code.

In collaboration with Peter Sewell (Cambridge University) and Scott Owens (University of Kent).

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 2013 we implemented several bug-fixes, we kept the theorem prouver backends up-to date with the prover evolution, and we have been working toward a closer integration with the Lem tool.

The development version of Lem is available from http://www.cs.kent.ac.uk/people/staff/sao/lem/.

5.11. Cmmtest: a tool for hunting concurrency compiler bugs

Participants: Francesco Zappa Nardelli [contact], Robin Morisset, Pankaj More, Anirudh Kumar, Pankaj Prateek Kewalramani, Pejman Attar.

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.

Cmmtest 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.

Cmmtest is available from http://www.di.ens.fr/~zappa/projects/cmmtest/ and a list of bugs reported thanks to cmmtest is available from http://www.di.ens.fr/~zappa/projects/cmmtest/gcc-bugs.html.

SPADES Team

5. Software and Platforms

5.1. Implementations of Synchronous Programs

Participant: Alain Girault.

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 SYNDEX[4]. Our first scheduling heuristic produces static multiprocessor schedules tolerant to a specified number of processor and communication link failures [62]. The basic principles upon which we rely to make the schedules fault tolerant are, on the one hand, the active replication of the operations [63], 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) [64]. Our second scheduling heuristic is multi-criteria: it produces a static multiprocessor schedule such that the reliability is maximized, the power consumption is minimized, and the execution time is minimized [12][4] [37], [38]. Our results on fault tolerance are summarized in a web page [5].

5.2. Apron and BddApron Libraries

Participant: Bertrand Jeannet.

5.2.1. Principles

The APRON library [6] is dedicated to the static analysis of the numerical variables of a program by abstract interpretation [51]. Many abstract domains have been designed and implemented for analysing the possible values of numerical variables during the execution of a program (see Figure 1). However, their API diverge largely (datatypes, signatures, ...), and this does not ease their diffusion and experimental comparison *w.r.t.* efficiency and precision aspects.

The APRON library 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.

The BDDAPRON library [7] 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, bit vectors) 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 combine BDDs and APRON abstract values for representing invariants holding on both finite-type variables and numerical variables.

[4] http://www-rocq.inria.fr/syndex
[5] http://pop-art.inrialpes.fr/~girault/Projets/FT
[6] http://apron.cri.ensmp.fr/library/
[7] http://pop-art.inrialpes.fr/~bjeannet/bjeannet-forge/bddapron/index.html

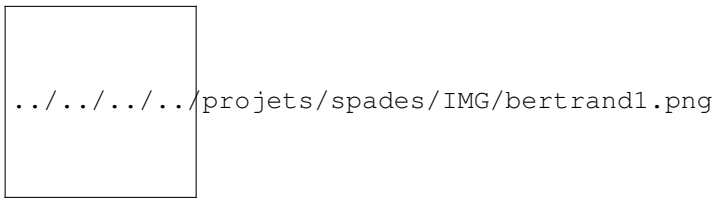

Figure 1. Typical static analyser and examples of abstract domains

5.2.2. 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). It 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.

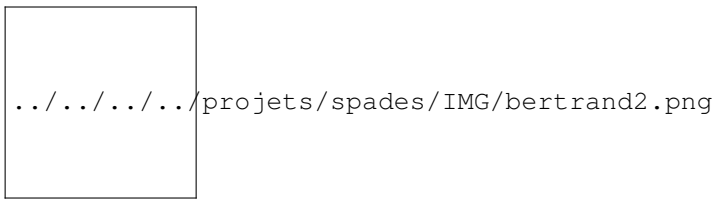

Figure 2. Organisation of the APRON library

5.3. ReaVer

Participant: Bertrand Jeannet.

REAVER (REActive VERifier [8]) 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.

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)

[8] http://members.ktvam.at/schrammel/research/reaver

with convex polyhedra, octagons, intervals, and template polyhedra; and front-ends 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 many more options regarding analysis techniques.

It has been used for several experimental comparisons published in papers. It integrates all the methods developed by Peter Schrammel in his PhD.

5.4. Prototypes

5.4.1. Logical Causality

Participant: Gregor Goessler.

We are developing LOCA, a prototype tool written in Scala that implements the analysis of logical causality described in 6.1.1 . 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.

5.4.2. Cosyma

Participant: Gregor Goessler.

We have developed COSYMA, a tool for automatic controller synthesis for incrementally stable switched systems based on multi-scale discrete abstractions. 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.

5.4.3. Automatic Controller Generation

Participant: Alain Girault.

We have developed a software tool chain to allow the specification of models, 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[9] and Mode Automata [10]. It is the result of a collaboration with Emil Dumitrescu (INSA Lyon) and Eric Rutten from the CTRL-A Inria 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.

5.4.4. 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.

- FIXPOINT[11]: 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 [66].
- INTERPROC[12]: 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 [13].
- CONCURINTERPROC extends INTERPROC with concurrency, for the analysis of multithreaded programs interacting via shared global variables. It is also deployed through a web-interface [14].

[9] http://www.irisa.fr/vertecs/Logiciels/sigali.html
[10] http://www-verimag.imag.fr
[11] http://http://pop-art.inrialpes.fr/people/bjeannet/bjeannet-forge/fixpoint
[12] http://pop-art.inrialpes.fr/people/bjeannet/bjeannet-forge/interproc
[13] http://pop-art.inrialpes.fr/interproc/interprocweb.cgi

- PINTERPROC extends INTERPROC with pointers to local variables. It is also deployed through a web-interface [15].

5.4.5. The SIAAM virtual machine

Participants: Quentin Sabah, Jean-Bernard Stefani [contact person].

The SIAAM abstract machine is an object-based realization of the Actor model of concurrent computation. Actors can exchange arbitrary object graphs in messages while still enjoying a strong isolation property. It guarantees that each actor can only directly access objects in its own local heap, and that information between actors can only flow via message exchange [10]. The SIAAM machine has been implemented for Java as a modified Jikes virtual machine. The resulting SIAAM software comprises:

- A modified Jikes RVM that implements actors and actor isolation as specified by the SIAAM machine.
- A set of static analyses build using the Soot Java optimization framework for optimizing the execution of the SIAAM/Jikes virtual machine, and for helping programmers diagnose potential performance issues.
- A formal proof using the Coq proof assistant of the SIAAM isolation property.

[14] http://pop-art.inrialpes.fr/interproc/concurinterprocweb.cgi
[15] http://pop-art.inrialpes.fr/interproc/pinterprocweb.cgi

FORMES Team

5. Software and Platforms

5.1. CoLoR

Participants: Frédéric Blanqui, Kim-Quyen Ly.

CoLoR is a Coq library on rewriting theory and termination of more than 83,000 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, finite graphs.
- Term structures: strings, algebraic terms with symbols of fixed arity, algebraic terms with varyadic symbols, pure and simply typed λ-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. It is currently developed by Frédéric Blanqui and Kim-Quyen Ly, but various people participated to its development since 2006.

5.2. HOT

Participant: Frédéric Blanqui.

HOT is an automated termination prover for higher-order rewrite systems based on the notion of computability closure and size annotation [24]. It won the 2012 competition in the category "higher-order rewriting union beta". The sources are not public.

5.3. Moca

Participant: Frédéric Blanqui.

Moca is a construction functions generator for OCaml 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 users defined 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. It is developed by Frédéric Blanqui, Pierre Weis (EPI Pomdapi) and Richard Bonichon (CEA).

5.4. Rainbow

Participants: Frédéric Blanqui, Kim-Quyen Ly.

Rainbow is a tool for automatically 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 (PhD subject of Kim-Quyen Ly).

Rainbow is distributed under the CeCILL license. It is currently developed by Frédéric Blanqui and Kim-Quyen Ly. See the web site for more information.

5.5. 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. 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~[16]. As a consequence, Coq and CoqMTU are the first proof assistants, of 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 in CoqMTU 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 done by the summer of 2014. We consider this work as a major scientific achievement of the team.

5.6. SimSoC

Participants: Vania Joloboff [correspondant], Antoine Rouquette, Shenpeng Wang.

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 is based on the open source libraries from the OSCI Consortium: SystemC version 2.3 and TLM 2.0.1 [39], [21]. 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, hence the simulator has a single simulation loop.

The third open source version of **SimSoC**, release 0.8.0, has been released in September 2013. It contains a full simulator for ARM (V5 and V6) and PowerPC both running at an average speed of about 100 Millions instructions per second in, and a deprecated simulator for the MIPS architecture. **SimSoC** is distributed under LGPL on Inria Gforge web site.

5.7. SimSoC-Cert

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

Simulators such as **SimSoC** make it possible to reduce development time and development cost, allowing for the software engineers to run fast iterative cycles without requiring a hardware development board. Then a critical issue is: *does the simulator actually simulate the real hardware?*

Considering only one module in **SimSoC**, namely the ARM simulator, it somehow encodes the 1138 pages of the ARM reference manual in C++. The whole simulator, which simulates ARM and PowerPC architecture, includes about 60,000 lines of manually coded C++ code. Then, mistakes in the hand written code are unavoidable and difficult to find due to the complexity. From the experiments performed on **SimSoC**, bugs bringing a wrong behavior were observed from time to time but it was hard to reveal where they were. Using intensive tests can cover most of the instructions, but still left some untested rare cases of instructions, which lead to potential problems.

Therefore, a better approach is required to gain confidence in the correctness of the simulator. Our proposal has been to certify the ARM CPU simulator from **SimSoC** using formal methods. We aimed at proving a significant part of the correctness of **SimSoC** in order to support the claim that the implementation of the simulator and the real hardware system will exhibit the same behavior.

In addition, we developed tools that can automatically generate in various C the core simulator, including the decoding functions and the instruction set of the ARMv6 architecture manual [18] (implemented by the ARM11 processor family). The input of SimSoC-Cert is the ARMv6 architecture manual itself.

In order to get the required flexibility and accuracy, we wanted to experiment a direct approach based on a general proof assistant such as Coq. Fortunately, an operational semantics formalized in Coq of a large enough subset of the C language is available from the **CompCert** project. We then decided to base our correctness proofs on this technology. Up to our knowledge, this is the first development of formal correctness proofs based on operational semantics, at least at this scale.

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 v0.8.

We can also generate similar programs for SH4 [20] but this is still experimental (work done by Frédéric Tuong in 2011).

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

SECSI Project-Team

5. Software and Platforms

5.1. Orchids

Participants: Jean Goubault-Larrecq [correspondant], Pierre-Arnaud Sentucq.

The ORCHIDS real-time intrusion detection system was created in 2003-04 at SECSI. Orchids is at the core of a contract between Inria and DGA, started in April 2013, for three years.

Progress in 2013 included:

- Creation of a collection of VirtualBox virtual machines with a pre-installed instance of Orchids, for easy testing and/or installation.
- A collection of scripts, allowing one to rebuild the above cited virtual machines automatically from the sources, as a nightly build (in progress).
- A new algorithm for evaluating the worst-case thread complexity of detection by Orchids, whose first principles were laid out by Jean Goubault-Larrecq, and with two prototype implementations done by Jean-Philippe Lachance, a young L2 intern from Université Laval, Québec. The purpose is to warn users of the complexity of the tasks they delegate to Orchids, and to avert denial of service attacks on Orchids itself.

Objectives for 2014 include:

- Simplifying the Orchids installation process, which has gotten complicated over the years.
- Implementing a frontend tool incorporating the full-fledged version of the worst-case thread complexity algorithm mentioned above, plus some other checks.

ABSTRACTION Project-Team

5. Software and Platforms

5.1. The Apron Numerical Abstract Domain Library

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

Keywords: 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).

Additionally, APRON is used internally by the team to assist the research on numeric domains and static analyses by enabling the development of fast prototypes. In 2013, APRON has been used to design a sufficient-condition generator prototype (6.3.2), the FUNCTION prototype analyzer for termination (5.6 ,6.12), a constraint solver based on numeric abstract domains (6.5), a prototype implementation and extension of the Two Variables Per Inequality abstract domain [27].

5.2. The Astrée Static Analyzer of Synchronous Software

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

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

ASTRÉE is a static analyzer for sequential programs based on abstract interpretation [41], [31], [42], [33].

The ASTRÉE static analyzer [30], [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 scientific leader, correspondent], Radhia Cousot, Jérôme Feret, Antoine Miné, Xavier Rival.

Keywords: 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], [35]. 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 1,050 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 (8.1.1.2).

5.4. The MemCAD static analyzer

Participants: Xavier Rival [correspondent], Antoine Toubhans.

Keywords: Shape analysis. MemCAD is a static analyzer that focuses on memory abstraction. It takes as input C programs, and computes invariants on the data structures manipulated by the programs. It can also verify memory safety. It comprises several memory abstract domains (flat representation, graph abstraction with summaries based on inductive definitions of data-structures, such as lists) and combination operators for memory abstract domains (hierarchical abstraction, reduced product). The current implementation comes with over 200 small size test cases that are used as regression tests.

5.5. A New Tactic Engine for Coq

Participant: Arnaud Spiwack [Correspondent].

Keywords: Coq, Proof assistant, Dependent types, Tactics, Proof search.

Coq is a proof assistant based on dependent type theory developed chiefly at Inria. This project addresses longstanding usability issues when developing proofs interactively: proofs, in Coq, are typically sequences of instructions – called tactics – which transform the proof into further proof obligations. The expressiveness of tactic affects the kind of proofs which can be written realistically in Coq. Two issues have been addressed. First, providing more backtracking primitives: in a typical automated procedure – which a Coq user could write to discharge proof obligations without human effort – there is some amount of non-determinism. It is hence important to be able to devise strategies, and Coq suffered from limited options on that front. The new tactics support further primitives loosely inspired by Prolog, in particular a backtracking choice (like disjunction in Prolog), and a primitive “once” which is akind to Prolog’s soft-cut control primitive.

The second issue is more fundamental: since Coq is based on dependent types, a proof can appear in the statement of a proof obligation. As a result, tactics should be able to handle so-called dependent subgoals (where several proof obligations are left to be discharged, and the proof of one of them is mentioned into the statement of another). This was not historically the case in Coq, which had a direct influence on some users.

Both of the backtracking primitive and the dependent subgoals are part of the development version of Coq and will be part of the next release.

5.6. FuncTion: An Abstract Domain Functor for Termination

Participant: Caterina Urban.

Keywords: Conditional termination, Ranking functions, Static analysis.

FUNCTION is a research prototype static analyzer to analyze the termination of programs written in a small non-deterministic imperative language: it can infer sufficient conditions so that all executions terminate (conditional definitive termination). Following the general framework to analyze termination by abstract interpretation proposed in [40], FUNCTION infers ranking functions using piecewise-defined abstract domains. A first version of FUNCTION implemented a domain of linear ranking functions partitioned by variable bounds [24], [23]. It has been generalized in [22] and [29] to support ordinal-valued ranking functions of the form $\sum_{i=1}^{n} \omega^i f_i$ where n is a constant and each f_i is a natural-valued linear function of the variables.

The analyzer is written in OCaml and implemented on top of the APRON library (5.1). It can be used on-line through a web interface: http://www.di.ens.fr/~urban/FuncTion.html.

FUNCTION entered the 3rd Competition on Software Verification (SV-COMP 2014) in the termination category (demonstration section, no ranking).

5.7. The OpenKappa Modeling Plateform

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

Keywords: Causal traces, Model reduction, Rule-based modeling, 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 [53], a static analyzer [52] (for debugging models), a simulator [51], a compression tool for causal traces [50], [48], and a model reduction tool [4], [49], [58].

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.A compiler from CellDesigner into Kappa has been released in 2013.

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).

5.8. Translation Validation

Participant: Xavier Rival [correspondent].

Keywords: 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 to 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.

5.9. Zarith

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

Keywords: 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.

CELTIQUE Project-Team

4. Software and Platforms

4.1. Javalib

Participants: Frédéric Besson [correspondant], David Pichardie, Pierre Vittet, Laurent Guillo.

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: 2.3
- Programming language: Ocaml

4.2. SAWJA

Participants: Frédéric Besson [correspondant], David Pichardie, Pierre Vittet, Laurent Guillo.

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: 1.5
- Programming language: Ocaml

4.3. Jacal

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

Static program analysis, Javacard, Certification, AFSCM

`Jacal` is a JAvaCard AnaLyseur developed on top of the SAWJA (see Section 4.2) 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 provides a CounterExample Guided Abstraction Refinement (CEGAR) tool for tree automata completion. The CEGAR part is based on the Buddy BDD library. Timbuk also provides an implementation of Lattice Tree Automata to (efficiently) represent built-in values such as integers, strings, etc. in recognized tree languages. See also the web page http://www.irisa.fr/celtique/genet/timbuk/.

- Version: 3.1
- Programming language: Ocaml

4.5. JSCert

Participants: Martin Bodin, Alan Schmitt.

The JSCert project aims to really understand JavaScript. JSCert itself is a mechanised specification of JavaScript, written in the Coq proof assistant, which closely follows the ECMAScript 5 English standard. JSRef is a reference interpreter for JavaScript in OCAML, which has been proved correct with respect to JSCert and tested with the Test 262 test suite.

We plan to build other verification and analysis projects on top of JSCert and JSRef, in particular the certification of derivations in program logics or static analyses.

This project is an ongoing collaboration between Inria and Imperial College. More information is available at http://jscert.org/.

DEDUCTEAM Exploratory Action

5. Software and Platforms

5.1. 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's core is based on the standard algorithm [42] for type-checking semi-full pure type systems and implements a state-of-the-art reduction machine inspired from Matita's [40] and modified to deal with rewrite rules.

Dedukti's input language features term declarations and definitions (opaque or not) and rewrite rule definitions. A basic module system allows the user to organize its project in different files and compile them separately.

Dedukti has been developed by Mathieu Boespflug, Olivier Hermant, Quentin Carbonneaux, and Ronan Saillard. It is composed of about 1000 lines of OCaml.

5.2. Coqine, Holide and Focalide

Dedukti comes with three companion tools: Holide, an embedding of HOL proofs through the OpenTheory format [51], Coqine, an embedding of Coq proofs, and Focalide, an embedding of FoCaLiZe certified programs. All of the OpenTheory standard library and a part of Coq's and FoCaLiZe's libraries are checked by Dedukti.

A preliminary version of Coqine supports the following features of Coq: the raw Calculus of Constructions, inductive types, and fixpoint definitions. Coqine is currently being rewritten to support universes. Coqine has been developed by Mathieu Boespflug, Guillaume Burel, and Ali Assaf.

Holide supports all the features of HOL, including ML-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.

Focalide supports the object-oriented features of FoCaLiZe, including inheritance, late-binding, redefinition and class parameters, and functional programming features of FoCaLiZe. It has been updated to work with the last version of FoCaLiZe. Focalide has been developed by Raphaël Cauderlier.

5.3. 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.

A tool that transforms axiomatic theories into polarized rewriting systems, thus making them usable in iProver Modulo, has also been developed. Autotheo supports several strategies to orient the axioms, some of them being proved to be complete, in the sense that Ordered polarized resolution modulo the resulting systems is refutationally complete, some others being merely heuristics. In practice, autotheo takes a TPTP input file and transforms the axioms into rewriting rules, and produces an input file for iProver Modulo.

iProver Modulo and autotheo have been developed by Guillaume Burel. iProver Modulo is released under a GPL license.

iProver Modulo entered CASC-24, the competition of Automated Theorem Provers held during the 24th CADE conference, in the first-order theorem division, using autotheo to orient the axioms of the problems.

5.4. Super Zenon and Zenon Modulo

Several extensions of the *Zenon* automated theorem prover (developed by Damien Doligez at *Inria* in the *Gallium* team) to Deduction modulo have been studied. These extensions intend to be applied in the context of the automatic verification of proof rules and obligations coming from industrial applications formalized using the *B* method.

The first extension, developed by Mélanie Jacquel and David Delahaye, is called *Super Zenon* and is an extension of *Zenon* to superdeduction, which can be seen as a variant of Deduction modulo. This extension is a generalization of previous experiments [10] together with Catherine Dubois and Karim Berkani (*Siemens*), where *Zenon* has been used and extended to superdeduction to deal with the *B* set theory and automatically prove proof rules of *Atelier B*. This generalization consists in allowing us to apply the extension of *Zenon* to superdeduction to any first order theory by means of a heuristic that automatically transforms axioms of the theory into rewrite rules. This work is described in [5], which also proposes a study of the possibility of recovering intuition from automated proofs using superdeduction.

The second extension, developed by Pierre Halmagrand, David Delahaye, Damien Doligez, and Olivier Hermant, is called *Zenon Modulo* and is an extension of *Zenon* to Deduction modulo. Compared to *Super Zenon*, this extension allows us to deal with rewrite rules both over propositions and terms. Like *Super Zenon*, *Zenon Modulo* is able to deal with any first order theory by means of a similar heuristic. To assess the approach of *Zenon Modulo*, we have applied this extension to the first order problems coming from the TPTP library. An increase of the number of proved problems has been observed, with in particular a significant increase in the category of set theory. This result in the set category allows us to be quite optimistic for the use of *Zenon Modulo* in the framework of the *BWare* project, since the *B* method is actually based on a set theory modeling technique. Over these problems of the TPTP library, we have also observed a significant proof size reduction, which confirms this aspect of Deduction modulo. These results are gathered into two publications [22], [23].

5.5. Zipperposition and Logtk

Zipperposition is an implementation of the superposition method. It experiments theory handling using its extensibility features. Current development includes splitting it into a generic library for representing logic data structures and algorithms, and a prover that uses this library. The library is called LOGTK("logic tool kit"). Zipperposition itself, in its development version, can deal with polymorphic logic, and integer and rational arithmetic. Theoretical work on an efficient inference system for arithmetic is ongoing. It entered CASC-24, the competition of Automated Theorem Provers held during the 24th CADE conference, in the first-order theorem division.

Zipperposition is developed by Simon Cruanes.

Logtk is in active development, in parallel with Zipperposition, and an unstable version is released here. The library, among other things, provides first-order terms, with polymorphic types and some type inference, first-order formulas, unification, term ordering, term rewriting, reduction of formulas to CNF, congruence closure, and an optional implementation of the meta-prover Simon Cruanes and Guillaume Burel have published about [21]. Logtk focuses on efficiency and generality of its constructs. Several term indexing structures usable for rewriting, resolution, or subsumption checking expose a functorial interface that allows to associate any data with indexed terms.

Logtk also relies on some smaller OCaml developments by Simon Cruanes, especially a full-fledged implementation of Datalog and efficient iterators.

5.6. CoLoR

CoLoR is a Coq library on rewriting theory and termination of more than 83,000 lines of code [2]. It provides definitions and theorems for:

- Mathematical structures: relations, (ordered) semi-rings.
- Data structures: lists, vectors, polynomials with multiple variables, finite multisets, matrices, finite graphs.
- Term structures: strings, algebraic terms with symbols of fixed arity, algebraic terms with varyadic symbols, pure and simply typed λ-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. It is currently developed by Frédéric Blanqui and Kim-Quyen Ly, but various people participated to its development since 2006 (see the website for more information).

5.7. HOT

HOT is an automated termination prover for higher-order rewrite systems based on the notion of computability closure and size annotation [44]. It won the 2012 competition in the category "higher-order rewriting union beta". The sources are not public. It is developed by Frédéric Blanqui.

5.8. Moca

Moca is a construction functions generator for OCaml 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. It is developed by Frédéric Blanqui, Pierre Weis (EPI Pomdapi) and Richard Bonichon (CEA).

5.9. Rainbow

Rainbow is a tool for automatically 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 (PhD subject of Kim-Quyen Ly).

Rainbow is distributed under the CeCILL license. It is currently developed by Frédéric Blanqui and Kim-Quyen Ly. See the website for more information.

GALLIUM Project-Team

5. Software and Platforms

5.1. OCaml

Participants: Damien Doligez [correspondant], Alain Frisch [LexiFi], Jacques Garrigue [Nagoya University], Fabrice Le Fessant, Xavier Leroy, Luc Maranget.

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, and compilation manager.

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

5.2. CompCert C

Participants: Xavier Leroy [correspondant], Sandrine Blazy [EPI Celtique], Jacques-Henri Jourdan.

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 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. The diy tool suite

Participants: Luc Maranget [correspondant], Jade Alglave [University College London], Susmit Sarkar [University of St Andrews], Peter Sewell [University of Cambridge].

The **diy** suite provides a set of tools for testing shared memory models: the **litmus** tool for running tests on hardware, various generators for producing tests from concise specifications, and **herd**, a memory model simulator. Tests are small programs written in x86, Power or ARM assembler that can thus be generated from concise specification, run on hardware, or simulated on top of memory models. Test results can be handled and compared using additional tools.

The tool suite and a comprehensive documentation are available from http://diy.inria.fr/.

5.4. 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/

5.5. JoCaml

Participant: Luc Maranget.

JoCaml is an experimental extension of OCaml that adds support for concurrent and distributed programming, following the programming model of the join-calculus.

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

MARELLE Project-Team

5. Software and Platforms

5.1. 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 in 2012.

5.2. 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/v8.4.

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

5.3. Easycrypt

Participants: Gilles Barthe [IMDEA Software Institute], François Dupressoir [IMDEA Software Institute], Benjamin Grégoire [correspondant], César Kunz [IMDEA Software Institute], Benedikt Schmid [IMDEA Software Institute], Pierre-Yves Strub [IMDEA Software Institute].

EasyCrypt is a toolset for reasoning about relational properties of probabilistic computations with adversarial code. Its main application is the construction and verification of game-based cryptographic proofs. EasyCrypt can also be used for reasoning about differential privacy.

ZooCrypt is an automated tool for analyzing the security of padding-based public-key encryption schemes (i.e. schemes built from trapdoor permutations and hash functions). ZooCrypt includes an experimental mechanism to generate EasyCrypt proofs of security of analyzed schemes.

5.4. CoqEAL

Participants: Maxime Dénès, Yves Bertot [correspondant].

CoqEAL is a library of certified algorithms for linear algebra to be used in Coq. It provides a collection of algorithms to compute efficiently on matrices and polynomials. These algorithms are designed to run efficiently directly in the Coq system and take the best advantage of the internal execution capabilities of the this system (virtual machine execution of native code execution after compilation).

5.5. CoqApprox

Participants: Nicolas Brisebarre [CNRS], Mioara Joldes, Érik Martin-Dorel, Micaela Mayero [Iut de Villetaneuse], Jean-Michel Muller, Ioana Paşca [Iut de Nimes], Laurence Rideau, Laurent Théry.

We develop a formalization of rigorous polynomial approximation using Taylor models inside the Coq proof assistant, with a special focus on genericity and efficiency for the computations.

5.6. CoqHensel

Participants: Érik Martin-Dorel, Laurent Théry, Micaela Mayero [Iut de Villetaneuse], Guillaume Hanrot [ENS Lyon].

The CoqHensel library provides a Coq formalization of Hensel's lemma for both univariate and bivariate cases, with some effective and modular certificate checkers for the univariate small integral roots problem, the bivariate small integral roots problem, as well as the integer small value problem (ISValP), with the ultimate goal to provide a fully formally verified chain for solving the Table Maker's Dilemma.

MEXICO Project-Team

5. Software and Platforms

5.1. Software and Platform

5.1.1. Software

5.1.1.1. 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/

5.1.1.2. 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/. Mole served as an experimentation platform for several of our papers this year, notably [38] and [46].

In the context of MExICo, we have created a new tool called Cunf [47], which is able to handle contextual nets, i.e. Petri nets with read arcs [12].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; 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 5.1.2.1). Cunf has also participated in the Model Checking Contest held at the Petri Nets conference in 2013.

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

Participant: 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/

5.1.2. Platforms

5.1.2.1. CosyVerif

Participants: Serge Haddad, Alban Linard [correspondant], Benoît Barbot.

CosyVerif (http://www.cosyverif.org/) is a platform dedicated to the formal specification and verification of dynamic systems. It allows to specify systems in a graphical editor, using several formalisms (such as automata and Petri nets) and to run verification tools on these models in a dedicated execution server. These tools are mainly developed by researchers of the MeFoSyLoMa group (a Parisian verification group, http://www.mefosyloma.fr/).

The platform is available as installable bundles, that contain both the client, the server, and the tools. It is also usable through two public servers: one with the latest release, one with the development version.

CosyVerif does not only handle several formalisms, but also allows to easily define new ones and integrate them within the platform. To the best of our knowledge, no other verification framework presents such a feature.

It has different kinds of users:

- Tool developers, that are usually researchers, can use the platform to distribute their tools, and have a demonstration version easily available.
- Students can use this platform in modeling and verification courses.
- Industrial case studies are also a target of the CosyVerif platform, in order to promote the practice of formal verification in industry.

The platform is managed by a steering committee consisting of researchers and engineers. This committee decides strategic orientations as well as technical choices.

This year, we have improved the platform in several ways.

- Tools: the platform handles two families of formalisms: automata and Petri nets, both with extensions. It currently integrates 10 tools with 4 new ones this year. Some of them perform structural analyses like invariant computations, while other tools perform behavioural analyses: symbolic reachability graph building, unfolding, stochastic simulations, etc.
- Server: the execution server has been enhanced with asynchronous executions, that allow to disconnect and reconnect the client in long executions. It has also been improved by the ability to communicate between servers to share their available tools.
- Client: a new command line client has been developed for scripting the executions.
- Usability: the client and server are new distributed as one bundle that can be installed easily on all platforms. The server and its tools are embedded within a virtual machine to achieve this portability.

All the developed software are open source and free software tools.

Two engineers have worked this year on CosyVerif:

- Francis Hulin-Hubard, part-time (CNRS engineer);
- Alban Linard, full-time (Inria engineer).

CosyVerif has been the subject of two international communications [28], [29]. It has been presented at the french-speaking PhD school ETR'2013 in Toulouse. and used for teaching in the master SAR of University Pierre et Marie Curie.

PARSIFAL Project-Team

5. Software and Platforms

5.1. Abella

Participants: Kaustuv Chaudhuri [correspondant], Matteo Cimini, Dale Miller, Olivier Savary-Bélanger, Yuting Wang.

Main web-site: http://abella-prover.org.

Abella is an interactive theorem prover based on the two-level 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 2013, as part of the RAPT Associated Team, Chaudhuri and Yuting Wang (former intern from Univ. Minnesota) released version 2.0 of Abella, a culmination of nearly two years of work and a significant improvement in its expressivity. Specifically,

> The Abella specification logic now supports the full higher-order hereditary Harrop logic of λProlog. This logic allows for very natural specifications of higher-order relations, and leads to cleaner and simpler proofs.
>
> The Abella reasoning logic was extended with support for arbitrary dynamic contexts and incremental backchaining. The design is based on fundamental insights from *focusing*, a core strength of the team.
>
> A number of illustrative examples of the use of higher-order reasoning were added to the Abella examples library, including a novel new characterization of marked β-reduction in the λ-calculus in terms of a simple higher-order inductive definition of λ-paths.
>
> These results were published in PPDP 2013 [26].

Abella continues to evolve as part of RAPT. In 2013, we hosted an intern from McGill University, Olivier Savary-Bélanger (supervised by Chaudhuri), who investigated extensions of Abella with *regular context schemas*. Among his contributions:

> Abella's reasoning level has been augmented with a *plugin* system that both extends the syntax of Abella theories and adds new tactics.
>
> The main plugin for context schemas allows definitions of regular contexts and context relations, with entirely automatic proofs of the main administrative lemmas.
>
> Experimentally, this extension can be used to eliminate up to 40% of the proof text, including nearly 100% of the administrative lemmas on contexts, from typical examples from the meta-theory of the λ-calculus.

We expect this extension to become part of the 2.1 release of Abella, scheduled for later in 2014.

One important application of Abella emerged in 2013: the formalization of bisimulation-up-to techniques for process calculi such as CCS and the π-calculus. Chaudhuri, Cimini, and Miller have formulated the correctness proof of a number of prominent up-to-techniques using the co-inductive and higher-order facilities of Abella. This work indicates an important emerging direction for Abella: modular reasoning.

In terms of development, we have welcomed Savary-Bélanger into the development team, and added a number of collaborators into the management team for the Abella web-site.

5.2. Bedwyr

Participants: Quentin Heath, Dale Miller [correspondant].

Main web-site: http://slimmer.gforge.inria.fr/bedwyr/.

During the first half of 2013, Quentin Heath was working as an engineer on the team, supported by the BATT ADJ project funded by Inria. During that time, we worked exclusively on making improvements to Bedwyr. In particular, he made extensive and important changes to the tabling mechanism of Bedwyr, a feature of model checking systems that is capable of remembering past successful proofs (it can even support a finite failure as a successful proof of a negation). These extension allow lemmas to be used to greatly extend the scope of what can be inferred from a table. For example, if we are attempting to show that there is a winning strategy for a given board position, we would certainly like to make use of a lemma that allows one to infer that winning strategies are preserved under symmetries of the board. There are a number of design issues that go along with the design of such a tabling mechanism: for example, should one use such lemmas in a forward-chaining or backward-chaining fashion. Quentin has tested both of these options in order to collect information as to what the trade-offs would be.

We should note that Quentin Heath is now a PhD student on the team and is addressing a number of theoretical questions related to his research on Bedwyr.

5.3. Profound

Participant: Kaustuv Chaudhuri [correspondant].

Profound is a new interactive theorem proving and proof-exploration tool based on the idea of building formal proofs *without* the use of formal proof languages. The core concepts are a generalization of *deep inference* for the underlying logical formalism, and *proof-by-pointing* for the user-interaction metaphors.

A user proves a theorem in *Profound* by using the keyboard and mouse to select subformulas of the theorem and dragging them to their suitable "destination". For instance, the formula $(A \to C) \to (A \wedge B \to C)$ is proved by dragging the two As and the two Cs to each other. This kind of *direct manipulation* is nevertheless constrained by the system to be both correct—meaning that no manipulation is logically unsound—and complete—meaning that every provable theorem can be proved using these metaphors.

The system is still in its early stages, but it currently supports first-order classical linear logic. It has been documented in a paper at ITP 2013 [18].

We are in the process of extending the system to intuitionistic logics, and adding a back-end exporter for more traditional proof systems with formal proof languages such as Coq and Isabelle.

5.4. Psyche

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

Psyche (*Proof-Search factorY for Collaborative HEuristics*) is a modular proof-search engine whose first version, 1.0, was released in 2012:
http://www.lix.polytechnique.fr/~lengrand/Psyche/

Its 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.

Psyche's proof-search mechanism is simply the incremental construction of proof-trees in the polarized and focused sequent calculus. Its architecture organizes 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.

In 2013, major developments were achieved in Psyche, which now handles classical propositional logic *modulo a theory* such as linear arithmetic, equality with uninterpreted symbols, arrays, etc. It therefore works in the same logic as Sat-Modulo-Theories (SMT) solvers and the architecture to handle such theories is the main contribution of 2013, in particular with the integration of the *simplex* algorithm.

Thanks to a plugin that simulates the behavior of a SAT-solver (DPLL) [21], the new version of Psyche can now simulate the behavior of SMT-solvers.

A lot of features inspired by SAT-solvers have now been lifted to proof-search in general, such as a *memoization table* to record and re-use known proofs, the technique of *2-watched literals* to efficiently propagate direct consequences of new hypotheses, machine learning techniques for *restart policies*, etc.

Psyche has been the topic of the 2013 publication [23].

PI.R2 Project-Team

4. Software and Platforms

4.1. COQ (http://coq.inria.fr)

Participants: Bruno Barras [Inria 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 [SpecFun team, Saclay], Julien Narboux [University of Strasbourg], Jean-Marc Notin [Ecole Polytechnique], Christine Paulin [Proval team, Saclay], Pierre-Marie Pédrot, Loïc Pottier [Marelle team, Sophia], Matthias Puech, Yann Régis-Gianas, François Ripault, Matthieu Sozeau, Arnaud Spiwack [Abstraction team, ENS], Pierre-Yves Strub [IMDEA, Madrid], Enrico Tassi [SpecFun team, Saclay], Benjamin Werner [Ecole Polytechnique].

4.1.1. Version 8.5

Version 8.5 is expected to be released after the summer of 2014. It will be a major release of the Coq proof assistant, including 6 major new features:

- Parallel development and compilation, inside files and across files, by Enrico Tassi (Inria SpecFun), a result of the Paral-ITP ANR project.
- New proof engine of Arnaud Spiwack (formerly pi.r2 postdoc), more expressive and with clearer semantics.
- Native compilation by Maxime Dénès and Benjamin Grégoire (Inria Marelle, M. Dénès is now at the University of Pennsylvania). A compilation scheme from Coq to OCaml to native code, considerably improving on the previous virtual machine implementation by B. Grégoire.
- A Universe Polymorphic extension by Matthieu Sozeau that allows universe-generic developments, as required by the Homotopy Type Theory library for example.
- Primitive projections for records by Matthieu Sozeau.
- A new document generation system by F. Ripault and Yann Régis-Gianas.

A more detailed description of all the new features will be made in next year's report, but some elements can already be found below.

4.1.2. Evaluation algorithms

Pierre Boutillier has worked on the practical implementation of the unfolding algorithm for global constants that he proposed last year, so that it could become the default process to simplify terms by tactics. A formal presentation has been written in his PhD, to be defended in February 2014.

4.1.3. Internal representation of projections

The change of representation for record projections implemented by Matthieu Sozeau will be part of the 8.5 release. It provides not only exponential gains in performance (type-checking and comparison time and space usage) but also a better basis to work with canonical structures in the unification algorithm, allowing to improve for example the ssreflect inference mechanism significantly. Benchmarks on the HoTT library and a groupoid model construction confirm the exponential gain in performance.

4.1.4. Universes

Matthieu Sozeau followed up his work on universe polymorphism and uncovered important theoretical problems regarding conversion and unification of universe polymorphic constants in the presence of cumulativity and the Prop $\leq$ Type rule. After a careful study of the alternative solutions, he designed a practical correction for the issue and developed a paper proof of conservativity of the complete new system over the original theory of Coq. A paper describing this work has been submitted. The universe polymorphic system, already in use by the HoTT community, will be part of the upcoming 8.5 release.

4.1.5. The Equations plugin

Matthieu Sozeau continued work on the Equations plugin and fixed the remaining bugs preventing full automation of a middle-size example of formalization of the normalization proof of a simply-typed lambda calculus. Wojciech Jedynak, a student of Dariusz Biernacki and Małgorzata Biernacka at the University of Wroclaw, is working under his supervision to do the remaining work before an official release of the plugin can be done. Wojciech Jedynak applied for a 4-month internship supervised by Matthieu Sozeau on an extension of Equations, to start in March 2014.

4.1.6. Internal architecture of the Coq software

With the help of many others, Pierre Letouzey organized in November 2013 the migration of the official Coq source repository from subversion to git. The native use of this decentralized version control system eases the exchange of code amongst Coq developpers (either from the Coq dev team or from external contributors).

Pierre Letouzey, Pierre-Marie Pédrot and Xavier Clerc have continued to work at improving the quality of the OCaml code which composes Coq :

- Many modules have been revised, in particular with cleaner naming convention.
- Almost all uses of the generic OCaml comparison has been chased and transformed into specific code. This avoided many potential bugs with advanced structures, while improving performances at the same time.
- The codes handling OCaml exceptions have been reworked to avoid undue interceptions of critical exceptions.
- Issues involving exceptions are now quite simpler to debug, thanks to easy-to-obtain backtraces.

4.1.7. Efficiency

Pierre-Marie Pédrot has been working on the overall optimization of Coq, by tracking hotspots in the code. Coq trunk is currently much more efficient than its v8.4 counterpart, and is about as quick as v8.3, while having been expanded with a lot of additional features.

Pierre Letouzey has improved the representation of Coq binary files : these files are now smaller (thanks to more sharing), and are reloaded quickly by Coq (thanks to deferred loading of opaque proof terms, which are large and almost never accessed by the user).

4.1.8. 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.

4.1.9. General maintenance

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

4.1.10. Modules in Coq

In 2013, Pierre Letouzey has proposed an important rework of the code implementing the module system of Coq. This code was inherited from the successive works of several PhD students, and was in pretty poor shape. While being equivalent in terms of features for the user, the new code should be quite more readable and robust, as well as more efficient: the memory sharing of modular structures should be better, leading to reduced memory footprint for Coq as well as smaller Coq compiled files.

4.1.11. The Coq extraction

Pierre Letouzey has collaborated with colleagues with the aim of extending the extraction tool to additional target languages:

- `C++` with Gabriel Dos Reis and his student Robert Schumacher
- `F#` with David Monniaux

These experiments have been quite promising. In the case of `C++`, an article has been written, it should be re-submitted soon for publication.

4.1.12. Formalisation in Coq

Hugo Herbelin's type-theoretic construction of semi-simplicial sets [22] has been formalised in Coq.

Matthieu Sozeau and Nicolas Tabareau formalised a groupoid model in Coq http://github.com/mattam82/groupoid.

Jaime Gaspar has verified in Coq the correctness of Jean-Louis Krivine's proof that Zermelo-Fraenkel set theory without choice ZF is contained in a variant ZF_ε (useful for Krivine's classical realisability).

4.2. Other software developments

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

Yann Régis-Gianas started the development of the "Hacking Dojo", a web platform to automatically grade programming exercises.

In a different vein, Jaime Gaspar showed that is not always possible to get cross-references right in LaTeX by presenting a LaTeX file where a cross-reference is always wrong (no matter how many times we compile the file).

SUMO Team

5. Software and Platforms

5.1. Sigali

Participants: Hervé Marchand, Nicolas Berthier.

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/TEA and SUMO 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. Sigali is connected with the Polychrony environment (Espresso/Tea 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 under the identification number IDDN.FR.001.370006.S.P.1999.000.10600.

Sigali is also integrated as part of the compiler of the language BZR (web site).

We are currently developing a new version of Sigali that will be able to handle numerical variables.

5.2. Tipex

Participants: Thierry Jéron, Hervé Marchand, Srinivas Pinisetty.

We are implementing a prototype tool named Tipex (TImed Properties Enforcement during eXecution) for the enforcement of timed properties, in collaboration with Ylies Falcone (LIG, Grenoble). Tipex is based on the theory and algorithms that we develop for the synthesis of enforcement monitors for properties specified by timed automata (TA) [45] (see Subsection 6.2.3). The prototype is developped in python, and uses the PyUPPAAL and DBMpyuppaal libraries of the UPPAAL tool. It is currently restricted to safety and co-safety timed property. The property provided as input to the tool is a TA that can be specified using the UPPAAL tool, and is stored in XML format. The tool synthesizes an enforcement monitor from this TA, which can then be used to enforce a sequence of timed events to satisfy the property. Experiments have been conducted on a set of case studies. This allowed to validate the architecture and feasibility of enforcement monitoring in a timed setting and to have a first assessment of performance (and to what extent the overhead induced by monitoring is negligible).

5.3. SOFAT

Participants: Loïc Hélouët, Rouwaida Abdallah.

SOFAT is the acronym for Scenario Oracle and Formal Analysis Toolbox. As this name suggests it is a formal analysis toolbox for scenarios. Scenarios are informal descriptions of behaviors of distributed systems. SOFAT allows the edition and analysis of distributed systems specifications described using Message Sequence Charts, a scenario language standardized by the ITU [54]. The main functionalities proposed by SOFAT are the textual edition of Message Sequence Charts, their graphical visualization, the analysis of their formal properties, and their simulation. The analysis of the formal properties of a Message Sequence Chart specification determines if a description is regular, local choice, or globally cooperative. Satisfaction of these properties allows respectively for model-checking of logical formulae in temporal logic, implementation, or comparison of specifications. All these applications are either undecidable problems or unfeasible if the Message Sequence

Chart description does not satisfy the corresponding property. The SOFAT toolbox implements most of the theoretical results obtained on Message Sequence Charts this last decade. It is regularly updated and re-distributed. The purpose of this software is twofold: provide a scenario based specification tool for developers of distributed applications; serve as a platform for theoretical results on scenarios and partial orders SOFAT provides several functionalities, that are: syntactical analysis of scenario descriptions, formal analysis of scenario properties, interactive simulation of scenarios when possible, and diagnosis. See also the web page.

This year, SOFAT has been extended with model transformation techniques that allow to transform non-implementable HMSCs into implementable ones [49].

APP: IDDN.FR.001.080027.000.S.P.2003.00.10600
Programming language: Java

5.4. DAXML

Participant: Loïc Hélouët.

DAXML is an implementation of Distributed Active Documents, a formalism for data centric design of Web Services proposed by Serge Abiteboul. This implementation is based on a REST framework, and can run on a network of machines connected to internet and equipped with JAVA. This implementation was realized during the post doc of Benoit Masson in 2011. A demo of the software is available at this web page. We plan to maintain this prototype as a demonstrator for our Web Services activities, and to distribute the sources.

TOCCATA Team

5. Software and Platforms

5.1. The CiME rewrite toolbox

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

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], Jean-Christophe Filliâtre, Guillaume Melquiond, Andrei Paskevich.

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, École Polytechnique, etc.), used by several research groups in the world, e.g at Fraunhofer Institute in Berlin [93], at Universidade do Minho in Portugal [54], 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], Claude Marché, Guillaume Melquiond, Andrei Paskevich.

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

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/.

[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 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.

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 [113] 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)

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.

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.

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

Participant: Jean-Christophe Filliâtre [contact].

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 [56].

5.9. Functory

Participant: Jean-Christophe Filliâtre [contact].

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 [92] and at TFP 2011 [91].

5.10. The Pff library

Participant: Sylvie Boldo [contact].

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 [64].

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.

Criteria for Software Self-Assessment: A-2, SO-3, SM-3, EM-3, SDL-5, 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 [6]. 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].

Criteria for Software Self-Assessment: A-3, SO-4, SM-4, EM-3, SDL-5, 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 [83] [119].

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].

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).

5.14. The Alea library for randomized algorithms

Participant: Christine Paulin-Mohring [contact].

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 used as a basis of the Certicrypt environment (MSR-Inria joint research center, Imdea Madrid, Inria Sophia-Antipolis) for formal proofs for computational cryptography [59]. It is also experimented in LABRI as a basis to study formal proofs of probabilistic distributed algorithms. ALEA version 8 distributed in May 2013 includes a module to reason with random variables with values in positive real numbers.

5.15. The Coccinelle library for term rewriting

Participant: Évelyne Contejean [contact].

Coccinelle is a *Coq* library for term rewriting. Besides the usual definitions and theorems of term algebras, term rewriting and term ordering, it also models some of the algorithms implemented in the CiME toolbox, such 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 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.

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

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 an *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.

VERIDIS Project-Team

5. Software and Platforms

5.1. The veriT solver

Participants: David Déharbe, Pablo Dobal, Haniel Barbosa, 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 over 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 at the veriT Web site.

Efforts in 2013 have been focused on efficiency, and more specifically on arithmetic. A preliminary prototype integrating the solver Redlog for non-linear arithmetic has been stabilized. First results are encouraging; this prepares the ground for the starting ANR project SMArT (Satisfiability Modulo Arithmetic Theories), involving both sites of the VeriDis team (veriT being developed in Nancy and Redlog being designed in Saarbrücken), as well as Systerel as an industrial partner.

In late 2013, Haniel Barbosa joined the team as a PhD student. He will work on theoretical and practical aspects of handling quantifiers in SMT frameworks, which is currently an important challenge for SMT, and he will implement his techniques in veriT.

We target applications where validation of formulas is crucial, such as the validation of TLA$^+$ and B specifications, and work together with the developers of the respective verification platforms to make veriT even more useful in practice. The solver is available as a plugin for the Rodin platform for discharging proof obligations generated in Event-B [39]; 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.

5.2. The TLA+ proof system

Participants: Bhargav Bhatt, Stephan Merz [correspondent], Hernán Vanzetto.

TLAPS, the TLA$^+$ proof system, is a platform for developing and mechanically verifying proofs about TLA$^+$ specifications. It is developed at the Joint MSR-Inria Centre. The TLA$^+$ proof language is hierarchical and explicit. 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.

The current version 1.2.1 of TLAPS was released in September 2013, it is distributed under a BSD-like license at http://tla.msr-inria.inria.fr/tlaps/content/Home.html. The prover currently handles the non-temporal part of TLA$^+$ and can 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, developed in Nancy, has been further improved in 2013 and is now considered by users as the most useful backend prover for system verification. During his internship in the summer of 2013, Bhargav Bhatt helped design and implement a standard library of TLA$^+$ theorems about functions, sequences, and finite sets that is now part of the TLAPS distribution. Development of support for temporal reasoning in TLAPS has started in late 2013.

CARTE Project-Team

5. Software and Platforms

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.
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.

CASSIS Project-Team

5. Software and Platforms

5.1. Protocol Verification Tools

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

5.1.1. AVISPA

Cassis has been 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 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 [77]) allows *CL-AtSe* to be correct and complete. Each protocol step is represented by a constraint on the protocol state, used to check for reachability of the next state. *CL-AtSe* includes a proper handling of sets, choice points, specification of any attack states through a language for expressing e.g. secrecy, authentication, fairness, or 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).

CL-AtSe has been successfully used [65] 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).

Cl-Atse has been enhanced in various ways. In particular, the tool fully supports Aslan semantics introduced in [63], including Horn Clauses (for intruder-independent deductions, e.g. for credential management), and LTL-based security properties. Also, bug information and correction are processed through a bugzilla server, and online analysis and orchestration are available on our team server (https://cassis.loria.fr). Cl-Atse supports negative constraints on the intruder's knowledge [66]. This extension of Cl-Atse allows us to reduce drastically the orchestrator's processing times. It has also been used to model e.g. separation of duties and non-disclosure policies. We have also extended the syntax and semantics of ASLan to better model lists of undefined length, directly inside messages. Cl-AtSe tool now supports membership predicates, deletion operators ans so on for managing these lists, and offers a first reference implementation for other tools in Avantssar. In particular, the ASLan translator has been updated by our partners.

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

5.1.3. Akiss

We develop the *Akiss* (Active Knowledge in Security Protocols) tool for verifying indistinguishability properties in cryptographic protocols. Indistinguishability properties are essential in formal verification of cryptographic protocols. They are needed to model anonymity properties, strong versions of confidentiality and resistance against offline guessing attacks, which can be conveniently modeled using process equivalences. *Akiss* implements a procedure to verify equivalence properties for a bounded number of sessions of cryptographic protocols. As in the applied pi-calculus, the protocol specification language is parametrized by a first-order sorted term signature and an equational theory which allows formalization of algebraic properties of cryptographic primitives. *Akiss* is able to verify trace equivalence for determinate cryptographic protocols. On determinate protocols, trace equivalence coincides with observational equivalence which can therefore be automatically verified for such processes. When protocols are not determinate *Akiss* can be used for both under- and over-approximations of trace equivalence, which proved successful on several examples. The procedure can handle a large set of cryptographic primitives, namely those that can be modeled by an optimally reducing convergent rewrite system.

The underlying procedure is based on a fully abstract modelling of the traces of a bounded number of sessions of the protocols into first-order Horn clauses on which a dedicated resolution procedure is used to decide equivalence properties. Although termination of the resolution procedure has not been proved, the procedure has been effectively tested on examples, some of which are outside the scope of other existing tools, including checking anonymity in several electronic voting protocols.

Recent developments include the possibility for checking everlasting indistinguishability properties. This feature was added when analyzing everlasting privacy properties in electronic voting protocols. We are currently working on a generalization of the procedure to allow associative-commutative operators and in particular a re-design of the resolution procedure for allowing analysis of protocols that use exclusive or. Expected case studies for this development include unlinkability in RFID protocols.

The *Akiss* tool is freely available at https://github.com/ciobaca/akiss.

5.1.4. Belenios

In collaboration with the Caramel team, we develop an open-source private and verifiable electronic voting protocol, named *Belenios*. Our system is an evolution of an existing system, Helios, developed by Ben Adida, and used e.g. by UCL and the IACR association in real elections. The main differences with Helios are the following ones:

- In Helios, the ballot box publishes the encrypted ballots together with their corresponding voters. This raises a privacy issue in the sense that whether someone voted or not shall not necessarily be publicized on the web. Publishing this information is in particular forbidden by the CNIL's recommendations. Belenios no longer publishes voters' identities, still guaranteeing the correctness of the tally.
- Helios is verifiable except that one has to trust that the ballot box will not add ballots. The addition of ballots is particularly hard to detect as soon as the list of voters is not public. We have therefore introduced an additional authority that provides credentials that the ballot box can verify but not forge.

This new version has been implemented by Stéphane Glondu and has been tested in July 2013 in a mock election in the teams Cassis and Caramel.

In a first step, *Belenios* has been implemented as an extension of existing Helios system. However, the existing software development of Helios is large and its security becomes difficult to assess. We have therefore re-implemented entirely the code of the bulletin box, yielding a now independent software [2].

[2] http://belenios.gforge.inria.fr/

In Helios as well as *Belenios*, votes are encrypted using the public key of the election. To ensure privacy, the corresponding decryption key is not known to anyone. Instead, several authorities detain a share of it. For robustness reasons (and as recommended by the CNIL), it is important to be able to decrypt even if some of the authorities are missing. We have implemented the threshold decryption scheme that we have proposed [40]. This implementation is currently available only within the Helios system and we plan to integrate it to *Belenios* in the next months.

5.2. Testing Tools

Participants: Fabrice Bouquet, Frédéric Dadeau, Kalou Cabrera.

5.2.1. Hydra

Hydra is an 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://members.femto-st.fr/sites/femto-st.fr.frederic-dadeau/files/content/pub/jMuHLPSL.jar.

5.3. Collaborative Tools

Participant: Abdessamad Imine.

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.

5.3.1. P2PEdit

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

5.3.2. 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.

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

COMETE Project-Team

5. Software and Platforms

5.1. Location Guard

Participants: Konstantinos Chatzikokolakis [correspondant], Marco Stronati.

The purpose of *Location Guard* is to implement obfuscation techniques for achieving location privacy, in a an easy and intuitive way that makes them available to the general public. Various modern applications, running either on smartphones or on the web, allow third parties to obtain the user's location. A smartphone application can obtain this information from the operating system using a system call, while web application obtain it from the browser using a javascript call.

Although both mobile operating systems and browsers require the user's permission to disclose location information, the user faces an "all-or-nothing" choice: either disclose his exact location and give up his privacy, or stop using the application. This forces many users to disclose their location, although ideally they would like to enjoy some privacy.

The API level of a browser or an operating system would be an ideal place for integrating a location obfuscation technique, in a way that is easy to understand for the average user, and readily available to all applications. When an application asks for the user's location, the browser or operating system can ask the user's permission, but including the option to provide an obfuscated location instead of the real one! Different levels of obfuscation can be also offered, so that the user can chose to provide more accurate location to applications that really need it, and more noisy location to those that don't.

A prototype of Location Guard has been already implemented for Google Chrome. In the future we plan to extend it to other desktop and mobile browsers (Firefox, Internet Explorer, etc), as well as to implement it in modern mobile operating systems, primarily on Android.

https://github.com/chatziko/location-guard

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 [29] 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 [30] 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

Participant: Catuscia Palamidessi [correspondant].

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 [32] and [31].

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

DICE Team

5. Software and Platforms

5.1. GPeer: a peer-to-peer javascript communication library

Our software development has been oriented towards systems working in browsers, with the support of an Inria ADT project in cooperation with the ASAP team. To answer our technological objectives, we are working on decentralized architectures, browser to browser, developed in javascript/HTML5. We rely on the WebRTC JavaScript protocol proposed by Google to develop a communication layer between peers. Many peer-to-peer protocols share common elements, that we group in a generic library for developing peer-to-peer systems. The joint library developed with the ASAP team handles any gossip based communication overlay. We design peer messages, tracker management and resilient behavior. The library is a standard bridge between complex browser to browser applications and low level networking layers such as WebRTC. With the use of our library, we can reproduce systems such as bitTorrent, but also provide new applications without the need of either native applications or identified servers.

5.2. Fluxion: a software plugin for flows in AngularJS

The joint project with Worldline aims at managing mobile code in complex Web architectures. Load variation in data-centers is currently poorly resolved. Most of the time, systems overestimate resource consumption in order to absorb burst usage. These consumption overestimation has a cost both in terms of the SLA negotiated with the client and the non-availability of reserved resources. With Wordline we focus on code mobility for high performance Web architectures and design a fast and reactive framework, transparently moving functions between running systems. The Fluxion model is our approach to design mobile application modules that are a mix of functional programming and flow based reactive systems.

5.3. BitBallot: a decentralized voting protocol

The BitBallot voting protocol is designed to target large scale communities. The protocol allows users to share only restricted amounts of their data and computation with central platforms as well as other peers. Convinced by the need of new election mechanisms, to support emerging forms of more continuous democracy, we are developing BitBallot, to allow elections to be organized independently of any central authority. The protocol guarantees the following properties, anonymity of the data sources, non interruptible run-time, global access to results, and non predictability of results through partial communication spying.

PRIVATICS Team

4. Software and Platforms

4.1. Mobilitics

Mobilitics is a joint project, started in 2012 between Inria and CNIL, which targets privacy issues on smartphones. The goal is to analyze the behavior of smartphones applications and their operating system regarding users private data, that is, the time they are accessed or sent to third party companies usually neither with user's awareness nor consent.

In the presence of a wide range of different smartphones available in terms of operating systems and hardware architecture, Mobilitics project focuses actually its study on the two more widespread mobile platforms which are IOS(Iphone) and Android.

Indeed, both versions of Mobilitics software should provide these common requirements: Be able to capture any event about private data access such as User location, Device Unique Identifier, Address Book... Store these events in a local database on the phone for offline analysis Send this local database to Mobilitics server for privacy leakage statistics

A Mobilitics prototype for Iphone has been developed since January 2012 at Privatics. It has already embedded the features listed above and much more. However, a separate prototype for Android has been also developed since September 2012 fulfilling the same equirements listed above because IOS and Android are different in either software or hardware level.

Indeed, some live experiments have been conducted by CNIL with Mobilitics prototype for IOS with the help of volunteers equipped with iphones which they have used for a period of four(4) months(September 2012-January 2013). As a result, some visualization tools have been developed for the data collected in order to showcase private data leakage by the apps which the participants of the experiment have used. Therefore, a press conference has been held by CNIL in Paris in April 2013 during which Mobilitics results for Iphone have been published onto several French newspapers (see Section 8.3)

Likewise, some live experiments will be conducted on Android this year in February 2014 for at least three(3) months with volunteers equipped with Galaxy Nexus smartphones on which Mobilitics will be deployed. As a consequence, a press release by CNIL will be scheduled for the publication of the results obtained for Android with a perspective of comparing Google privacy policy to Apple one.

PROSECCO Project-Team

5. Software and Platforms

5.1. ProVerif

Participants: Bruno Blanchet [correspondant], Xavier Allamigeon [April–July 2004], Vincent Cheval [Sept. 2011–], Benjamin 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. Cryptosense Analyzer

Participants: Graham Steel [correspondant], Romain Bardou.

See also the web page http://cryptosense.com.

Cryptosense Analyzer (formerly known as 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. Cryptosense Analyzer 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. It has been used to find at least a dozen previously unknown flaws in commercially available devices.

In June 2013 we submitted a patent application (13 55374) on the reverse engineering process. We also concluded a license agreement between Inria PROSECCO and the nascent spin-off company Cryptosense to commercialize the tool.

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 was published at IEEE S&P 2013, and two updates to the software were released in 2013. The software and associated research materials are available from http://mitls.rocq.inria.fr.

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.