Khalil Ghorbal

Simulating and Verifying Dynamical and Hybrid Systems

The presentation will highlight two specific challenges when it comes to simulating and verifying dynamical and hybrid systems.
The first is the automated generation and verification of invariant properties, that are those quantities that do not vary with respect to time.
To some extent, this first challenge can be thought of as a generalization of the perhaps more familiar notion of fixed-points in the standard settings of imperative programs.
The second challenge concerns the operational semantics of component-based modeling languages. Those languages are well suited to model multi-physics systems as they combine algebraic constraints, resulting from the laws of physics, in interaction with the non-smooth behavior like impact laws. Yet, the correct simulation of such hybrid models poses several subtle problems such as solving under and over-determined systems.

Eric Goubault

A Topological Method for Finding Invariant Sets of Switched Systems

We revisit the problem of finding controlled invariants sets (viability), for a class of differential inclusions, using topological methods based on Wazewski property. In many ways, this generalizes the Viability Theorem approach, which is itself a generalization of the Lyapunov function approach for systems described by ordinary differential equations. We give a computable criterion based on SoS methods for a class of differential inclusions to have a non-empty viability kernel within some given region. We use this method to prove the existence of (controlled) invariant sets of switched systems inside a region described by a polynomial template, both with time-dependent switching and with state-based switching through a finite set of hypersurfaces.

Nicolas Halbwachs

Disjunction of polyhedra

Program analysis by abstract interpretation using relational abstract domains --- like polyhedra or octagons --- easily extends from state analysis (construction of reachable states) to relational analysis (construction of input-output relations). In this paper, we exploit this extension to enable interprocedural program analysis, by constructing relational summaries of procedures. In order to improve the accuracy of procedure summaries, we propose a method to refine them into disjunctions of relations, these disjunctions being directed by preconditions on input parameters.

Maxime Jacquemin

Relative error-bound estimation

Rigorous estimation of bounds on errors in finite precision computation has become a key point of many formal verification tools. The primary interest of the use of such tools is generally to obtain worst-case bounds on the absolute errors. However, the natural bound on the elementary error committed by each floating-point arithmetic operation is a bound on the relative error, which suggests that relative error bounds could also play a role in the process of computing tight error estimations.
In this work, we introduce a very simple interval-based abstraction, combining absolute and relative error propagations. We demonstrate with a prototype implementation how this simple product allows us in many cases to improve absolute error bounds, and even to often favorably compare with state-of-the art tools, that rely on much more costly relational abstraction or optimization-based estimations.

Valentin Montmirail

REGAR / CEGAR with Sat Solvers

Counter-Example-Guided Abstraction Refinement (CEGAR) has been very successful in model checking. Since then, it has been applied to many different problems. It is especially proved to be a highly successful practical approach for solving the PSPACE complete QBF problem. In this paper, we propose a new CEGAR-like approach for tackling PSPACE complete problems that we call RECAR (Recursive Explore and Check Abstraction Refinement). We show that this generic approach is sound and complete.

Marie Pelleau \& Ghiles Ziat

AbSolute (demo)

AbSolute is a constraint solver based on abstract domains. The source code is available on GitHub. This solver can solve continuous problems, discrete problems, but also mixed problems (containing both discrete and continuous variables). AbSolute uses Apron, an Ocaml library for abstract domains, one can thus solve problems using abstract domains other than intervals. Note that, abstract domains can also correspond to a product of abstract domains.

Christian Schulte

Unison 101

This talk and demo show how Unison improves code generation in compilers by using constraint programming (CP) as a method for solving combinatorial optimization problems. It presents how register allocation (assigning program variables to processor registers) and instruction scheduling (reordering processor instructions to increase throughput) can be modeled and solved using CP. Unison is significant as its addresses the same aspects as traditional code generation algorithms, yet is based on simple models and can robustly generate better code.

Christian Schulte

Efficient constraint propagation engines

How idempotence reasoning and events help track fixpoints more accurately?
Blackboard presentation of the key idea in the paper Efficient constraint propagation engines, Christian Schulte, Peter J. Stuckey, ACM Trans. Program. Lang. Syst. 31(1): 2:1-2:43 (2008).

Pierre Talbot

Lattice framework for Constraints solving

Constraint solvers are highly hierarchical as the notions of domain, constraint and search tree depend on each other.We formulate a lattice theory of constraint programming that defines a hierarchy of lattices where each level is built on the powerset of the previous level.
As far as we know, our theory is the first attempt to formalize the structures underlying the inference and search components in a unified framework.Then, we attempt to situate abstract domain in this hierarchy in order to draw links between constraint programming and abstract interpretation.

Ghiles Ziat

Finding solutions by finding non-solutions

In continuous constraint programming, the solving process alternates propagation steps, which reduce the search space according to the constraints, and branching steps. In practice, the solvers spend a lot of computation time in propagation to separate feasible and infeasible parts of the search space. The constraint propagators cut the search space into two subspaces: the inconsistent one, which can be discarded, and the consistent one, which may contain solutions and where the search continues. The status of all this consistent subspace is thus indeterminate. In this article, we introduce a new step called elimination. It refines the analysis of the consistent subspace by dividing it into an indeterminate one, where the search must continue, and a

satisfied one, where the constraints are always satisfied. The latter can be stored and removed from the search process. Elimination relies on the propagation of the negation of the constraints, and a new difference operator to efficiently compute the obtained set as an union of boxes, thus it uses the same representations and algorithms as those already existing in the solvers. Combined with propagation, elimination allows the solver to focus on the frontiers of the constraints, which is the core difficult part of the problem. We have implemented our method in the AbSolute solver, and present experimental results on classic benchmarks with good performances.