Bounded Model Checking and Inductive Verification of Hybrid Discrete-continuous Systems

Bernd Becker[1], Markus Behle[2], Fritz Eisenbrand[2], Martin Fränzle[3]
Marc Herbstritt[1], Christian Herde[4], Joerg Hoffmann[1], Daniel Kröning[5]
Bernhard Nebel[1], Ilia Polian[1], Ralf Wimmer[1]

[1]Institut für Informatik, Albert-Ludwigs-Universität Freiburg
[2]Max-Planck-Institut für Informatik Saarbrücken
[3]Informatics and Mathematical Modelling Dept., DTU, Kgs. Lyngby, Denmark
[4]Department für Informatik, Carl v. Ossietzky Universität Oldenburg
[5]School of Computer Science, Carnegie Mellon University, Pittsburgh, USA

Abstract

We present a concept to significantly advance the state of the art for bounded model checking (BMC) and inductive verification (IV) of hybrid discrete-continuous systems. Our approach combines the expertise of partners coming from different domains, like hybrid systems modeling and digital circuit verification, bounded planning and heuristic search, combinatorial optimization and integer programming. After sketching the overall verification flow we present first results indicating that the combination and tight integration of different verification engines is a first step to pave the way to fully automated BMC and IV of medium to large-scale networks of hybrid automata.

Keywords: Hybrid System verification, Bounded model checking, Inductive verification, Verification engines

1 Introduction

Many embedded systems operate within or even comprise coupled networks of both discrete and continuous components. The behavior of such hybrid discrete-continuous systems cannot be fully understood without explicitly modeling the interaction of discrete and continuous dynamics. Tools for building such models and for simulating their dynamics are commercially available, e.g. Simulink with the Stateflow extension [1, 2] or Statemate MAGNUM with the VisSim extension [3]. Simulation is, however, inherently incomplete and has to be complemented by verification, which amounts to showing that the coupled dynamics of the embedded system and its environment is well-behaved, e.g. that it may never reach an undesirable state or that it will converge to a desirable state, regardless of the actual disturbance. Unfortunately, theories and tool support for verifying hybrid systems are not yet mature. Recent industrial trials, e.g. performed by Ford in the context of the Mobies initiative[1], indicate that current verification tools fall short with respect to both the dimensionality of the continuous state spaces and the size of the discrete state spaces they can handle.

[1]Mobies phase 1 tool evaluation report, Mobies project, July 2002

On the other hand, during the last ten years, formal verification of digital systems has evolved from an academic subject to an approach accepted by the industry, with dozens of commercial tools now available and used by major companies. Among the most successful methods in formal verification of discrete systems are Bounded Model Checking (BMC) and Inductive Verification (IV) [11, 19, 10, 12, 26, 30]. In general, BMC, i.e., checking properties on finite unravellings of the transition relation, and IV, i.e., verifying invariance of properties under finite composition of the transition relation, provide only approximate analyzes of the systems considered. On the other hand, both verification methods using BDD based as well as resolution based propositional satisfiability checkers have been successfully applied to very large discrete-state systems that are infeasible for proof engines based on complete methods. As mentioned above, similar technology for hybrid systems is still in its infancy. In general we observe a large gap between the systems that can be specified and those that can be verified.

Based on these observations, the German Research Council (DFG) approved the Transregional Collaborative research center AVACS (Automatic Verification and Analysis of Complex Systems; `www.avacs.org`) between the Universities of Oldenburg, Freiburg and Saarbrücken, and the Max-Planck-Institut für Informatik Saarbrücken. Researchers from Technical University of Denmark, Kgs. Lyngby (Denmark) and Eidgenössische Technische Hochschule Zürich (Switzerland) are involved as associated partners. The AVACS project consists of three project areas: system-level verification (project area S); verification of real-time systems (project area R), and verification of hybrid systems (project area H). The technology described in this paper corresponds to the sub-project H2 in project area H, which is one of four sub-projects in this project area. The other three sub-projects are concerned with: deduction and automata based approaches (H1), automatic abstraction (H3) and hybrid system stability (H4). The sub-project H2 sets out to advance the state of the art in BMC and IV for hybrid systems by enhancement of several existing verification methods for the hybrid domain and development of novel ones. Moreover, we combine stand-alone methods into a tightly integrated framework. This will finally result in a platform including translators, tightly integrated SAT engines and selection heuristics thereby making feasible the verification of medium to large scale networks of hybrid automata.

In this paper we describe the concept how we plan to achieve these aims and on the other hand also give first results to show that our approach is very promising.

The remainder of the paper is structured as follows: In Sec. 2 we give the overall verification flow and in particular discuss basic procedures necessary for the development of the platform. Combining the expertise of partners coming from different domains, like hybrid systems modeling and digital circuit verification, bounded planning and heuristic search, combinatorial optimization and integer programming, essential milestones towards our overall goals are *compilation technology necessary for translating the hybrid system and the verification goal into SAT representations* (Sec. 3) and *incremental integration of DPLL, bounded planning and LP procedures into a solver for many-sorted logics* (Sec. 4). In Sec. 5 we present first results indicating that the combination and tight integration of different verification engines is a promising approach. We sketch the relation between and combination of "classical" ILP solvers and BDD based ILP solvers. The concept of a pseudo-Boolean DPLL (Davis-Putnam-Loveland-Logemann) solver and its combination with (MI)LP is discussed in 5.1. Finally we summarize and draw some conclusions.

2 Goals and Requirements

We aim at advancing scalability of BMC and IV for large hybrid verification problems. The first step towards this goal is the translation of the system description from the

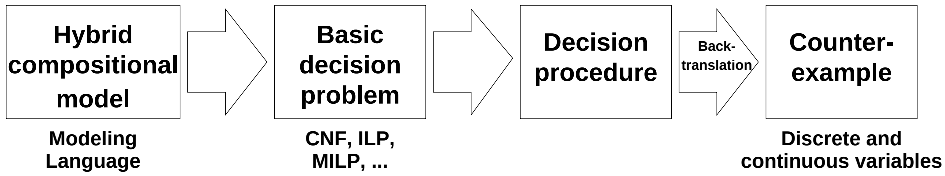

Figure 1: Flow of BMC/IV for hybrid systems

hybrid discrete-continuous domain into a combined SAT/LP instance. Then, our target is the tight integration of different verification and analysis techniques for satisfiability and optimization problems in order to combine their virtues. The overall verification flow is depicted in 1. (Details on the several components are given in the following sections.)

Basic procedures used to realize this concept are:

Linear programming is well-suited for dealing with constant differential inclusions but does not cover piecewise constant differential inclusions or discrete-continuous interaction.

Resolution based propositional satisfiability checking efficiently tackles control aspects of discrete models but falls short on, e.g., integer arithmetic.

Integer linear programming is much better on discrete arithmetic facts (whenever they fall into its domain), but not the predominant choice for the control part.

Heuristic search is a method for rapidly finding satisfying valuations in complex domains.

Binary decision diagrams are data structures that can represent the complete proof tree. For instance, they are capable of automatically constructing invariants of discrete components, a prerequisite for efficient automatic inductive verification.

A tight integration of such heterogeneous computational procedures will provide a major breakthrough in the performance of verification engines for hybrid systems, as witnessed by existing, less ambitious, combinations, e.g. the linear programming based and resolution based planner LPSAT [33] or satisfiability engines dedicated to bounded model checking of timed automata [5, 28].

Our long-term vision is to provide a complete workbench comprising translators, multiple tightly integrated satisfiability engines, and selection heuristics invoking these such that BMC and IV can be fully automated for medium to large-scale networks of hybrid automata. This will in particular involve an extension of the many-sorted propositional satisfiability procedures phase to first-order logics, together with methods for (approximate and thus computationally cheap) quantifier elimination facilitating automatic abstraction refinement.

3 Translation and Backtranslation

We address the translation of hybrid systems modeled as interacting hybrid automata into satisfiability problems. In industrial practice, hybrid systems are frequently modeled as highly concurrent compositions of heterogeneous signal transducers. Instrumental to breakthrough in the scalability of push-button verification methods for such systems is to

1. avoid blow-up when translating such descriptions into a predicative format suitable for symbolic model checking,
2. find suitable encodings for different types of components such that efficient SAT solving procedures can be applied.

We contribute to the state of the art in this domain by exploiting the formal semantics of interacting hybrid automata for providing encodings of component and interface dynamics in various logical languages, and verifying their correspondence with the formal semantics. Depending on the type of component, such encodings may involve just one of the aforementioned formalisms (e.g. a propositional formula in case of pure control logic) or a conjunction of different encodings with some common variables (e.g. a propositional formula plus an MILP containing some zero-one variables).

A novel technique to be explored is the idea of *redundant encodings*, where abstractions of different fidelity coexist in a satisfiability problem. While this may appear to just introduce overhead in the encoding, as the tightest overapproximation appearing in the satisfiability problem subsumes all constraints arising from the laxer approximations, it is beneficial in a satisfiability search based on combined engines. The most expensive operation within a combined satisfiability solver (e.g., a DPLL based propositional SAT procedure combined with a linear programming package) is the hand-over between the two procedures. An instance of a (potentially) useful redundant encoding is a finite-state overapproximation of continuous dynamics in addition to an LP-based continuous-state representation. In this case, the DPLL procedure may perform inferences and refute existence of certain traces without hand-over to the LP package; the latter will only be queried when the discrete approximation turns out to be insufficient. Similar savings on hand-over cost can be expected between other satisfiability components. With consistent use of such "redundant" encodings, it may be possible to combine even expensive (semi-)decision procedures, like quantifier elimination for elementary geometry or flow-pipe approximations for ordinary differential equations , into a heterogeneous decision procedure for hybrid systems, as these expensive procedures would rarely be activated. To the best of our knowledge, such a technique has not yet been implemented. In its full generality, i.e. including above-mentioned expensive analysis procedures and activating these whenever knowledge on the coarser levels turns out to be insufficient, it could provide a viable alternative to abstraction refinement, where refinement is substituted by conflict analysis (in a subordinated decision procedure) and learning (in the calling procedure).

Given this kind of translation, the solvers described in Section 4 are processing a piecewise linear overapproximation of the actual hybrid system. As a consequence, an error trace identified by the solver is inherently approximate. To "backtranslate" this information into valid error traces of the actual system is a non-trivial task. This problem will be reduced to a sequence of search problems, where the search space is the space of values of the input variables for one state transition. For each such state transition, goal conditions are generated from the values in the abstract error trace, which are tried to be reached using local search methods. The idea is to search in the transition system given by the automaton to be checked, starting in the initial state of the attacked search problem, and using solutions to a relaxed version of the problem as a measure of the distance to the nearest state fulfilling the goal condition. Search can then prefer states that appear to be closer to the goal. Techniques of this kind have been applied very successfully in the context of planning [14, 24, 23].

4 Core Technology: Tightly Integrated Solvers

This section provides an overview on the solvers that perform the actual verification on the data translated from the hybrid domain by the methods described above. The core of the approach is the DPLL-style satisfiability engine, which invokes further routines, such as ILP, when needed. The DPLL solver core is discussed next, followed by a description of how ILP methods can be fitted to the hybrid system verification environment. Some thoughts on the tight integration conclude this section.

4.1 DPLL solver

In both BMC and IV, the main reasoning is done by deciding satisfiability problems. In our context these are heterogeneous satisfiability problems, i.e. problems that combine discrete and continuous aspects. The best currently known procedures to decide satisfiability of discrete problems are based on backtracking in the space of partial value assignments to the decision variables, using constraint propagation to prune search branches when conflicts occur. In particular, variations of the Davis-Putnam-Loveland-Logemann (DPLL) procedure (which propagate the constraints given by unit clauses) remain the state of the art solvers for satisfiability of Boolean formulae in CNF. Recent results [17, 21], suggest good transferability of DPLL techniques to more expressive logics than propositional, e.g. to pseudo-Boolean formulae.

Our solver will incorporate all major components employed by the state of the art SAT engines in the Boolean domain such as branching heuristics and conflict analysis and adapt them appropriately for the SAT/LP instances under consideration. We also will develop techniques that take the structure of the problem description into account (similar to structural SAT for CNF or ATPG for digital circuits). To fit the needs of bounded model checking and inductive verification the solver will be designed to work in an incremental fashion in the sense that it allows to add (as well as delete) successively sets of constraints to (from) an existing problem and then redo the satisfiability check without starting SAT search from scratch each time.

4.2 SAT & Planning

Planning is a sub-field of Artificial Intelligence where, in a declaratively specified transition system, a path – a plan – is sought that leads from the initial configuration to a state that satisfies a goal condition. Evidently, this problem is very similar to the search for an error path in model-checking. First successful attempts have been made to transport BDD-based model-checking techniques into planning [8, 9]. Just like it is done in bounded model-checking, a possible approach to planning (called bounded planning) is to perform (DPLL-style) satisfiability tests on finite unravellings of the transition relation. It was recently observed in bounded planning that, within the DPLL solver that does the reasoning, "relaxation-based" branching heuristics can by far outperform the traditional criticality-based branching heuristics [22]. Our hope is that the same will hold true in the satisfiability instances we will consider in our research. A relaxation is a simplified version of the transition system under consideration. The idea is, during a DPLL-style search, to extract a relaxed solution (an error path in the relaxed system) in each search state (respecting the current search decisions), and base the variable selection on the information provided by this relaxed solution. In bounded planning, the relaxation ignores certain restrictive aspects of the possible state transitions. Similar relaxations can be defined in the context of model-checking, e.g. by abstraction techniques or LP relaxations. DPLL search can then pick one (most critical) variable to branch on that corresponds to a transition used by the relaxed solution.

Of course, extracting a relaxed solution is likely to be more costly than the traditional criticality-based branching heuristics, which basically keep counters of literal occurrences in clauses. One can experiment with incremental computation of the relaxed solutions, with lazy schemes that extract relaxed solutions only in few search states, and with DPLL-style procedures that use strong constraint propagation techniques (like, binary instead of unary resolution) and thus invest more effort into the single search nodes anyway. In bounded planning, the latter option has been taken, i.e. relaxation-based branching heuristics have successfully been combined with a rather costly planning-specific constraint propagation technique [22].

4.3 LP-SAT

The LP-SAT routine deals with the satisfiability of (mixed integer) linear programs. A new approach to mixed integer satisfiability (SAT/LP) is introduced by incorporating BDD techniques in a branch-and-cut framework. Moreover, the information generated by our methods can be used for guiding the decision process of the DPLL procedure. The reasons for inconsistency of an MILP (which can be determined by calculating an *infeasible subsystem*) can be employed for pruning the search space of the combined DPLL/LP procedure. The efficiency of this pruning approach is closely connected to the conciseness of the reason supplied, i.e. to the minimality of the computed infeasible subsystem of the MILP. In order to cope with difficulties induced by numerical instabilities of an LP procedure, we incorporate result verification techniques based on exact arithmetic.

4.4 Integration

To allow the integration of various kinds of first-order solvers within the SAT solver, the latter will offer a flexible and carefully designed interface which enables the interaction between the different engines. The basic idea of the integration itself is to replace each non-propositional constraint in the input formula with a new Boolean variable and to pass the corresponding constraint to the mathematical decision procedure whenever the SAT solver assigns that variable true. The first-order solver, working under control of the SAT engine, decides the feasibility of the set of constraints residing in its database when being called and reports a conflict to the SAT solver if the conjunction of constraints turns out to be inconsistent. In the latter case it derives sufficiently general reasons for that conflict which are communicated back and learned by the SAT solver to avoid repeated failure due to these reasons in future search.

The integration scheme sketched above is eager in the sense that the first order solver is invoked after each assignment of a Boolean variable referring to a non-propositional constraint. As an alternative we will also consider more lazy schemes which postpone certain invocations of the first-order solver, thereby avoiding the computational overhead induced by frequent hand-overs between the different engines. The design of heuristics for how fine-granular the interleaving should be will require a careful theoretical and empirical investigation of the trade-off between frequent cache invalidations because of fine-granular interleaving of different algorithms on the one hand, and redundant interference steps that could have been avoided through early detection of inconsistencies in another engine if that engine had been activated more frequently, on the other hand.

5 First Steps

5.1 ILP & BDD

A 0/1-ILP problem is given by a matrix $A \in \mathbb{Z}^{m \times n}$, and vectors $b \in \mathbb{Z}^m$ and $c \in \mathbb{Z}^n$. A vector $x \in \{0,1\}^n$ is looked for, which satisfies the *constraints* $Ax - b \leq 0$ and minimizes the *goal function* $g(x) := c^T x$. BDDs [15] can represent a Boolean function $f : \{0,1\}^p \to \{0,1\}$, Kronecker Multiplicative Binary Moment Diagrams (K*BMDs) [16, 20] are an extension of BDDs to functions from $\mathbb{B}$ to $\mathbb{Z}$. Our approach uses BDDs for representing the constraints and the goal function, and K*BMDs as a vehicle during build-up.

Consider the ith constraint (out of the total number of m) $C_i(x_1, x_2, \ldots, x_n)$. We build up the K*BMD of this function and transform it to a BDD of the *characteristic function* χ_i (where $\chi_i(x_1, \ldots, x_n) = 1$ iff $C_i(x_1, \ldots, x_n) \geq 0$). Then, we calculate the conjunction of the characteristic functions $\bigwedge_{i=1}^{m} \chi_i(x_1, \ldots, x_n)$. To minimze BDD size, we employ

Problem	Divide & Conquer	Binary Search	lp_solve
stein9	0.02 s	0.03 s	0.01 s
stein15	0.07 s	0.04 s	0.02 s
stein27	2.62 s	0.41 s	4.59 s
stein45	1290.90 s	127.85 s	355.03 s
p0033	0.31 s	758.97 s	0.32 s
p0040	0.10 s	—	0.03 s
bm23	14.19 s	30.18 s	0.09 s

Table 1: Run time comparison for ILP+BDD approach

dynamic BDD variable ordering optimization and also have several strategies regarding the order in which the conjunction is computed. Finally, we minimize the goal function g (represented as a K*BMD) without violating the constraints (given as the BDD of $\bigwedge_{i=1}^{m} \chi_i$). We developed two alternative approaches for the latter problem.

The first method (which is based on the algorithm from [25] but is adapted to K*BMDs instead of EVBDDs) works in a divide-and-conquer fashion. It solves the problem (recursively) for both cofactors of the function and derives the global minimum from this information. The K*BMD and the BDD are traversed simultaneously.

The second alternative is based on binary search. For this method, an upper bound U and a lower bound L of the minimum are needed. Trivial values (such as 0 and sum of all non-negative coefficients of the ILP instance, respectively) can be used as U and L. More preferably, some known algorithm should be used to determine tighter bounds.

We set $M := \lfloor (U + L)/2 \rfloor$. Then, we add $g(x_1, \ldots, x_n) \leq M$ as a new constraint and try to find an assignment of $x_1, \ldots x_n$ which satisfies all constraints (the original ones as well as the new one) *without minimizing any goal function*. If this succeeds (a solution $(x'_1, \ldots x'_n)$ has been delivered), then we set the upper bound U to $g(x'_1, \ldots x'_n)$, otherwise we set the lower bound L to $M + 1$. We calculate a new M and solve the problem formulated above again (with updated new constraint). This is iterated until L and U assume the same value. The assignment found in the last iteration solves the original ILP problem.

Table 1 contains the run times for both our methods in comparison with lp_solve [7], a public domain (integer) linear programming solver, on benchmarks from MIPLIB3.0 [13]. It can be seen that the divide & conquer approach is typically either as fast as lp_solve or slower, but not dramatically.[2] The picture is different for binary search, which seems to be orthogonal to lp_solve. The reason for this performance profile is as follows: once all the BDDs are built up, the satisfiability check needed for the binary search is trivial and is done quickly. On the other hand, the build-up of the BDDs is complicated by adding the new constraint, which typically depends on all or almost all variables of the problem in contrast to most other constraints. Consequently, binary search is efficient for the instances with a goal function having a compact BDD representation. This is the case for the stein problems, which are highly symmetric.

Currently, we implement a solver, which combines BDD techniques with branch-and-cut. Preliminary results with this approach are encouraging.

[2] Note that the implementation of both our methods is in an early prototype stage.

5.2 DPLL-SAT & ILP

To integrate DPLL proof search with decision procedures for real arithmetic, we have concentrated on:

1. generalizing acceleration techniques from recent satisfiability engines for CNFs to linear constraint systems over the Booleans, and

2. coupling the resulting SAT solver with linear programming, resulting in an efficient solver for mixed Boolean-linear problems as well as for massively disjunctive (MI)LPs.

The rationale behind step 1 is that rewriting the propositional formulae occurring in e.g. bounded model checking of discrete-state systems [11] to CNF requires a blow-up in either the formula size (worst-case exponential) or in the number of propositional variables (linear, leading to a worst-case exponential blow-up of the search space). These blow-ups can be attributed to the low expressiveness of conjunctive normal forms, and could be partially avoided when using SAT solving on more concise logics.

It has previously been observed that the DPLL procedure generalizes smoothly to linear constraint systems over the Booleans [6, 32, 4]. Such constraint systems are expressive enough to facilitate a linear-size encoding of, e.g., gate-level netlists without use of auxiliary variables, thus avoiding the blow-up encountered upon CNF encoding. We generalized the acceleration techniques like observation lists and lazy clause evaluation [27], as well as the more traditional non-chronological backtracking and learning techniques to Davis-Putnam-like resolution procedures for linear constraint systems over the Booleans [21]. Despite the more expressive input language, the performance of our prototype implementation comes surprisingly close to that of state-of-the-art CNF-SAT engines like ZChaff [27]. Our algorithm is very close to that independently developed by Chai and Kuehlmann [17], yet avoids re-minimization of observation-set size upon backtracking. This proves to be effective, as our lazy clause evaluation scheme shows speed-ups on arbitrary clause types [21], while Chai and Kuehlmann decided, based on benchmarking their implementation, to constrain lazy clause evaluation to CNF and cardinality clauses (i.e. linear clauses where all literals have weight 1) [18].

Step (2) is our first instance of a tight integration of DPLL proof search with arithmetic decision procedures. Combining linear constraint systems over the Booleans with (MI)LPs, the many-sorted logic supported by this integrated solver is actually only marginally more expressive than MILPs, which can encode Boolean properties as well as disjunctions of bounded LPs via 0-1-variables. Nevertheless, the underlying solver technology is substantially different from traditional LP solvers (e.g., Simplex or interior point algorithms), such that their synergistic interaction is feasible. New techniques for resolving e.g. the problems of traditional LP methods on highly symmetric problems—a standard situation in model-checking of multi-component systems—can be implemented.

Our work on integrating our DPLL-style SAT solver with linear programming packages has recently led to a first operational prototype. In contrast to the methods from [33, 5, 29, 31], our system is based on a SAT solver manipulating a considerably more concise logic, namely linear constraint systems over the Booleans instead of CNF. Our solver has been extended with the concept of Boolean variables guarding arithmetic facts[3] and with an interface to arithmetic decision procedures that allows to incrementally construct and deconstruct an arithmetic constraint system via calls to the decision procedure's API. The solver has been coupled with library GLPK[4], which provides the feasibility check for

[3] i.e., if such a guard variable is set to true then the corresponding arithmetic fact is conjoined to an initially empty arithmetic constraint system

[4] `http://www.gnu.org/software/glpk/glpk.html`

(MI)LPs (yet, currently not the construction of small infeasible subsystems that can be handed back to the DPLL procedure via the interface, and used there to learn a Boolean conflict). The resulting interaction between DPLL proof search and feasibility check via LP is illustrated in Figure 2.

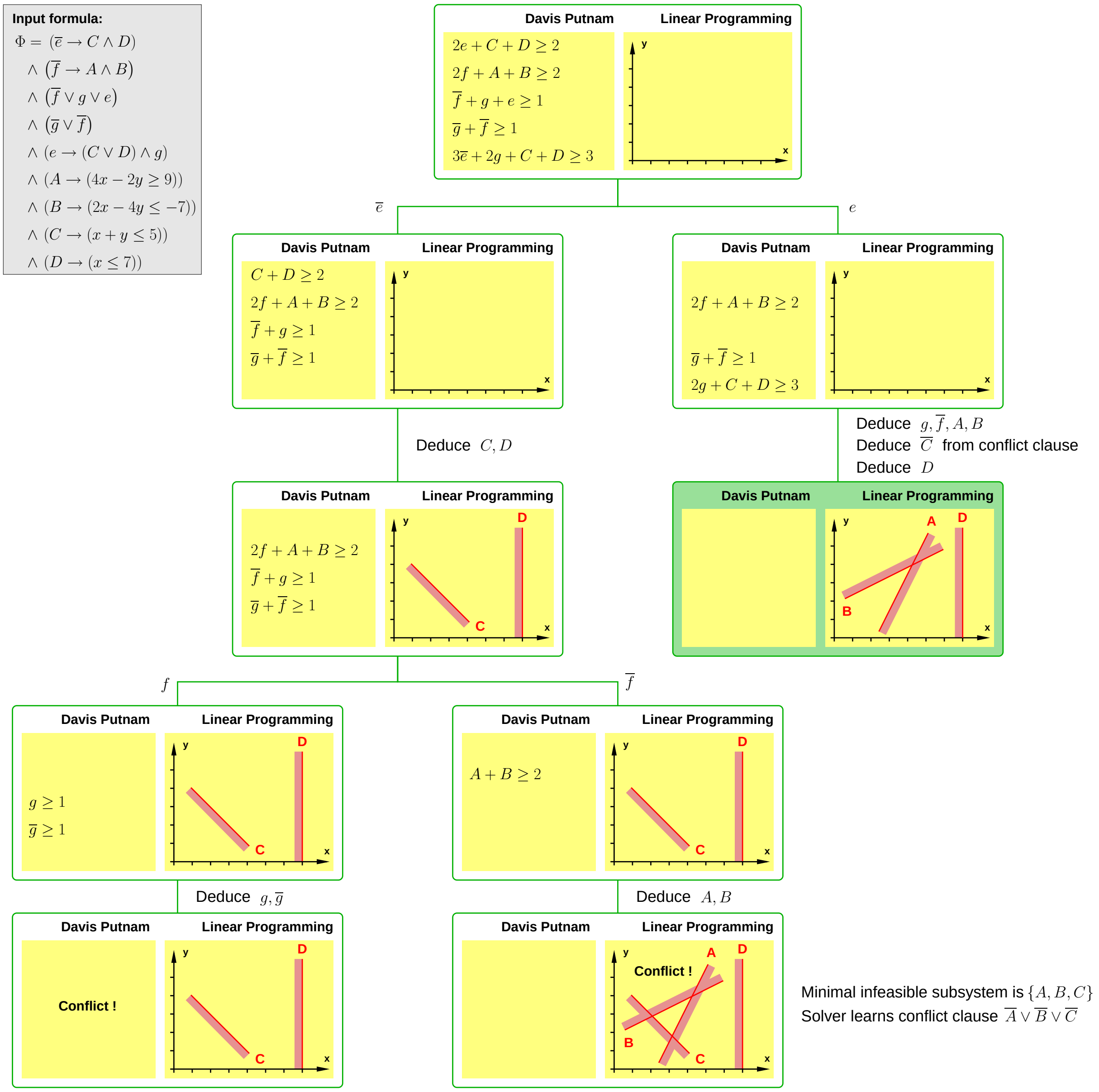

Figure 2: DPLL proof search on many-sorted (here: Boolean and continuous) problems. x and y are real-valued, while e, f, g and A, B, C, D are Boolean. A, B, C, D are, furthermore, guard variables for arithmetic facts.

6 Conclusions

During the last ten years, formal verification of digital systems has evolved from an academic subject to an approach accepted by the industry, with dozens of commercial tools now available and used by major companies. Among the most successful methods in formal verification of discrete systems are Bounded Model Checking (BMC) and Inductive

Verification (IV). This is not yet the case for verification of hybrid systems, where in general we observe a large gap between the systems that can be specified and those that can be verified.

Our goal is to make BMC and IV applicable to hybrid systems of realistic size and thus to make these method accessible for the industry. Our approach is based on translating the verification problem under consideration into a Boolean satisfiability instance enriched by numerical information; processing the problem using tightly integrated solvers; and "backtranslating" the error trace into the hybrid domain. Our team combines skills and competence in such different areas of Computer Science as Propositional Satisfiability, (Integer) Linear Programming, Heuristic Search, and Decision Diagrams. First results demonstrate that for some of these fields, incorporating concepts and ideas from other areas can increase the efficiency of standard methods. We are confident that tight integration in a way that takes the specifics of hybrid system modeling and verification into account will make this task as feasible as formal verification of digital systems is today.

7 References

[1] `http://www.mathworks.com/products/simulink/`.

[2] `http://www.mathworks.com/products/stateflow/`.

[3] `http://www.ilogix.com/products/magnum/statemate_magnum.pdf`.

[4] Fadi A. Aloul, Arathi Ramani, Igor L. Markov, and Karem A. Sakallah. Generic ILP versus specialized 0-1 ILP: An update. In *Int'l Conf. on CAD*, pages 450–457, November 2002.

[5] G. Audemard, P. Bertoli, A. Cimatti, A. Kornilowics, and R. Sebastiani. A SAT-based approach for solving formulas over boolean and linear mathematical propositions. In A. Voronkov, editor, *Automated Deduction — CADE-18*, volume 2392 of *Lecture Notes in Computer Science*, pages 193–208. Springer-Verlag, 2002.

[6] Peter Barth. A Davis-Putnam based enumeration algorithm for linear pseudo-boolean optimization. Technical Report MPI-I-95-2-003, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 1995.

[7] M. Berkelaar, J. Dirks, K. Eikland, and P. Notebaert. lp-solve. ftp://ftp.ics.ele.tue.nl/pub/lp_solve/.

[8] Piergioirgio Bertoli, Alessandro Cimatti, John Slaney, and Sylvie Thiebaux. Solving power supply restoration problems with planning via symbolic model checking. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI-02)*, pages 576–80.

[9] Piergiorgio Bertoli, Alessandro Cimatti, Marco Roveri, and Paolo Traverso. Planning in nondeterministic domains under partial observability via symbolic model checking. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 473–478.

[10] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *DAC 99*. ACM, 1999.

[11] A. Biere, A. Cimatti, and Y. Zhu. Symbolic model checking without BDDs. In *TACAS'99*, volume 1579 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

[12] A. Biere, E. M. Clarke, R. Raimi, and Y. Zhu. Verifying safety properties of a PowerPC microprocessor using symbolic model checking without BDDs. In *Computer Aided Verification (CAV '99)*, volume 1633 of *Lecture Notes in Computer Science*. Springer-Verlag, 1999.

[13] R. E. Bixby, S. Ceria, C. M. McZeal, and M. W. P. Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. Submitted to SIAM News, March 1996.

[14] Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1–2):5–33, 2001.

[15] R.E. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[16] R.E. Bryant and Y.-A. Chen. Verification of Arithmetic Functions with Binary Moment Diagrams. In *IEEE Design Automation Conference*, pages 535–541, 1995.

[17] Donald Chai and Andreas Kuehlmann. A fast pseudo-boolean constraint solver. In *Proc. of the 40th Design Automation Conference (DAC 2003)*, pages 830–835, Anaheim (California, USA), June 2003. ACM.

[18] Donald Chai and Andreas Kuehlmann. A fast pseudo-boolean constraint solver. In *Proc. of the 40th Design Automation Conference (DAC 2003)*, pages 830–835, Anaheim (California, USA), June 2003. ACM.

[19] Fady Copty, Limor Fix, Rana Fraer, Enrico Giunchiglia, Gila Kamhi, Armando Tacchella, and Moshe Y. Vardi. Benefits of bounded model checking at an industrial setting. In G. Berry, H. Comon, and A. Finkel, editors, *Computer Aided Verification (CAV 2001)*, volume 2102 of *Lecture Notes in Computer Science*, pages 436–453. Springer-Verlag, 2001.

[20] R. Drechsler, B. Becker, and S. Ruppertz. The K*BMD: A Verification Data Structure. *IEEE Design and Test*, pages 51–59, 1997.

[21] Martin Fränzle and Christian Herde. Efficient SAT engines for concise logics: Accelerating proof search for zero-one linear constraint systems. In Andrei Voronkov Moshe Y. Vardi, editor, *Proceedings LPAR '03*, volume 2850 of *Lecture Notes in Computer Science (subseries LNAI)*, pages 302–316. Springer-Verlag, 2003.

[22] Jörg Hoffmann. Branching matters: Alternative branching in graphplan. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS-03)*, pages 22–31, Trento, Italy, 2003. Morgan Kaufmann.

[23] Jörg Hoffmann. The Metric-FF planning system: Translating "ignoring delete lists" to numerical state variables. *Journal of Artificial Intelligence Research*, 2003. Special issue on the 3rd International Planning Competition, to appear.

[24] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[25] Y.-T. Lai, M. Pedram, and S.B.K. Vrudhula. EVBDD-based algorithms for integer linear programming, spectral transformation, and functional decomposition. *IEEE Trans. on CAD*, 13(8):959–975, 1994.

[26] Magnus Ljung. Formal modelling and automatic verification of lustre programs uisng np-tools. Master's thesis, Royal Institute of Technology, Dpt. of Teleinformatics, Sweden, 1999.

[27] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, June 2001.

[28] Peter Niebert, Moez Mahfoudh, Eugene Asarin, Marius Bozga, Oded Maler, and Navendu Jain. Verification of timed automata via satisfiability checking. In Ernst-Rüdiger Olderog and Werner Damm, editors, *International Symposium on Formal Techniques in Real-Time and Fault-Tolerant systems (FTRTFT 2002)*, volume 2469 of *Lecture Notes in Computer Science*, pages 224–244. Springer-Verlag, 2002.

[29] R. Sebastiani. Integrating SAT solvers with math reasoners: Foundations and basic algorithms. Technical Report 0111-22, ITC-IRST, November 2001.

[30] M Sheeran, S. Singh, and G. Stråmark. Checking safety properties using induction and a SAT-solver. In Warren A. Hunt Jr. and Steven D. Johnson, editors, *FMCAD*, volume 1954 of *Lecture Notes in Computer Science*, pages 407–420. Springer, 2000.

[31] A. Stump, C. Barrett, and D. Dill. CVC: a cooperating validity checker. In *14th International Conference on Computer-Aided Verification*, 2002.

[32] Jesse Whittemore, Joonyoung Kim, and Karem Sakallah. SATIRE: A new incremental satisfiability engine. In *Proc. of the 38th Design Automation Conference (DAC 2001)*, pages 542–545, Las Vegas (Nevada, USA), June 2001.

[33] Steven A. Wolfman and Daniel S. Weld. The LPSAT engine & its application to resource planning. In Thomas Dean, editor, *Proc. 16th International Joint Conference on Artificial Intelligence*, pages 310–315. Morgan Kaufmann Publishers, 1999.