Tool-support for the analysis of hybrid systems and models

Andreas Bauer Markus Pister Michael Tautschnig
Institut für Informatik, Technische Universität München
{baueran, pister, tautschn}@informatik.tu-muenchen.de

Abstract

This paper introduces a method and tool-support for the automatic analysis and verification of hybrid and embedded control systems, whose continuous dynamics are often modelled using MATLAB/Simulink. The method is based upon converting system models into the uniform input language of our efficient multi-domain constraint solving library, ABSOLVER*, which is then used for subsequent analysis. Basically,* ABSOLVER *is an extensible SMT-solver which addresses mixed Boolean and (nonlinear) arithmetic constraint problems as they appear in the design of hybrid control systems. It allows the integration and semantic connection of various domain specific solvers via a logical circuit, such that almost arbitrary multi-domain constraint problems can be formulated and solved. Its design has been tailored for extensibility, and thus facilitates the reuse of expert knowledge, in that the most appropriate solver for a given task can be integrated and used. As such the only constraint over the problem domain is the capability of the employed solvers. Our approach to systems verification has been validated in an industrial case study using the model of a car's steering control system. However, additional benchmarks show that other hard instances of problems could also be solved by* ABSOLVER *in respectable time, and that for some instances,* ABSOLVER*'s approach was the only means of solving a problem at all.*

1 Introduction

Hybrid and embedded control systems in general impose a challenge for automatic verification and validation due to the complex interplay with and often at the pace of their environment, and the continuous, nonlinear system artifacts to be considered. However, verification and validation tasks of such systems are often translatable into a multi-domain satisfiability problem, and can then be tackled using dedicated solvers. In a nutshell, this particular class of problems constitutes a Boolean combination of arithmetic formulae building up constraints over the logical model of the mixed-domain problem. Checking as to whether a Boolean combination of linear equations is satisfiable or not has gotten a lot of attention in recent years due to the availability of highly efficient SAT-solvers that are able to process hundreds of thousands of Boolean variables and clauses within an instant, but especially the emerging field of *satisfiability modulo theories* (SMT) delivered promising results in this area [1, 3, 8].

A possible representation of Boolean-linear satisfiability problems is as follows. Let $V = \{v_0, \ldots, v_m\}$, with $m \in \mathbb{N}_0$, be a set of variables that are instantiated by $\alpha : V \to \mathbb{B}$ in the usual way. Further, let $A = \{\sum_{i=0}^{n} a_i x_i \gtrless c \mid a_i, x_i, c \in \mathbb{R}, 0 \leq i \leq n\}$, with $n \in \mathbb{N}_0$, be the set of linear arithmetic expressions, where $\gtrless \in \{<, >, \leq, \geq, =\}$. A valuation function $\delta : A \to \mathbb{B}$ assigns a Boolean value to each linear arithmetic expression from A, depending on whether or not the comparison is satisfiable. A formula F with the usual Boolean connectives over V, constitutes a *mixed Boolean-linear system*, if each of its occurring variables $v \in V$ is substitutable by an expression from A.

An algorithmic approach to solve an instance of F is to first split the mixed problems into individual constituents. Next, a SAT-solver is queried for the solution of the Boolean constituent, returning a valid (or possibly empty) assignment α to all the variables $v \in V$ occurring in F. As each linear expression $a \in A$ occurring in F is associated with a variable $v_a \in V$ the valuations of δ are implied:

$$\forall a \in A : \quad \delta(a) \Leftrightarrow \alpha(v_a).$$

This makes up the linear constraint system, where each element $a \in A$ or the complement $\neg a$ must be satisfied. In case $a \equiv \sum_{i=0}^{n} a_i x_i = c$ is an equation and $\neg a$ is implied by α, either $\sum_{i=0}^{n} a_i x_i < c$, or $\sum_{i=0}^{n} a_i x_i > c$ must be satisfiable. The resulting system of linear (in)equalities is tested by a linear solver, and satisfiability of the overall system is shown by iterating this process until a solution is found, or all possible assignments have been shown infeasible.

1.1 Contribution

Despite the impressive results on tackling this particular class of multi-domain satisfiability problems [3, 9], there

exist many verification problems which cannot be expressed using this kind of formalism due to the natural limitations of linear arithmetic. ABSOLVER addresses these problems

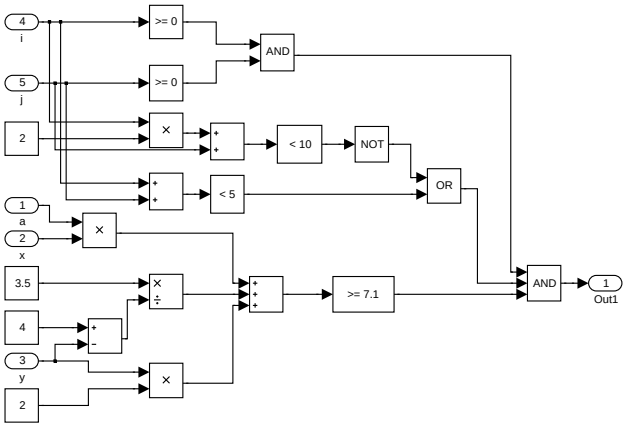

Figure 1. A MATLAB/Simulink example model.

by providing means for solving both linear as well as a set of nonlinear problems, referred to as $\mathcal{AB}$-problems (see Sec. 2), with a single engine. Moreover, due to its extensibility, it benefits from individual state-of-the-art solvers that can be integrated at ease. As such, it does not rely on a fixed algorithm for a fixed class of problems, as is the case with most other mixed-domain solvers (see Sec. 1.2).

Consequently, we formally introduce the class of $\mathcal{AB}$-problems that can be tackled by ABSOLVER, and show how various reasoning tasks can be expressed in terms of an $\mathcal{AB}$-problem. The class $\mathcal{AB}$ comprises, in addition to Boolean and linear constraints, continuous and nonlinear constraints.

To combine the different domains, we have developed a straightforward input syntax which integrates seamlessly into standard DIMACS format used by most modern SAT-solvers, i. e., apart from the Boolean clauses, we parse custom extensions to a comment line. Thus, our format is still understood by any Boolean solver not aware of the extensions. As a result, we provide a highly customisable solution for solving multi-domain satisfiability problems without having to study the internals of various third-party solvers.

The capabilities of our input language are best described by a straightforward example, using the MATLAB/Simulink model depicted in Fig. 1 as an input model. Such models are a common representation of complex behaviour of reactive and embedded systems with respect to their environment or the physical processes they control. Fig. 2 then shows how such a model can be expressed in terms of our input language (for an automatic conversion, see Sec. 3), where frames are used for emphasising the correspondence between Boolean variables and arithmetic constraints as induced by the model. The arithmetic constraints are encoded in the comment-lines starting with `c`, whereas the Boolean system consisting of four Boolean variables $(1 \dots 4)$ is encoded on top. For example, the Boolean variable 3 is associated with the constraint $i + j < 5$, and so forth. Note that the mathematical representation on the top right is not part of the input language and appears here only for explanation.

```
p cnf 4 3
1 0                                          ((i ≥ 0) ∧ (j ≥ 0))
-2 [3] 0      ∧ (¬(2i + j < 10) ∨ [( i + j < 5)])
4 0                                  ∧ (a · x + 3.5/(4−y) + 2y ≥ 7.1)

c def int 1 i >= 0
c def int 1 j >= 0
c def int 2 2*i + j < 10
c def int [3] [i + j < 5]
c def real 4 a * x + 3.5 / ( 4 - y ) +
2 * y >= 7.1
```

Figure 2. The $\mathcal{AB}$-problem given by Fig. 1 and ABSOLVER's representation.

ABSOLVER can use standard components like zChaff [7] or COIN [5] for solving their part of the problem, as well as any other solver. Due to its internal bookkeeping it is able to compute *all* models for a given satisfiability problem, should it be required either to help resolve conflicts or if more than one solution is requested by the user, should it be required to decide some real-world problem. The various constituents of our solver are customisable via command line parameters, say, to allow the use of specific heuristics.

In the following we give first results from having used ABSOLVER as an efficient validation vehicle for reactive systems designed using MATLAB/Simulink. Moreover, we discuss the internals of ABSOLVER and describe how it can be integrated in already existing modelling tools, e. g., as a means for a validation or verification engine. ABSOLVER is available as an open source tool from `http://absolver.sf.net/`, along with benchmarks and documentation.

1.2 Related work

There exist various multi-domain solvers, in particular, for the above described class of Boolean-linear problems; one of the most well-known being MathSAT [3]. MathSAT integrates both a Boolean as well as a linear solver and benefits from a tight integration of its constituents. However, it does not cater for integration of third-party solvers and is

not suited for other problem classes, such as imposed by nonlinear equation systems. Nonlinear equation systems play a crucial role in the verification of systems this paper is concerned with. For instance, the physical environment of embedded control systems is often approximated by complex, continuous differential equations whose (automatic) solution imposes nonlinear constraint problems. Other multi-domain solvers, such as CVC lite [1] target such nonlinear problems, and work similar to MathSAT, in that they offer integrated specialised solvers, but in practice their limitations are not always obvious to the users of such systems (see Sec. 5). Moreover, we found in our industrial research projects that commonly used verification suites, such as SCADE which can also be used in combination with MATLAB/Simulink designs of reactive systems [4], do not address $\mathcal{AB}$-problems. As such, only a specific subset of a model may be validated using these tools, or additional linearisation must be performed before validation.

The arbitrary combination of discretionary solvers described in this paper, combined with a uniform interface in a single library is, to the best of our knowledge, not provided by any other solving framework at the moment. However, we are aware of at least one ongoing research project whose technical infrastructure appears somewhat similar to our approach, but whose focus rests on the tight integration and optimal distribution of domain-specific problems and corresponding solvers [10]. At the time of writing, the technical infrastructure of this project was in its early stages and, therefore, not reflected in our comparative benchmarks.

1.3 Outline

The rest of the paper is structured as follows. The next section gives a brief introduction to the class of $\mathcal{AB}$-problems, which can be tackled by our solver. In Sec. 3, we discuss the feasibility of $\mathcal{AB}$-problems in the real world by presenting first results from an industrial case study, validating a car's steering control system modelled with MATLAB/Simulink. Sec. 4 then describes ABSOLVER's internals as well as the interface provided to the user of the system. Sec. 5 contains comparative benchmarks for ABSOLVER with respect to a number of standard test suites (SMT-LIB) as well as a number of artificial multi-domain problems, imposing linear as well as nonlinear constraints. Finally, conclusions of our work are summarised in Sec. 6.

2 The $\mathcal{AB}$-satisfiability problem

The class of problems tackled by our solver is referred to as $\mathcal{AB}$ (short for arithmetic-Boolean) and characterised as follows. We first fix a set of variables V. Let $\mathcal{B} = \mathbb{B} \cup \{?\}$, and $\mathcal{A} = \{a_0 x_0 \ \mathrm{op}_1 \ldots \mathrm{op}_n \ a_n x_n \gtrless c \mid a_i, x_i, c \in \mathbb{R}, 0 \leq i \leq n\}$, with $n \in \mathbb{N}_0$ and $\mathrm{op}_i \in \{+, -, *, /\}$, defining the set of (possibly) nonlinear arithmetic expressions. The two functions α and δ extend canonically as in $\alpha' : V \to \mathcal{B}$ and $\delta' : \mathcal{A} \to \mathcal{B}$. Notice, $\mathcal{B}$ resembles a 3-valued semantics for our substitution, which is necessary as long as ABSOLVER has not determined a solution to one of its subproblems (see Sec. 4). Although, the current implementation is restricted to the well-known arithmetic expressions, extension to other operators, such as $\sin$, $\cos$ or $\exp$ is straightforward and not limited by a design decision. For instance, adding the division operator involved less than an hour of programming effort.

3 Applicability and case study

We found in our industrial cooperations that the extension of the mixed Boolean-linear problems to $\mathcal{AB}$-satisfiability-problems meets the need for finding intuitive descriptions for the behaviour of hybrid and embedded control systems along with their environment. The relevance of solving such problems can be emphasised by an example taken from a safety analysis of a car's steering control system. Amongst others, such a system obtains data from a yaw sensor ($-7.0 \leq x \leq 7.0$), a lateral acceleration sensor ($-20.0 \leq x \leq 20.0$), speed sensors at each wheel ($-400.0 \leq x \leq 400.0$), and the steering angle ($-1.0 \leq x \leq 1.0$). It is due to the controller to check whether the car is currently in a stable driving situation, or not. If the car shows a tendency towards over-steering or under-steering, the controller calculates the correct steering angle for keeping the car stabilised. The difference between the correct angle for stable driving and the current steering angle is added or subtracted by a step motor. This enables the controller to prevent some critical driving situations.

The continuous dynamics of the controller and its environment are modelled using MATLAB/Simulink, where the environment consists of nonlinear functions modelling the physical behaviour of the car. In industry, the analysis of the model focuses on testing the complete system in several test cases and in simulations. However, verification tools used in industry, such as SCADE, are not laid out for solving $\mathcal{AB}$-problems directly. Thus, the proof of properties in such a setup is commonly restricted to the linear designed controller, without looking at the whole nonlinear control-circuit which commonly includes the environment as well.

To be able to check correctness regarding a set of defined mathematical predicates, we implemented a protoype toolchain to facilitate the automated conversion (see Fig. 3) of the steering control model from MATLAB/Simulink to ABSOLVER's input format. Note that conversion takes advantage of the SCADE modelling and verification suite which can import MATLAB/Simulink models. However, using SCADE in the conversion was merely a matter of convenience, because internally, SCADE uses a textual repre-

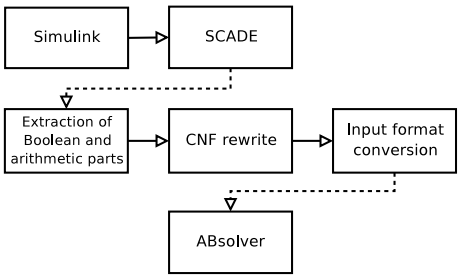

Figure 3. Automated conversion work-flow.

sentation of the model in terms of the programming language LUSTRE, from which we could then extract the multi-domain constraint satisfaction problems. This particular conversion resulted in 976 CNF-clauses, and 24 (non-) linear expressions representing the constraints. Computing a solution required less than a minute on a standard notebook using COIN (for the linear part), zChaff (for the Boolean part), and IPOPT [11] (for the nonlinear part).

4 Design and implementation of ABSOLVER

To be able to tackle such problems we aimed for an extensible design that neither limits the possible ways of problem input, nor the set of external solvers used in the various domains. The result is a layered approach which should also provide an easy starting point for programmers joining the project.

In case of the stand-alone executable, the *input layer* (see Fig. 4) is instantiated by a parser accepting an extended version of the DIMACS format. However, ABSOLVER may as well be used as a native C++ library, e. g., for building or extending existing verification tools, in which case the input must be provided directly via the ABSOLVER API.

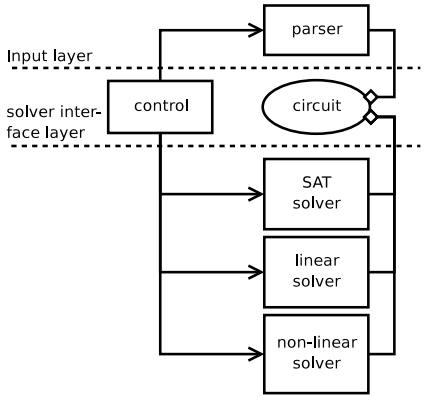

Figure 4. Architecture.

ABSOLVER's core comprises a data structure for modelling an *integrated circuit* where arithmetic and Boolean operations are represented as gates taking either a single (e. g., negation), a pair (e. g., arithmetic comparison), or an arbitrary number of inputs. The variables are then seen as the input pins of a circuit, and the single output pin provides the formula's truth value, which is either tt, ff, or ? indicating that further treatment is necessary, internally. Fig. 4 illustrates this on a subset of the example given in Sec. 1.

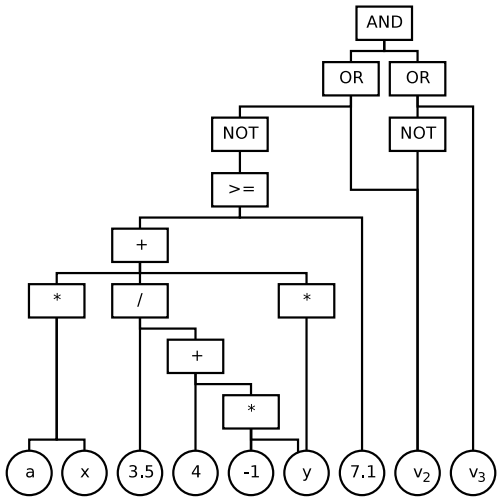

Figure 5. Internal representation.

While the input layer builds up such a circuit-object, the *solver interface layer* (see Fig. 4) uses this object for computing variable assignments. The main task of the solver interface is, thus, adapting the circuit to the external solver's input format, as well as extraction of domain specific components of the $\mathcal{AB}$-problem, e. g., extraction of linear and nonlinear equations.

To ensure extensibility to new solvers the communication between the tools is restricted to the well-defined interface that provides the circuit, a data structure for returning solutions, and a structure to support refinement of conflicts detected by a solver in response to a possible assignment computed by the Boolean solver. Although standard solvers, such as zChaff can be used with ABSOLVER, it can also make use of specialised solvers like LSAT as described in [2], which not only determines satisfiability, but is also able to provide all satisfying assignments. Hence, the use of LSAT is desirable for applications such as consistency-based diagnosis, where more than one Boolean solution may be required to reason about the failure state of systems. Most importantly, even if a SAT-solver other than LSAT is used, which is not able to determine *all* truth assignments, ABSOLVER's internal bookkeeping makes it possible to iteratively call the solver, such that, effectively, all solutions can be computed. This, however, happens at the expense of the time required for restarting the entire solving process externally. On the other hand, we currently do not provide

our own linear and nonlinear solvers.

The control loop follows the algorithm sketched in Sec. 1 and works as follows. ABSOLVER queries a SAT-solver for a single solution—or all solutions at once—for the Boolean part of the $\mathcal{AB}$-problem. The results are considered by the interface to the linear solver, and the resulting constraint system is built up, and solutions are computed by the linear solver. If infeasibility is detected, the smallest conflicting subset is computed and returned as a hint for further queries to the SAT-solver. In case the output pin's value of the circuit is not yet known (i. e., $\alpha'(\cdot) = ?$), the nonlinear solver is called. Note that at each of those steps a list of solvers is used, if more than one solver is enabled for some domain and the preceeding solvers thereof failed to provide a decent result.

5 Detailed benchmarks

Besides the industrial case study sketched in Sec. 3, we have performed a number of standard SMT benchmarks as well as combinatorial puzzles, e. g., Sudoku, imposing linear and nonlinear problems.

The presented results and the underlying test suites are also available from ABSOLVER's web site (see `http://absolver.sf.net/`), such that these can be repeated or compared with further third-party solvers.

5.1 Nonlinear problems

The following benchmarks are available for download from the ABSOLVER web site, however, excluding the original car steering model due to obvious issues with the protection of intellectual property. The additional entries in the table display the number of Boolean clauses, variables, and linear, respectively, nonlinear sub-problems. The employed solvers for ABSOLVER were IPOPT for the nonlinear, COIN for the linear, and zChaff for the Boolean part. The time required for a run is then denoted in minutes and seconds and summarised in Table 1.

Table 1. Results: nonlinear problems.

Benchmark	#Cl.	#Var.	#linear	#nonlin.	ABSOLVER
Car steering	976	24	4	20	0m58.344s
esat_n11_-m8_nonlinear	11	8	9	2	0m0.469s
nonlinear_unsat	1	1	0	2	0m0.260s
div_operator	1	1	4	1	0m0.233s

Comparative results are not available in this case, since both CVC Lite and MathSAT rejected the problems due to the nonlinear arithmetic inequalities contained, e. g., in the environment model of the car steering controller.

5.2 SMT-LIB

The benchmarks presented in this section were converted automatically to ABSOLVER's input format from the satisfiability-modulo-theories benchmark library. Their original source is `http://combination.cs.uiowa.edu/smtlib/`. The problems impose a combination of Boolean and linear problems, such as can also be tackled by solvers like MathSAT or CVC Lite. The results for ABSOLVER were created using COIN for the linear constraints and zChaff for the Boolean part. Each row in Table 2 represents one run of the benchmark performed with three different solvers.

Table 2. Results: SMT-LIB benchmarks.

Benchmark	ABSOLVER	CVC Lite	MathSAT
FISCHER1-1-fair.smt	0m0.556s	0m0.020s	0m0.045s
FISCHER2-1-fair.smt	0m0.907s	0m0.023s	0m0.095s
FISCHER3-1-fair.smt	0m2.243s	0m0.027s	0m0.177s
FISCHER4-1-fair.smt	0m3.003s	0m0.031s	0m0.281s
FISCHER5-1-fair.smt	0m2.789s	0m0.036s	0m0.422s
FISCHER6-1-fair.smt	0m5.770s	0m0.040s	0m0.604s
FISCHER7-1-fair.smt	0m10.597s	0m0.043s	0m0.791s
FISCHER8-1-fair.smt	0m14.521s	0m0.052s	0m1.031s
FISCHER9-1-fair.smt	0m18.748s	0m0.057s	0m1.343s
FISCHER10-1-fair.smt	0m22.925s	0m0.067s	0m2.913s
FISCHER11-1-fair.smt	0m28.179s	0m0.073s	0m2.129s

The results show that ABSOLVER is competetive in terms of the SMT problems with the established Boolean-linear solvers, although it does not come out as being the fastest solver. The reason for this lies in that the SMT benchmarks impose only very simple Boolean and linear problems, and in order to determine a valid linear solution, many Boolean solutions need to be examined first. The internals of MathSAT as well as CVC Lite allow a more efficient communication between the respective solvers, whereas ABSOLVER basically uses two separate entities for solving.

5.3 Sudoku

The following benchmarks were converted from daily Sudoku puzzles taken from `http://sudoku.zeit.de/`; dates indicate the magazine's respective issue and puzzle. Sudoku is a logic problem, where the human player has to arrange numbers from 1 to 9 horizontally as well as vertically in 9×9 squares, such that no numbers appear twice in a row. There are various works that describe how to

translate a Sudoku problem to a SAT-instance, e. g., [6, 12]. However, having a solver at hand which solves Boolean as well as linear problems, the Sudoku puzzle can be tackled more efficiently as a mixed problem and the encoding is more natural as it can make use of integers.

The results as shown in Table 3 using ABSOLVER were achieved with COIN for the linear part and LSAT for the Boolean part. Notice, that results marked as * indicate out-of-memory aborts.

Table 3. Results: Sudoku puzzles.

Benchmark	ABSOLVER	CVC Lite	MathSAT
2006_05_23_hard	0m0.283s	–*	84m7.385s
2006_05_24_hard	0m0.283s	–*	99m48.447s
2006_05_25_hard	0m0.282s	–*	107m0.860s
2006_05_26_hard	0m0.289s	–*	112m30.929s
2006_05_27_hard	0m0.289s	–*	89m48.470s
2006_05_28_hard	0m0.282s	–*	117m29.500s
2006_05_29_easy	0m0.279s	–*	81m27.008s
2006_05_29_hard	0m0.283s	–*	137m31.245s
2006_05_30_easy	0m0.287s	–*	75m17.435s
2006_05_30_hard	0m0.283s	–*	94m35.672s

Here, ABSOLVER clearly outperforms existing solvers due to ABSOLVER's ability to use a combination of the most *suitable* solvers for difficult instances. Basically, Sudoku problems reflect more involved integer programming sub-problems than are present in the SMT benchmarks above. The specialised selection of solvers then results in a better performance than is achieved in other all-in-one tools.

6 Conclusions

The presented approach to tackling both mixed Boolean-linear as well as nonlinear satisfiability problems has, in terms of the subclass of $\mathcal{AB}$-problems, delivered promising results. More so, for a number of problems, ABSOLVER was able to outperform existing solutions, and was used in its present form to analyse parts of an industrial case-study, i. e., that of a steering control system used in present-day cars. As such, ABSOLVER can complement modelling tools such as MATLAB/Simulink or SCADE, in that it provides means for reasoning about even complex continuous and nonlinear designs. It allows the reuse of expert knowledge by enabling users to specify the most suitable domain-specific constraint solver for a specific problem.

The benchmarks show that the use of ABSOLVER is advisable in a setting where some or all of the sub-problems are very involved. Simple instances of, e. g., Boolean-linear constraint problems are handled faster by specialists like MathSAT or CVC lite due to a tighter integration of Boolean and linear solvers.

Besides the examples presented in this paper, ABSOLVER can be used as a stand-alone tool with its intuitive-to-use input language for specifying multi-domain constraint problems, or via its C++ API for tight integration with existing code bases. Further possible use-cases of ABSOLVER include the automatic generation of test cases. Since ABSOLVER, internally, determines the solutions by computing all possible assignments, common coverage metrics like path coverage can be obtained for free in this setting. However, a more thorough examination of the types and properties of input models is then required, in order to determine sensible test criteria/cases.

References

[1] C. Barrett and S. Berezin. CVC Lite: A new implementation of the cooperating validity checker. In *CAV'04*, volume 3114 of *LNCS*. Springer, 2004.

[2] A. Bauer. Simplifying diagnosis using LSAT: a propositional approach to reasoning from first principles. In *CPAIOR'05*, volume 3524 of *LNCS*. Springer, 2005.

[3] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Schulz, and R. Sebastiani. An incremental and layered procedure for the satisfiability of linear arithmetic logic. In *TACAS'05*, volume 3440 of *LNCS*. Springer, 2005.

[4] P. Caspi, A. Curic, A. Maignan, C. Sofronis, S. Tripakis, and P. Niebert. From Simulink to SCADE/Lustre to TTA: a layered approach for distributed embedded applications. In *Proc. of the 2003 ACM SIGPLAN conference on Language, compiler, and tool for embedded systems*. ACM, 2003.

[5] R. Lougee-Heimer. The common optimization interface for operations research: Promoting open-source software in the operations research community. *IBM J. Res. Dev.*, 47(1):57–66, 2003.

[6] I. Lynce and J. Ouaknine. Sudoku as a SAT problem. In *Proc. of the Ninth International Symposium on Artificial Intelligence and Mathematics*. Springer, 2006.

[7] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proc. of the 38th Design Automation Conference*. ACM, 2001.

[8] S. Ranise and C. Tinelli. The SMT-LIB Standard: Version 1.2. Technical report, Dep. of Computer Science, University of Iowa, 2006. Available at `www.SMT-LIB.org`.

[9] H. M. Sheini and K. A. Sakallah. A SAT-based decision procedure for mixed logical/integer linear problems. In *CPAIOR'05*, volume 3524 of *LNCS*. Springer, 2005.

[10] P. J. Stuckey, M. J. G. de la Banda, M. J. Maher, K. Marriott, J. K. Slaney, Z. Somogyi, M. Wallace, and T. Walsh. The G12 project: Mapping solver independent models to efficient solutions. In *ICLP*, volume 3668 of *LNCS*. Springer, 2005.

[11] A. Wächter and L. T. Biegler. Line search filter methods for nonlinear programming: Motivation and global convergence. *SIAM Journal on Optimization*, 16(1):1–31, 2005.

[12] T. Weber. A SAT-based Sudoku solver. In *LPAR-12, Short paper proc.*, 2005.