AVACS – Automatic Verification and Analysis of Complex Systems

REPORTS

of SFB/TR 14 AVACS

Editors: Board of SFB/TR 14 AVACS

Proofs of Unsatisfiability for mixed Boolean and Non-linear Arithmetic Constraint Formulae

by
Stefan Kupferschmid Tino Teige Bernd Becker
Martin Fränzle

AVACS Technical Report No. 40
June 2008
ISSN: 1860-9821

Publisher: Sonderforschungsbereich/Transregio 14 AVACS
(Automatic Verification and Analysis of Complex Systems)
Editors: Bernd Becker, Werner Damm, Martin Fränzle, Ernst-Rüdiger Olderog,
Andreas Podelski, Reinhard Wilhelm
ATRs (AVACS Technical Reports) are freely downloadable from `www.avacs.org`

Author(s) contact: Martin Fränzle (`mf@avacs.org`).

Proofs of Unsatisfiability for mixed Boolean and Non-linear Arithmetic Constraint Formulae*

Stefan Kupferschmid[1], Tino Teige[2], Bernd Becker[1], and Martin Fränzle[2]

[1] Albert-Ludwigs-Universität Freiburg, Germany
{skupfers|becker}@informatik.uni-freiburg.de
[2] Carl von Ossietzky Universität Oldenburg, Germany
{teige|fraenzle}@informatik.uni-oldenburg.de

Abstract. In recent years, the use of constraint solvers for analysis and verification of industrial systems has become increasingly popular. The correctness and reliability of the results returned by the employed solvers are a vital requirement, in particular for safety-critical systems. If a solver proves the satisfiability of a constraint system and returns a solution then the answer of the solver can be validated by means of the solution. However, in case the solver claims that the constraint system is unsatisfiable without any certificate then the validation of this answer is not that simple. In this paper, we establish the theoretical basis of generating proofs of unsatisfiability in the mixed Boolean and non-linear arithmetic constraint framework over the reals and integers. To do so, we propose a simple rule-based calculus for unsatisfiable mixed Boolean-arithmetic constraints to produce and check unsatisfiability proofs. Furthermore, we enhance the iSAT constraint solver to generate unsatisfiability proofs, we implement a tool validating such proofs, and give some experimental results.

1 Introduction

Over the last decade, computer-aided analysis and verification of real-world systems have become increasingly popular. The application areas range from hardware verification of digital circuits to the analysis of embedded hybrid discrete-continuous systems in the car and aviation industry. A typical verification goal for some system $\mathcal{S}$ is whether $\mathcal{S}$ finally reaches some desired or undesired property P. Among the most successful verification methods is bounded model checking (BMC), as suggested by Groote et al. in [GKvV95] and by Biere et al. in [BCZ99]. The idea of BMC is to encode the system's behavior for some bounded depth k and some reachability property P as a symbolic formula φ_k s.t. the system fulfills P within depth k iff φ_k is satisfiable. For checking the satisfiability of φ_k, common BMC approaches employ underlying proof engines for solving φ_k. The choice of such solvers depends on the problem structure, e.g. a propositional satisfiability (SAT) solver is chosen for the verification of finite-state systems. For more expressive systems involving arithmetic constraints like, e.g., linear expressions or complex data-structures like, e.g., arrays, in recent years very efficient so-called satisfiability modulo theories (SMT) solvers (e.g., cf. [RT06,NOT06]) have been proposed. Facilitating satisfiability checking also for mixed Boolean and non-linear arithmetic constraint formulae over the reals and integers, the authors of [FHT+07] introduced the iSAT algorithm.

If the underlying solver proved the satisfiability of the formula φ_k then it usually delivers a solution of φ_k, e.g. a value assignment to the variables of φ_k. Thus, the BMC layer is able to verify the result of the solver by simply calculating the truth

* This work has been partially supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS, www.avacs.org).

value of φ_k under the suggested solution. However, if the solver returns that φ_k is unsatisfiable without any certificate then the BMC layer is not able to validate the result and has to trust in the solver. However, recent solvers are very complex pieces of software and we cannot guarantee in general the correctness of the tool. Contrary, guaranteed results are an absolutely necessary requirement within the verification of industrial and, in particular, safety-critical systems. Thus, there is an essential and vital industrial need for certifying also the unsatisfiability of formulae.

In 2003, Zhang & Malik [ZM03b] and Goldberg & Novikov [GN03] simultaneously published their work on proofs of unsatisfiability for propositional formulae. The approach by Zhang & Malik verifies unsatisfiability of a formula by a subsequent resolution process until the empty clause is reached. Therefore, a SAT solver was enhanced to produce a trace during the proof search, namely to output the IDs of all clauses which were involved in the generation of a conflict clause. The approach by Goldberg & Novikov is similar but is implemented the other way round. After the SAT solver stops on some formula φ with result "unsatisfiable", each conflict clause $c \in \psi$, where ψ is the set of all generated conflict clauses, is tested in counter-chronological order, whether the assignment given by $\neg c$ implies a conflict by Boolean constraint propagation with the original formula φ and the set of conflict clauses $\psi - \{c\}$ without c. Thereafter, c is removed from the set ψ. Besides checking the validity of unsatisfiability results, both papers consider the extraction of small unsatisfiable subformulae as another application field. Further work also investigates minimal and minimum unsatisfiable cores (e.g., cf. [ZM03a] and [LS04]). Such a small unsatisfiable core can potentially help to analyze why some system does not reach a desired property, e.g. when considering some planning or scheduling problems.

In the context of SMT problems, most recently Moskal [Mos08] proposed an approach to certifying unsatisfiability results of SMT problems. This work enables SMT solvers to produce and check unsatisfiability proofs for first order formulae, possibly with quantifiers, interpreted under uninterpreted function symbols, integer linear arithmetic and arrays theories.

Validating the outcome of a solver is not the only useful application domain of unsatisfiability proofs. In [McM03], McMillan exploited such proofs of a given propositional formula in order to construct Craig Interpolants [Cra57]. With the help of those, he modified a bounded model checking procedure for Kripke structures in such a way that the procedure gets unbounded, i.e. the procedure is able to show that a given system has a certain property or not. More recently, McMillan extended his work on constructing Craig Interpolants to the quantifier-free theory of linear inequalities and uninterpreted function symbols [McM05]. These generated interpolants are used in the software model checking system Blast [BHJM05].

In this paper, we first propose a simple rule-based calculus $\mathcal{RC}$ for unsatisfiable mixed Boolean and non-linear arithmetic constraint formulae φ over the reals and integers. Given a formula φ as a set of clauses, this calculus derives new clauses which are implied by the original ones. In case the empty clause can be generated the given problem φ is unsatisfiable. Then, all derived clauses together with the information of their antecedent clauses are a proof of unsatisfiability of φ. Furthermore, in order to facilitate subsequent validations of unsatisfiability results of the iSAT tool [FHT+07], we enhance iSAT to produce such unsatisfiability proofs. For validating the proofs, we implement an external proof checker which simply reproduces all single proof steps of the $\mathcal{RC}$ calculus. Thus, the main contribution of this paper is the theoretical foundation of extracting and validating unsatisfiability proofs in the mixed Boolean and non-linear arithmetic framework, and its experimental evaluation, which permit more trustworthiness, e.g., in the analysis of safety-critical systems.

Structure of the paper. In Section 2, we briefly recall the iSAT algorithm and give some necessary definitions. The $\mathcal{RC}$ calculus and the unsatisfiability proof generation are introduced in Section 3. Experimental results are provided in Section 4 while Section 5 concludes the paper.

2 Preliminaries

In [FHT+07], we introduced the iSAT algorithm for solving the in general undecidable satisfiability problem for mixed Boolean and non-linear arithmetic constraints (involving transcendental functions like sin) over real- and integer-valued variables. This algorithm is a tight integration of SAT solving based on the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [DP60,DLL62] and interval constraint propagation (ICP, cf. [BG06] for an extensive survey) enriched by enhancements like conflict-driven learning and non-chronological backtracking. A very detailed theoretical description of the algorithm and benchmark results can be found in [FHT+07]. The input constraint formulae of iSAT are arbitrary Boolean combinations of non-linear arithmetic constraints over the reals. By the front-end of our constraint solver, these constraint formulae are rewritten to linearly sized, equi-satisfiable formulae in conjunctive normal form (CNF), with atomic propositions and arithmetic constraints confined to a form resembling three-address code. This rewriting is based on the standard mechanism of introducing auxiliary variables for the values of arithmetic sub-expressions and of logical sub-formulae. Thus, the *internal* syntax[3] of constraint formulae is as follows:

$$\begin{aligned}
\mathit{formula} &::= \{\mathit{clause} \wedge\}^* \mathit{clause} \\
\mathit{clause} &::= (\{\mathit{atom} \vee\}^* \mathit{atom}) \\
\mathit{atom} &::= \mathit{simple_bound} \mid \mathit{arithmetic_predicate} \\
\mathit{simple_bound} &::= \mathit{variable}\ \mathit{relation}\ \mathit{rational_const} \\
\mathit{arithmetic_predicate} &::= \mathit{variable}\ \mathit{relation}\ \mathit{uop}\ \mathit{variable} \mid \\
&\qquad \mathit{variable}\ \mathit{relation}\ \mathit{variable}\ \mathit{bop}\ \mathit{variable} \\
\mathit{relation} &::= < \mid \leq \mid = \mid \geq \mid >
\end{aligned}$$

where *uop* and *bop* are unary and binary operation symbols, resp., including $+$, $-$, $\times$, sin, etc., and *rational_const* ranges over the rational constants.

The definition of satisfaction is standard: a constraint formula φ is satisfied by a valuation of its variables iff all its clauses are satisfied, that is, iff at least one atom is satisfied in any clause. Satisfaction of atoms is wrt. the standard interpretation of the arithmetic operators and the ordering relations over the reals. A constraint formula φ is *satisfiable* iff there exists a satisfying valuation, called *solution*, of φ. Otherwise, φ is *unsatisfiable*. We remark that by definition of satisfaction, a formula φ including or implying the empty clause, denoted $\emptyset$, cannot be satisfied at all, i.e. if $\emptyset \in \varphi$ or $\varphi \Rightarrow \emptyset$ then φ is unsatisfiable. Instead of real-valued valuations of variables, our constraint solving algorithm manipulates interval-valued valuations $\rho : \mathit{Var} \to \mathbb{I}_{\mathbb{R}}$, where Var is a set of variables and $\mathbb{I}_{\mathbb{R}}$ is the set of convex subsets of $\mathbb{R}$. Note that we also support discrete domain (integer and Boolean) variables. To this end it suffices to clip the interval of integers variables accordingly, such that $[-3.4, 6.0)$ becomes $[-3, 5]$, for example. The Boolean domain is represented by $\mathbb{B} = [0, 1] \subset \mathbb{Z}$. If both ρ' and ρ are interval valuations then ρ' is called a *refinement* of ρ iff $\rho'(v) \subseteq \rho(v)$ for each variable $v \in \mathit{Var}$. The left and right interval borders of an interval $\rho(x)$ for a variable x can be encoded as simple bounds. We denote the left and right interval border of the interval $\rho(x)$ by $\mathit{left}(\rho(x))$ and $\mathit{right}(\rho(x))$,

[3] For examples of the user-level syntax, consult the benchmark files on `http://hysat.informatik.uni-oldenburg.de/`.

respectively. E.g., for the interval $\rho(x) = (-4, 9]$ we have $\mathit{left}(\rho(x)) = x > -4$ and $\mathit{right}(\rho(x)) = x \leq 9$.

Let x, y be variables, ρ be an interval valuation, and $\circ$ be a binary operation. Then $\rho(x \circ y)$ denotes the *interval hull* of $\rho(x) \hat{\circ} \rho(y)$ (i.e. the smallest enclosing interval which is representable by machine arithmetic), where the operator $\hat{\circ}$ corresponds to $\circ$ but is canonically lifted to sets. This is done analogously for unary operators.

We say that an atom a is *inconsistent* under an interval valuation ρ, denoted $\rho \sharp a$, iff no values in the intervals $\rho(x)$ of the variables x in a can satisfy the atom a, i.e.

$$\begin{array}{lll} \neg\exists v \in \rho(x) & : v \sim c & \text{if } a = (x \sim c), \\ \neg\exists v \in \rho(x), \neg\exists v' \in \rho(\circ y) & : v \sim v' & \text{if } a = (x \sim \circ y), \\ \neg\exists v \in \rho(x), \neg\exists v' \in \rho(y \circ z) & : v \sim v' & \text{if } a = (x \sim y \circ z) \end{array}$$

where $\sim \in \{<, \leq, =, \geq, >\}$. Otherwise a is *consistent* under ρ. For our purpose we do not need the definition of interval satisfaction. It is sufficient to talk about atoms, which are still consistent (i.e., either *true* or *inconclusive*). For the issue of interval satisfaction the reader is referred to [FHT+07, Subsection 4.5]. Note that deciding (in)consistency of an atom under an interval valuation is straightforward. The unique reason $\mathit{reason\_inconsistent}(a, \rho)$ for the inconsistency of an atom a under an interval valuation ρ is given by the set of all interval borders of the variables x occurring in a. More formally, $\mathit{reason\_inconsistent}(a, \rho) = \{\mathit{left}(\rho(x)), \mathit{right}(\rho(x)) : x \in a\}$.

One essential ingredient of the iSAT algorithm is the use of interval constraint propagation implemented by so-called *contractors*. Given a constraint c and an interval valuation ρ, a *contractor* for c potentially computes a refinement ρ' of ρ s.t. ρ' contains all solutions of c in ρ (cf. the notion of *narrowing operator* [BMH94,Ben96]). As stated by our syntax, we can restrict ourselves to just consider contractors for *primitive constraints*, i.e. constraints containing *one* relational and at most *one* arithmetic operator, and at most *three* variables. Thus, atoms of our syntax, i.e. simple bounds like, e.g., $x \geq 2$ and arithmetic predicates like, e.g., $x < y \cdot z$, are primitive constraints.

Given the primitive constraint $z = x + y$ and the interval valuation ρ with $\rho(x) = [1, 4]$, $\rho(y) = [2, 3]$, and $\rho(z) = [0, 5]$: We can solve the primitive constraint for each of the free variables, arriving at $x = z - y$, $y = z - x$, and $z = x + y$. Each of these forms allows us to contract the interval associated with the variable on the left-hand side of the equation: Using the first solved form we subtract the interval $[2, 3]$ for y from the interval $[0, 5]$ for z, concluding that x can only be in $[-3, 3]$. Intersecting this interval with the original interval $[1, 4]$, we know that x can only be in $[1, 3]$. Proceeding in a similar way for the solved form $y = z - x$ does not change any interval, and finally, using the solved form $z = x + y$, we can conclude that z can only be in $[3, 5]$. There is extensive literature [Neu90,HJvE01] providing precise formulae for interval arithmetic for addition, subtraction, multiplication, division, and the most common transcendental functions. The floating point results are always rounded outwards, such that the result remains correct also under rounding errors.

Note that each individual contraction ρ' can be decomposed into a set of individual contractions affecting just one face of ρ' each, and each having a subset of the bounds describing the faces of ρ as a reason. E.g., in the above example, the first interval contraction derives the new *simple* bound $x \leq 3$ from the reasons $y \geq 2$ and $z \leq 5$ also being *simple* bounds, using equation $z = x + y$ (in its solved form $x = z - y$). In the sequel, we denote such an atomic derivation of ICP by $(b_1, \ldots, b_m) \overset{a}{\rightsquigarrow} (b)$ where a is a primitive constraint and $b, b_1, \ldots, b_m$ are simple bounds of the form $x \sim k$ where x is a variable, $\sim \in \{>, \geq, \leq, <\}$ (i.e. these simple bounds are inequalities), and k is a rational number. We require soundness of atomic ICP derivations in the sense that if $(b_1, \ldots, b_m) \overset{a}{\rightsquigarrow} (b)$ then $(b_1 \wedge \ldots \wedge b_m \wedge a) \Rightarrow b$

logically holds. In our example, we have $(y \geq 2, z \leq 5) \overset{z=x+y}{\rightsquigarrow} (x \leq 3)$. Note that, an atomic derivation of ICP is in particular defined for primitive constraints being simple bounds. Then, the reason for such a derivation is allowed to be empty, e.g. $() \overset{x\geq 2}{\rightsquigarrow} (x \geq 2)$ or $() \overset{x=2}{\rightsquigarrow} (x \geq 2)$.

For the sake of a logically consistent description of the rules later on, we abuse the notion of interval contraction in the sense that we allow any interval contraction b from a primitive constraint a which is inconsistent under the current interval valuation ρ. More precisely, if $\rho \sharp a$ for some atom a then $(b_1, \ldots, b_m) \overset{a}{\rightsquigarrow} (b)$ for any simple bound b, where $\{b_1, \ldots, b_m\} = \{\mathit{left}(\rho(x)), \mathit{right}(\rho(x)) : x \in a\}$. Note that our soundness requirement of atomic ICP derivations remains untouched, i.e. $(b_1 \wedge \ldots \wedge b_m \wedge a) \Rightarrow b$ holds for any simple bound b since $(b_1 \wedge \ldots \wedge b_m \wedge a)$ is unsatisfiable by $\rho \sharp a$.

For an example of the iSAT algorithm running on a formula in CNF as specified above, the reader is referred to the original paper on pages 217–219.

3 Certifying the unsatisfiability

In this section, we introduce the resolution-based calculus $\mathcal{RC}$ for the unsatisfiability of mixed Boolean and arithmetic constraint formulae represented in syntax of Section 2. Due to the undecidability of the unsatisfiability problem of such formulae, $\mathcal{RC}$ is not terminating in general. Actually, $\mathcal{RC}$ is just a special version of the iSAT algorithm, but simplified in its presentation for the purpose of proving unsatisfiable formulae. Thus, $\mathcal{RC}$ can be simulated by the iSAT algorithm. Since $\mathcal{RC}$ is able to generate a simple proof of unsatisfiable formulae, we can therefore enhance iSAT to return such a proof.

3.1 A resolution calculus for unsatisfiable formulae

The resolution calculus $\mathcal{RC}$ consists of two rules both deriving new clauses which are implied by the original formula. The first rule generates clauses by a *deduction* process while the second rule generates clauses by *resolution*.

A clause c of our syntax is a logical disjunction of atoms, i.e. $c = (a_1 \vee \ldots \vee a_n)$. We call a clause c *normalized* iff c contains no simple bound b which implies another simple bound $b' \neq b$ in c. I.e., $\neg\exists b \in c$ s.t. b is a simple bound and $\exists b' \in c$ s.t. b' is a simple bound and $b' \neq b : b \Rightarrow b'$. E.g., the clause $(x \geq 2 \vee x > 4)$ is not normalized since $x > 4 \Rightarrow x \geq 2$. For any clause c the clause $\mathit{normalize}(c)$ contains all simple bounds b in c which do not imply any other simple bound $b' \neq b$ in c, and all arithmetic predicates from c. In other words, $\mathit{normalize}(c)$ removes all simple bounds from c which imply other simple bounds. Formally, $a \in \mathit{normalize}(c)$ iff $a \in c$ and if a is a simple bound then there is no simple bound $b \in c$ with $b \neq a : a \Rightarrow b$. E.g., $\mathit{normalize}(x \geq 2 \vee x > 4) = (x \geq 2)$. It is not hard to see that the clauses c and $\mathit{normalize}(c)$ are semantically equivalent (i.e. $c \equiv \mathit{normalize}(c)$), and that $\mathit{normalize}(c)$ is a normalized clause.

We call a clause c *unit* under an interval valuation ρ iff all atoms in c but at most one are inconsistent under ρ, i.e. $\exists a \in c : \forall a' \in c$ with $a' \neq a : \rho \sharp a'$. Then, the atom a is called *remaining atom*. We remark here that –contrary to the common notion in propositional SAT solving– also *all* atoms in a unit clause can be inconsistent under ρ. This small generalization does not influence the fact that a unit clause potentially implies new interval bounds of some variables. In particular, a unit clause c in which all atoms are inconsistent under ρ logically implies any bound, since c is logically `false` under ρ. The rationale for this more general notion of unit clauses is the much easier description of checking the applicability of the rules as we will see later on. The unique reason $\mathit{reason\_unit}(c, a, \rho)$ for the fact that c is unit under ρ where $a \in c$

is a remaining atom, is given by the union of the reasons for the inconsistencies of all atoms $a' \neq a$ in c, i.e. $\mathit{reason\_unit}(c, a, \rho) = \bigcup_{a' \in c, a' \neq a} \mathit{reason\_inconsistent}(a', \rho)$.

If a clause c becomes unit under an interval valuation ρ within iSAT's proof search and the remaining atom $a \in c$ is still consistent then a has to hold in order to satisfy c under a refinement of ρ. Otherwise, c and thus the whole formula cannot be satisfied. Then, iSAT is potentially able to deduce new simple bounds b from a by ICP, i.e. $(b_1, \ldots, b_m) \overset{a}{\rightsquigarrow} (b)$ where the simple bounds $b_1, \ldots, b_m$ can be simply generated by the current interval borders of the variables in a given by ρ. If clause c becomes or is unit and all atoms in c are inconsistent under ρ, then a conflict situation occurred and we are allowed to deduce any bound as explained before. This conflict will be detected by iSAT and triggers iSAT's conflict resolution procedure which will be mentioned later on.

E.g., let $c = (x > \sin(y) \vee d \geq e)$ and $\rho(x) = [-10, -2.1]$, $\rho(y) = [-44.2, 55.3)$, $\rho(d) = [-5, 20]$, and $\rho(e) = [2, 12]$. Then, c is unit under ρ since $\rho \sharp (x > \sin(y))$ since $x > \sin(y) \geq -1$ and $x \leq -2.1$. Thus, the reason that c is unit under ρ with the remaining atom $d \geq e$ is $\mathit{reason\_unit}(c, d \geq e, \rho) = \mathit{reason\_inconsistent}(x > \sin(y), \rho) = \{x \geq -10, x \leq -2.1, y \geq -44.2, y < 55.3\}$. From $d \geq e$ and $e \geq 2$ we are able to deduce that $d \geq 2$, i.e. $(e \geq 2) \overset{d \geq e}{\rightsquigarrow} (d \geq 2)$. The deduction process in iSAT can also be explained by learning new implied clauses instead of refining interval valuations. For the fact that c is unit under ρ, we can write

$$\Big(\overbrace{(x > \sin(y) \vee d \geq e)}^{c} \wedge \overbrace{x \geq -10 \wedge x \leq -2.1 \wedge y \geq -44.2 \wedge y < 55.3}^{\mathit{reason\_unit}(c, d \geq e, \rho)}\Big) \Rightarrow \Big(d \geq e\Big)$$
$$\equiv$$
$$\Big(x > \sin(y) \vee d \geq e\Big) \Rightarrow \Big(\big(x \geq -10 \wedge x \leq -2.1 \wedge y \geq -44.2 \wedge y < 55.3\big) \Rightarrow \big(d \geq e\big)\Big).$$

For deducing $d \geq 2$ from $d \geq e$ and $e \geq 2$, we can write

$$\big(d \geq e \wedge e \geq 2\big) \Rightarrow \big(d \geq 2\big) \quad \equiv \quad \big(d \geq e\big) \Rightarrow \big(e \geq 2 \Rightarrow d \geq 2\big).$$

Then, we have

$$\begin{aligned}
&\Big(x > \sin(y) \vee d \geq e\Big) \\
\Rightarrow\ &\Big(\big(x \geq -10 \wedge x \leq -2.1 \wedge y \geq -44.2 \wedge y < 55.3\big) \Rightarrow \big(d \geq e\big)\Big) \\
\Rightarrow\ &\Big(\big(x \geq -10 \wedge x \leq -2.1 \wedge y \geq -44.2 \wedge y < 55.3\big) \Rightarrow \big(e \geq 2 \Rightarrow d \geq 2\big)\Big) \\
\equiv\ &\Big(\big(x < -10 \vee x > -2.1 \vee y < -44.2 \vee y \geq 55.3\big) \vee \big(e < 2 \vee d \geq 2\big)\Big).
\end{aligned}$$

I.e., c implies the clause $l = (x < -10 \vee x > -2.1 \vee y < -44.2 \vee y \geq 55.3 \vee e < 2 \vee d \geq 2)$ just containing simple bounds. Such clauses l are implicity given during iSAT's proof search and can be simply generated.

The first rule of $\mathcal{RC}$ generates such clauses containing just simple bounds and are implied by an already existing clause.

$$\frac{\begin{array}{c} cl = (a_1 \vee \ldots \vee a_n), \\ \exists \rho \in \mathcal{V} : \exists i \in \{1, \ldots, n\} : \forall j \neq i : \rho \sharp a_j, \\ \mathit{reason\_unit}(cl, a_i, \rho) = \{b_1, \ldots, b_m\}, (b'_1, \ldots, b'_k) \overset{a_i}{\rightsquigarrow} (b'), \\ \{b'_1, \ldots, b'_k\} \subseteq \{\mathit{left}(\rho(x)), \mathit{right}(\rho(x)) : x \in a_i\} \end{array}}{\mathit{normalize}(\neg b_1 \vee \ldots \vee \neg b_m \vee \neg b'_1 \vee \ldots \vee \neg b'_k \vee b')} \tag{1}$$

where $\mathcal{V}$ is the set of all interval valuations. The following lemma holds, for which the proof is sketched by the example above ($\mathit{normalize}(\cdot)$ yields a semantically equivalent clause).

Lemma 1. *For rule 1, it holds that* $(a_1 \vee \ldots \vee a_n) \Rightarrow \mathit{normalize}(\neg b_1 \vee \ldots \vee \neg b_m \vee \neg b'_1 \vee \ldots \vee \neg b'_k \vee b')$.

The iSAT algorithm employs a conflict resolution procedure based on a so-called *implication graph* which keeps track of all decisions and deductions done so far [FHT+07,THFA08]. Whenever a conflict occurs during the proof search, i.e. deducing a simple bound b_1 contradicting another simple bound b_2 like, e.g., $b_1 = (x < 5)$ and $b_2 = (x \geq 13.2)$, the conflicting situation is resolved by undoing all involved decisions and deductions. Moreover, iSAT generates a *conflict clause* by tracing back the reasons for the conflict in the implication graph. Such an implied conflict clause excludes the conflicting interval valuation and thus prevents the solver from revisiting the same conflict again.

There are several techniques for the generation of such conflict clauses from an implication graph (cf. [THFA08] for a very detailed analysis). The main idea, however, is to consecutively apply some *resolution* steps until a suitable conflict clause is computed. One resolution step in iSAT's conflict resolution procedure can be specified by rule 2. E.g., if the clauses $(x < 5 \vee y \geq 9 \vee g \leq -3.9)$ and $(x \geq 13.2 \vee d > -1.1)$ are unit under some interval valuation ρ deducing the simple bounds $x < 5$ and $x \geq 13.2$, resp., and thus implying a conflict, we can learn the (conflict) clause $(y \geq 9 \vee g \leq -3.9 \vee d > -1.1)$ in order to avoid the conflicting interval valuation ρ in the future search.

$$\text{(2)} \qquad \frac{\begin{array}{c}(a \vee a_1 \vee \ldots \vee a_n), (b \vee b_1 \vee \ldots \vee b_m), \\ a, a_1, \ldots, a_n, b, b_1, \ldots, b_m \text{ are simple bounds}, \\ a = (x \sim k), b = (x \sim' k'), x \in \mathit{Var}, \{v \sim k\} \cap \{v \sim' k'\} = \emptyset\end{array}}{\mathit{normalize}(a_1 \vee \ldots \vee a_n \vee b_1 \vee \ldots \vee b_m)}$$

where *Var* is a set of variables.

Lemma 2. *For rule 2, it holds that* $\big((a \vee a_1 \vee \ldots \vee a_n) \wedge (b \vee b_1 \vee \ldots \vee b_m)\big) \Rightarrow \mathit{normalize}(a_1 \vee \ldots \vee a_n \vee b_1 \vee \ldots \vee b_m)$.

Proof. The atoms a and b cannot hold under the same valuation. Thus, each valuation satisfying $\big((a \vee a_1 \vee \ldots \vee a_n) \wedge (b \vee b_1 \vee \ldots \vee b_m)\big)$ also satisfies at least one atom in $(a_1 \vee \ldots \vee a_n \vee b_1 \vee \ldots \vee b_m) = c$. The clause $\mathit{normalize}(c)$ is semantically equivalent to c. □

For each clause c, we define a set of clauses $\mathit{antecedents}(c)$, called *antecedent clauses of* c, which is empty if not otherwise stated. For a clause c derived by rule 1, we set $\mathit{antecedents}(c) = \{(a_1 \vee \ldots \vee a_n)\}$ where $c = \mathit{normalize}(\neg b_1 \vee \ldots \vee \neg b_m \vee \neg b'_1 \vee \ldots \vee \neg b'_k \vee b')$, and for a clause c' derived by rule 2, we set $\mathit{antecedents}(c') = \{(a \vee a_1 \vee \ldots \vee a_n), (b \vee b_1 \vee \ldots \vee b_m)\}$ where $c' = \mathit{normalize}(a_1 \vee \ldots \vee a_n \vee b_1 \vee \ldots \vee b_m)$.

We say that a clause c is $\mathcal{RC}$*-derivable from one clause* c' or *from two clauses* c_1 *and* c_2 iff c contains simple bounds only and rule 1 applied on c' or rule 2 applied on c_1, c_2, resp., yields clause c.

Checking applicability of rule 2 is easy. For checking the applicability of rule 1, we have to show that there is some interval valuation ρ s.t. clause c can be derived from c' under ρ by rule 1. The clause c contains simple bounds only. Thus, $\neg c$ is a conjunction of simple bounds describing an interval valuation ρ'. Note that ρ' is the maximum interval valuation satisfying $\neg c$. I.e., each simple bound in $\neg c$ is actually satisfied under ρ' and under each refinement of ρ'. Assume that rule 1 is applicable with some ρ. Then, there exists some simple bound $b' \in c$ for which $(b'_1, \ldots, b'_k) \overset{a_i}{\rightsquigarrow} (b')$ in the premise of rule 1 holds. Then by construction of c, $\neg c''$ describes the interval valuation ρ where $c'' = c - \{b'\}$. By $\neg c = \neg c'' \wedge \neg b'$, the interval valuation ρ' is a refinement of ρ. Assume that the atomic ICP derivation $(b'_1, \ldots, b'_k) \overset{a_i}{\rightsquigarrow} (b')$ was applied, then $\rho' \sharp a_i$ since $\neg b' \in \neg c$. Thus, all atoms in c' are inconsistent under ρ', in particular all but one, i.e. $\forall a \in c' : \rho' \sharp a$. Moreover,

$\forall a \in c', \forall b \in c : (\neg c) \overset{a}{\rightsquigarrow} (b)$, since atom a is inconsistent under ρ' (= maximum interval model of $\neg c$) and thus we can deduce anything, in particular b. Then, rule 1 is also applicable with ρ'. The reverse direction trivially holds. I.e., if rule 1 is applicable with ρ' then there is some ρ s.t. rule 1 is applicable with ρ, namely $\rho := \rho'$. Summing up, rule 1 applied on clause c' yields clause c iff $\forall a \in c' : \rho' \sharp a$ holds where ρ' is the interval valuation described by $\neg c$.

Definition 1 ($\mathcal{RC}$ calculus). *Given a mixed Boolean and arithmetic formula φ in syntax of Section 2. The* $\mathcal{RC}$ calculus *is defined by the consecutive application of rule 1 or 2 with one or two clauses from $\varphi \cup \psi$, resp., where ψ is the set of all already derived clauses. $\mathcal{RC}$ stops and returns ψ iff $\emptyset \in \psi$ or no rule is applicable.*

Immediately by Definition 1, Lemma 1, and Lemma 2 the next result follows.

Corollary 1. *If $\mathcal{RC}$ applied on φ returns ψ s.t. $\emptyset \in \psi$ (i.e. $\mathcal{RC}$ generates the empty clause) then φ is unsatisfiable.*

3.2 Proofs of unsatisfiability

We are interested in a *proof* or *certificate* $\mathcal{P}$ of unsatisfiability of a given mixed Boolean and arithmetic formula φ in CNF s.t. $\mathcal{P}$ enables us to prove unsatisfiability of φ by simple instructions. I.e., from $\mathcal{P}$ we can extract a simple, efficient, and deterministic calculation which shows unsatisfiability of φ without using, e.g., exhaustive search methods.

Before we introduce the notion of a proof of unsatisfiability, we need another definition. For any clause r, we define the set $Gr(r)$ of clauses, called *directed graph with root r*, as the smallest set satisfying the following two conditions.

1. $r \in Gr(r)$.
2. Let $c \in Gr(r)$. If $c' \in \mathit{antecedents}(c)$ then $c' \in Gr(r)$ and (c, c') is a directed edge.

Definition 2 (Proof of unsatisfiability). *Given a formula φ in syntax of Section 2. We call a finite set $\mathcal{P}$ of clauses a* proof of unsatisfiability of φ *if the following conditions are satisfied.*

1. *$\mathcal{P}$ contains the empty clause, i.e. $\emptyset \in \mathcal{P}$.*
2. *The directed graph with root $\emptyset \in \mathcal{P}$ is acyclic and only contains clauses from $\mathcal{P} \cup \varphi$, i.e. $Gr(\emptyset)$ is acyclic and $Gr(\emptyset) \subseteq \mathcal{P} \cup \varphi$.*
3. *For each clause $c \in Gr(\emptyset)$ the number of its antecedent clauses is at most 2, i.e. $|\mathit{antecedents}(c)| \leq 2$.*
4. *Each clause $c \in Gr(\emptyset)$ without antecedent clauses is a clause in φ, i.e. if $\mathit{antecedents}(c) = \emptyset$ then $c \in \varphi$.*
5. *Each clause $c \in Gr(\emptyset)$ with $1 \leq |\mathit{antecedents}(c)| \leq 2$, i.e. with one or two antecedent clauses, is $\mathcal{RC}$-derivable from their antecedent clauses.*

Lemma 3. *If $\mathcal{P}$ is a proof of unsatisfiability of φ then φ is unsatisfiable.*

Proof. By Definition 2, Lemma 1, and Lemma 2 all clauses in $Gr(\emptyset)$ are implied by (some clauses in) φ, in particular the empty clause. □

Proposition 1. *If $\mathcal{RC}$ applied on φ returns ψ with $\emptyset \in \psi$ then ψ is a proof of unsatisfiability of φ.*

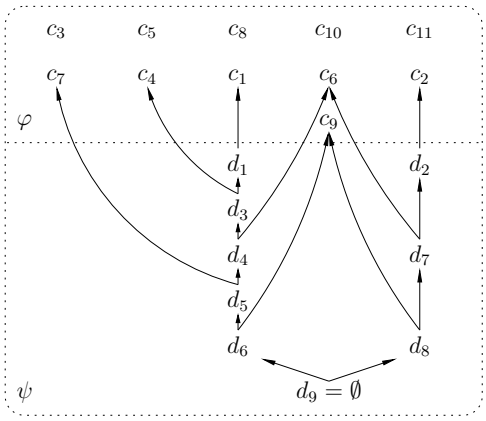

Fig. 1. Clauses of φ and ψ. Directed edges connect clauses with their antecedents.

Proof. All conditions of Definition 2 are satisfied by ψ. By condition, $\emptyset \in \psi$. By Definition 1, $Gr(\emptyset)$ is acyclic since whenever $\mathcal{RC}$ adds a clause to ψ its antecedent clauses already exist in ψ, $Gr(\emptyset) \subseteq \psi \cup \varphi$, and each clause in $Gr(\emptyset) \subseteq \psi \cup \varphi$ has at most two antecedents (due to rules 1 or 2). Moreover, each clause in $Gr(\emptyset)$ without antecedents lies in φ, since each clause in ψ has at least one antecedent and each clause in φ has no antecedents. By construction of ψ, all clauses in ψ are $\mathcal{RC}$-derivable from their antecedent clauses. □

Definition 2 actually implements a simple tool for validating unsatisfiability results of the $\mathcal{RC}$ calculus. As mentioned in Subsection 3.1, the $\mathcal{RC}$ calculus itself can be simulated by the iSAT solver and thus iSAT is capable of generating a proof of unsatisfiability for mixed Boolean and non-linear arithmetic constraint formulae. More precisely, for each atomic ICP derivation, iSAT outputs the derived clause c linked to its antecedent clause as specified by rule 1. Whenever iSAT performs the conflict resolution procedure all intermediate clauses are output together with their antecedent clauses (rule 2). We remark here that it is sufficient to output only these clauses derived by rule 1 which are actually involved in the generation of a conflict clause. Since all ICP derivations are stored in the implication graph during the search, we output a clause c derived by rule 1 whenever c is needed within the conflict analysis.

3.3 Example of the $\mathcal{RC}$ calculus

Given the formula $\varphi = c_1, \ldots, c_{11}$ where

$$c_1 = (x = \sin(y) \vee a > b), \quad c_2 = (a < b \cdot x), \quad c_3 = (x \geq y^2 \vee y < 15.3),$$

and $c_4 = (a \geq -100), c_5 = (a \leq 0), c_6 = (b > 3.71), c_7 = (b \leq 100), c_8 = (x \geq -100), c_9 = (x \leq -2.1), c_{10} = (y \geq -100), c_{11} = (y \leq 100)$, where a, b, x, and y are real-valued variables. Note that clauses $c_4, \ldots, c_{11}$ encode the ranges of the variables. By rule 1, we can derive from c_1 the clause

$$d_1 = (a < -100 \vee a > 0 \vee b \leq 3.71 \vee b > 100 \vee x \geq -1)$$

since for all ρ s.t. $\rho(a) = [-100, 0], \rho(b) = (3.71, 100]$, $\rho \sharp (a > b)$ holds, then, *reason_unit*$(c_1, x = \sin(y), \rho) = \{a \geq -100, a \leq 0, b > 3.71, b \leq 100\}$, and, trivially, $() \overset{x=\sin(y)}{\rightsquigarrow} (x \geq -1)$. The clause

$$d_2 = (b \leq 0 \vee x \geq 0 \vee a < 0)$$

depth	#vars	#clauses	#conflicts	no proof	proof	ratio	proof size	size ratio	file size	proof check
bouncing ball										
15	969	886	5,235	15.7s	17.9s	1.1	51,656	58.3	28.0M	5.4s
16	1,032	944	125	0.1s	0.6s	6.0	7,452	7.9	7.3M	1.6s
17	1,095	1,002	225	0.3s	1.2s	4.0	14,328	14.3	14.0M	2.7s
18	1,158	1,060	809	0.8s	1.7s	2.1	15,352	14.5	13.0M	2.7s
19	1,221	1,118	586	0.7s	1.9s	2.7	18,754	16.7	19.0M	3.8s
20	1,284	1,176	7,182	38.6s	45.1s	1.2	104,280	88.7	90.0M	21.9s
gasburner										
15	738	882	4,378	7.3s	7.5s	1.0	11,272	12.8	2.1M	0.2s
16	786	940	5,114	9.7s	9.9s	1.0	13,638	14.5	2.6M	0.3s
17	834	998	5,779	12.5s	12.9s	1.0	18,024	18.1	4.0M	0.5s
18	882	1,056	7,769	22.8s	22.8s	1.0	22,610	21.4	4.8M	0.6s
19	930	1,114	8,026	24.8s	24.8s	1.0	21,778	19.5	4.3M	0.5s
20	978	1,172	7,772	23.9s	24.0s	1.0	20,844	17.8	4.2M	0.5s
minigolf										
5	1,231	830	334	0.3s	0.7s	2.3	6,404	7.7	5.5M	1.2s
6	1,473	995	794	1.0s	2.0s	2.0	15,126	15.2	14.0M	3.3s
7	1,715	1,160	1,577	5.1s	10.2s	2.0	48,230	41.6	70.0M	24.1s
8	1,957	1,325	3,840	34.5s	55.4s	1.6	140,363	105.9	280.0M	125.5s
9	2,199	1,490	9,587	341.0s	466.3s	1.4	501,653	336.7	1651.0M	1152.0s
renault clio										
6	2,117	2,111	4,138	8.9s	9.4s	1.1	18,720	8.9	9.3M	2.3s
7	2,464	2,460	6,032	19.2s	20.0s	1.0	27,768	11.3	15.0M	4.0s
8	2,811	2,809	7,620	30.0s	30.6s	1.0	31,982	11.4	14.0M	2.8s
9	3,158	3,158	9,801	50.0s	52.2s	1.0	46,466	14.7	36.0M	16.2s
10	3,505	3,507	13,535	98.0s	101.3s	1.0	66,998	19.1	65.0M	35.8s

Table 1. Experimental results

is $\mathcal{R}C$-derivable from c_2 since $(b > 0, x < 0) \overset{a<b\cdot x}{\rightsquigarrow} (a < 0)$. The antecedent clauses are thus given by $\mathit{antecedents}(d_1) = \{c_1\}$ and $\mathit{antecedents}(d_2) = \{c_2\}$. Starting with d_1, we can perform a resolution chain, i.e. a consecutive application of rule 2:

$$\begin{aligned}
d_3 &= (\ a > 0 \vee b \leq 3.71 \vee b > 100 \vee x \geq -1\ ) && \text{from } d_1, c_4,\\
d_4 &= (\ a > 0 \qquad\quad \vee b > 100 \vee x \geq -1\ ) && \text{from } d_3, c_6,\\
d_5 &= (\ a > 0 \qquad\qquad\quad \vee x \geq -1\ ) && \text{from } d_4, c_7,\\
d_6 &= (\ a > 0 \qquad\qquad\qquad\quad\ \ ) && \text{from } d_5, c_9.
\end{aligned}$$

Furthermore, the next applications of rule 2 finally yields the empty clause:

$$\begin{aligned}
d_7 &= (\ x \geq 0 \vee a < 0\ ) && \text{from } d_2, c_6,\\
d_8 &= (\qquad a < 0\ ) && \text{from } d_7, c_9,\\
d_9 &= (\qquad\qquad\ ) && \text{from } d_8, d_6.
\end{aligned}$$

By Corollary 1, the formula φ is unsatisfiable, and by Proposition 1 the set $\psi = \{d_1, \ldots, d_9\}$ is a proof of unsatisfiability of φ. The original clauses in φ and all derived clauses in ψ as well as the directed (acyclic) graph $Gr(\emptyset)$ are shown in Fig. 1. Note that only the clauses $c_1, c_2, c_4, c_6, c_7, c_9 \in \varphi$ are leaves of $Gr(\emptyset)$. I.e., even the clauses $c_1, c_2, c_4, c_6, c_7, c_9$ together are unsatisfiable (aka. *unsatisfiable core* of a formula) for which ψ is also a proof. To potentially achieve smaller unsatisfiable cores, this method can be iterated on the correspondingly previous unsatisfiable core until a fixed point is reached as, e.g., in [ZM03b].

4 Experimental results

This section compiles empirical results using benchmarks from the hybrid systems domain. We considered bounded model checking (BMC) problems, i.e., proving a property of a hybrid discrete-continuous transition system for a fixed unwinding depth k. Besides the well-known *bouncing ball* example, these BMC problems are

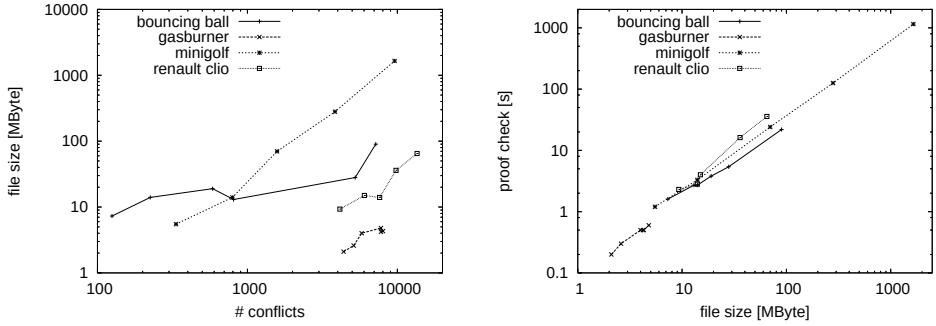

Fig. 2. Number of encountered conflicts vs. size of the generated proof file (left) and file size vs. runtime of the proof checker (right)

a model of a *leaking gas burner* [ACH+95], a model of the discrete-continuous behavior of a golf ball on a *minigolf course*, and a model of a *Renault Clio* equipped with a cruise controller [TB01]. All benchmarks were performed on a 2.13 GHz Intel Core 2 Duo machine with 1 GByte physical memory running Linux. For each benchmark, the results for some unsatisfiable unwinding instances are listed in Table 1. The columns *depth*, *#vars*, *#clauses*, and *#conflicts* state the corresponding BMC unwinding depth of the benchmark, the corresponding number of variables, the number of clauses, and the number of conflicts occurred during the search, respectively. The runtimes of iSAT in its usual mode (i.e. without generating a proof of unsatisfiability) and in the proof-generating mode are given in columns *no proof* and *proof*, resp., while *ratio* compiles the factor of extra work for the proof generation, i.e. $ratio = (proof)/(no\ proof)$. Column *proof size* gives the number of clauses in the proof of unsatisfiability, and *size ratio* computes the size of the proof in relation to the size of the original formula, i.e. $size\ ratio = (proof\ size)/(\#clauses)$.

The unsatisfiability proofs are written into a text file. More precisely, we output the declaration of the variables and their initial intervals, the problem and conflict clauses, and all clauses with their antecedents which are derived by rules 1 and 2, and which are needed to explain the conflict clauses. One worth mentioning technicality is that we dump floating point numbers in their binary representation and *not* as decimals. The reason is that re-converting decimals into floats does not yield the original floats and, thus, the proof-checks frequently failed. The corresponding sizes in MByte of these text files are shown in column *file size*. Our simple proof checker reads the potential unsatisfiability proof from a text file, and then tests whether the given proof actually satisfies each condition from Definition 2. The runtimes of the proof checker are listed in column *proof check*.

Interpreting the empirical results, we first conclude that the overhead of generating a proof of unsatisfiability is very small. However, as expected, the size of the proofs is quickly increasing the harder the problems, and strongly depends on the number of encountered conflicts as, e.g., indicated by the minigolf benchmark, and as depicted on the left of Fig. 2. The text file size of the minigolf-9 instance already exceeded 1.6 GByte, although the number of problem clauses is relatively small. For greater unwinding depths, the file sizes reached scores of GBytes. The runtimes of the external proof checker are in a very clear proportion to the sizes of the text files, cf. the right-hand side of Fig 2.

5 Conclusion and future work

In this paper, we presented a technique to verify unsatisfiability results for mixed Boolean and non-linear arithmetic constraints over the reals and integers. Based on the introduced $\mathcal{RC}$ calculus, we extended the iSAT tool to produce proofs of unsatisfiability, and implemented a simple proof checker to verify these proofs. Furthermore, we proved the concept of this approach on benchmarks from the hybrid system's domain. Therefore, this work contributes to more trustworthiness, e.g., in the analysis of safety-critical systems.

The main focus for future work is on the generation of Craig Interpolants for the hybrid Boolean and non-linear arithmetic framework by means of unsatisfiability proofs. With the aid of interpolation, we will extend the bounded-model-checking layer of the iSAT tool to the *unbounded* case. Such an unbounded model checking procedure will be potentially able to prove the *safety* of a hybrid discrete-continuous system in the sense that a specified undesired property will *never* be satisfied by the system, and not only within some bounded number of system steps.

References

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, *The algorithmic analysis of hybrid systems*, Theoretical Computer Science **138** (1995), no. 1, 3–34.

[BCZ99] A. Biere, A. Cimatti, and Y. Zhu, *Symbolic model checking without BDDs*, TACAS'99, LNCS, vol. 1579, Springer Verlag, 1999.

[Ben96] F. Benhamou, *Heterogeneous Constraint Solving*, Proc. of the Fifth International Conf. on Algebraic and Logic Programming, LNCS, vol. 1139, Springer, 1996.

[BG06] F. Benhamou and L. Granvilliers, *Continuous and interval constraints*, Handbook of Constraint Programming (F. Rossi, P. van Beek, and T. Walsh, eds.), Foundations of Artificial Intelligence, Elsevier, Amsterdam, 2006, pp. 571–603.

[BHJM05] D. Beyer, T. A. Henzinger, R. Jhala, and R. Majumdar, *The software model checker* Blast*: Applications to software engineering*, International Journal on Software Tools for Technology Transfer (STTT) **9** (2007. Invited to special issue of selected papers from FASE 2004/05), no. 5-6, 505–525.

[BMH94] F. Benhamou, D. McAllester, and P. Van Hentenryck, *CLP(Intervals) revisited*, Int. Symp. on Logic Progr. (Ithaca, NY, USA), MIT Press, 1994, pp. 124–138.

[Cra57] W. Craig, *Linear reasoning: A new form of the Herbrand-Gentzen theorem*, Journal of Symbolic Logic **22** (1957), no. 3, 250–268.

[DLL62] M. Davis, G. Logemann, and D. Loveland, *A Machine Program for Theorem Proving*, CACM **5** (1962), 394–397.

[DP60] M. Davis and H. Putnam, *A Computing Procedure for Quantification Theory*, Journal of the ACM **7** (1960), no. 3, 201–215.

[FHT+07] M. Fränzle, C. Herde, T. Teige, S. Ratschan, and T. Schubert, *Efficient Solving of Large Non-linear Arithmetic Constraint Systems with Complex Boolean Structure*, JSAT Special Issue on SAT/CP Integration **1** (2007), 209–236.

[GKvV95] J. F. Groote, J. W. C. Koorn, and S. F. M. van Vlijmen, *The Safety Guaranteeing System at Station Hoorn-Kersenboogerd*, Conference on Computer Assurance, National Institute of Standards and Technology, 1995, pp. 57–68.

[GN03] E. Goldberg and Y. Novikov, *Verification of Proofs of Unsatisfiability for CNF Formulas*, Proc. of DATE 2003, IEEE, 2003, pp. 10886–10891.

[HJvE01] T. J. Hickey, Q. Ju, and M. H. van Emden, *Interval arithmetic: from principles to implementation*, Journal of the ACM **48** (2001), no. 5, 1038–1068.

[LS04] I. Lynce and J. P. Marques Silva, *On Computing Minimum Unsatisfiable Cores*, Proc. of SAT 2004, 2004.

[McM03] K. L. McMillan, *Interpolation and SAT-based model checking*, CAV, 2003, pp. 1–13.

[McM05] ———, *An interpolating theorem prover*, Theor. Comput. Sci. **345** (2005), no. 1, 101–121.

[Mos08] M. Moskal, *Rocket-fast proof checking for SMT solvers*, Proc. of TACAS 2008, 2008.

[Neu90] A. Neumaier, *Interval methods for systems of equations*, Cambridge Univ. Press, 1990.

[NOT06] R. Nieuwenhuis, A. Oliveras, and C. Tinelli, *Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T)*, Journal of the ACM **53** (2006), no. 6, 937–977.

[RT06] S. Ranise and C. Tinelli, *Satisfiability modulo theories*, IEEE Intelligent Systems **21** (2006), no. 6.

[TB01] F. D. Torrisi and A. Bemporad, *Discrete-time hybrid modeling and verification*, Proc. 41th IEEE Conf. on Decision and Control, 2001, pp. 2899–2904.

[THFA08] T. Teige, C. Herde, M. Fränzle, and E. Ábrahám, *Conflict Analysis and Restarts in a mixed Boolean and Non-linear Arithmetic Constraint Solver*, Reports of SFB/TR 14 AVACS 34, SFB/TR 14 AVACS, January 2008, ISSN: 1860-9821, http://www.avacs.org.

[ZM03a] L. Zhang and S. Malik, *Extracting Small Unsatisfiable Cores from Unsatisfiable Boolean Formulas*, Proc. of SAT 2003, 2003.

[ZM03b] ———, *Validating SAT Solvers Using an Independent Resolution-Based Checker: Practical Implementations and Other Applications*, Proc. of DATE 2003, IEEE, 2003, pp. 10880–10885.