Journal on Satisfiability, Boolean Modeling and Computation (2007)

Efficient Solving of Large Non-linear Arithmetic Constraint Systems with Complex Boolean Structure[*]

Martin Fränzle fraenzle@informatik.uni-oldenburg.de
Christian Herde herde@informatik.uni-oldenburg.de
Tino Teige teige@informatik.uni-oldenburg.de
Department of Computing Science,
Carl von Ossietzky Universität Oldenburg, Germany

Stefan Ratschan stefan.ratschan@cs.cas.cz
Institute of Computer Science,
Academy of Sciences of the Czech Republic, Prague, Czech Republic

Tobias Schubert schubert@informatik.uni-freiburg.de
Faculty of Applied Sciences,
Albert-Ludwigs-Universität Freiburg, Germany

Abstract

In order to facilitate automated reasoning about large Boolean combinations of non-linear arithmetic constraints involving transcendental functions, we provide a tight integration of recent SAT solving techniques with interval-based arithmetic constraint solving. Our approach deviates substantially from lazy theorem proving approaches in that it directly controls arithmetic constraint propagation from the SAT solver rather than delegating arithmetic decisions to a subordinate solver. Through this tight integration, all the algorithmic enhancements that were instrumental to the enormous performance gains recently achieved in propositional SAT solving carry over smoothly to the rich domain of non-linear arithmetic constraints. As a consequence, our approach is able to handle large constraint systems with extremely complex Boolean structure, involving Boolean combinations of multiple thousand arithmetic constraints over some thousands of variables.

Keywords: *interval-based arithmetic constraint solving; SAT modulo theories*

1. Introduction

Within many application domains, among them the analysis of programs involving arithmetic operations and the analysis of hybrid discrete-continuous systems, one faces the problem of solving large Boolean combinations of non-linear arithmetic constraints over the reals, where solving means to find a satisfying valuation or to prove nonexistence thereof. This gives rise to a plethora of problems, in particular (a) how to efficiently and sufficiently completely solve conjunctive combinations of constraints in the undecidable domain of non-

[*] 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).

linear constraints involving transcendental functions and (b) how to efficiently maneuver the large search spaces arising from the rich Boolean structure of the overall formula.

While promising solutions for these two individual sub-problems exist, it seems that their combination has hardly been attacked. Arithmetic constraint solving based on interval constraint propagation (e.g., [Dav87, BG06]), on the one hand, has proven to be an efficient means for solving robust combinations of otherwise undecidable arithmetic constraints [Rat06]. Here, robustness means that the constraints maintain their truth value under small perturbations of the constants in the constraints. Modern SAT solvers, on the other hand, can efficiently find satisfying valuations of very large propositional formulae (e.g., [MMZ$^+$01, GN02]), as well as —using the SAT modulo theory (SMT) paradigm— of complex propositional combinations of atoms from various decidable theories (e.g., [Gan02, dMOR$^+$04]).

Within this paper, we describe a tight integration of SAT-based proof search with interval-based arithmetic constraint propagation, thus providing an algorithm that reasons over the undecidable arithmetic domain of Boolean combinations of non-linear constraints involving transcendental functions. Within our approach, a DPLL-based propositional satisfiability solver traverses the proof tree originating from the Boolean structure of the constraint formula, as is characteristic for SMT. Yet, in contrast to the SMT techniques of lazy theorem proving and DPLL(T), we do not pass a corresponding conjunctive constraint system over the respective theory T to a subordinate decision procedure serving as an oracle for consistency of the constraint set (as in lazy theorem proving) and providing forward inferences wrt. implied truth values of other T-atoms occurring in the input formula (the additional DPLL(T) mechanism). Instead, we exploit the algorithmic similarities between DPLL-based propositional SAT solving and constraint solving based on constraint propagation for a much tighter integration, where the DPLL solver directly manipulates theory atoms instead of a propositional abstraction of the input formula. It has full introspection into and control over constraint propagation within the theory T, and it directly integrates any new theory atoms generated by the constraint propagation into the search space of the DPLL solver. This tight integration has a number of advantages. First, by sharing the common core of the search algorithms between the propositional and the theory-related, interval-constraint-propagation-based part of the solver, we are able to transfer algorithmic enhancements from one domain to the other: in particular, we thus equip interval-based constraint solving with all the algorithmic enhancements that were instrumental to the enormous performance gains recently achieved in propositional SAT solving, like watched-literal schemes or conflict-driven learning based on implication-graph analysis. Second, the introspection into the constraint propagation process allows fine-granular control over the necessarily incomplete arithmetic deduction process, thus enabling a stringent extension of SMT to an undecidable theory. Finally, due to the availability of learning, we are able to implement an almost lossless restart mechanism within an interval-based arithmetic constraint propagation framework.

Related work. From the extensive literature on SMT techniques, the approach coming closest to ours is the *splitting-on-demand technique in SMT* of Barett, Nieuwenhuis, Oliveras, and Tinelli [BNOT06]. There, the set of theory atoms manipulated by the DPLL solver is made dynamic by a special split rule extending the formula with a tautologous clause introducing new theory atoms. In contrast to this, our tighter integration does not need such helper mechanisms modifying the formula, allows new theory atoms to be generated by both splits and constraint propagations in the theory, and due to its direct manipulation of theory atoms generates atoms on-the-fly and locally to the different branches of the proof-search tree. We conjecture that the more direct integration and the locality help the algorithm to perform stably even under enormous numbers of new theory atoms thus being generated. A further crucial distinction to DPLL(T) is that our algorithm distinguishes between a large (and undecidable) theory that theory propagation acts on (non-linear arithmetic including transcendental functions) and a small kernel thereof used in consistency checks (real-valued inequation systems). In DPLL(T) approaches, the roles are generally reversed: consistency check has to cover the full theory T, while theory propagation (fwd. inference) may be more confined, covering a subset of T only, up to being completely missing.

Jussien's and Lhomme's *dynamic domain splitting* technique for numeric constraint satisfaction problems (numeric CSPs) [JL98] is related to our approach in that both implement conflict-driven learning and non-chronological backtracking within arithmetic constraint solving based on domain splitting and arithmetic constraint propagation. Their approach extends dynamic backtracking (cf. [Gin93]) and provides filtering algorithms for domain reductions (i.e., constraint propagation) enhanced by nogood learning and non-chronological backtracking. Nevertheless, the algorithm described in [JL98] is not general enough for our problem domain, as it focuses on conjunctive constraint systems. Our algorithm relaxes that limitation and handles non-linear arithmetic constraint systems with an arbitrary, complex Boolean structure. Another technical difference to [JL98] is the procedure applied for learning conflicts and the shape of conflict clauses thus obtained. The explanations of conflicts in [JL98] are confined to be sets of *splitting constraints*, i.e. of choice points decided in branch steps of the branch-and-reduce algorithm. Our algorithm is able to generate more compact and more general conflict clauses entailing both splitting constraints and arbitrary deduced constraints. Like in modern propositional SAT solvers, this is achieved by maintaining and analyzing an implication graph (cf. Sect. 4.3) storing the immediate reasons for each deduction. We are thus able to generalize all techniques for determining conflict clauses which have proven beneficial within propositional SAT, e.g. the 1UIP technique [ZMMM01].

Sharing our goal of checking satisfiability of large and complex-structured Boolean combinations of non-linear arithmetic constraints, Bauer, Pister, and Tautschnig have recently presented the ABsolver tool [BPT07]. ABsolver is an SMT solver addressing a blend of Boolean and polynomial arithmetic constraint problems. It is an extensible and modular implementation of the SMT scheme which permits integration of various subordinate solvers for the Boolean, linear, and non-linear parts of the input formula. ABsolver itself coordinates the overall solving process and delegates the currently active constraint sets to the corresponding subordinate solvers. The currently reported implementation [BPT07] uses the numerical optimization tool IPOPT (https://projects.coin-or.org/Ipopt) for solving the non-linear constraints. Consequently, it may produce incorrect results due to the local nature of the solver, and due to rounding errors. Nonetheless, even though in our method we implement strictly correct solving of non-linear constraints, benchmarks reported in Sect. 5.3 show that our tighter integration consistently outperforms ABsolver, usually by orders of magnitude when formulae with non-trivial Boolean structure are involved. Furthermore, our solver uses interval constraint propagation to address a larger class of formulae than polynomial constraints, admitting arbitrary smooth functions in the constraints, including transcendental ones.

Compared to *interval constraint solving* (ICP, for a survey cf. [BG06]), our approach is complementary: the interval constraint solving community is primarily concerned with solving —often in the sense of "paving" the solution set— intricate conjunctive non-linear constraint systems, and thus concentrates on powerful constraint propagation operators. Our focus is on satisfiability tests for extremely large formulae featuring a complex Boolean structure, which we make feasible by mechanisms for tracking and exploiting the dependencies between sub-formulae within an SMT framework. Thus, our approach could easily be enhanced by importing more powerful constraint propagation operators, while our mechanisms for maneuvering through large Boolean combinations of non-linear constraint systems are a contribution to interval constraint solving.

Structure of the paper. In Section 2, we expose the syntax and semantics of the arithmetic satisfiability problems our algorithm addresses. Section 3 provides a brief introduction to related technologies that our development builds on. Thereafter, we provide a detailed explanation of our new algorithm in Sect. 4. After presenting some benchmark results within Sect. 5, we conclude with an overview of ongoing work and planned extensions.

2. Logics

Aiming at automated analysis of programs operating over the reals, our constraint solver addresses satisfiability of non-linear arithmetic constraints over real-valued variables plus Boolean variables for encoding the control flow. The user thus may input constraint formulae built from quantifier-free constraints over the reals and from propositional variables using arbitrary Boolean connectives. The atomic real-valued constraints are relations between potentially non-linear terms involving transcendental functions, like $\sin(x+\omega t)+ye^{-t} \leq z+5$. By the front-end of our constraint solver, these constraint formulae are rewritten to equi-satisfiable quantifier-free formulae in conjunctive normal form, with atomic propositions ranging over propositional variables 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, thereby also eliminating common sub-expressions and common sub-formulae through re-use of the auxiliary variables, thus reducing the search space of the SAT solver and enhancing the reasoning power of the interval contractors used in arithmetic reasoning [BG06]. Thus, the *internal* syntax[1.] of constraint formulae is as follows:

$$\begin{array}{rcl}
\mathit{formula} & ::= & \{\mathit{clause} \wedge\}^* \mathit{clause} \\
\mathit{clause} & ::= & (\{\mathit{atom} \vee\}^* \mathit{atom}) \\
\mathit{atom} & ::= & \mathit{bound} \mid \mathit{equation} \\
\mathit{bound} & ::= & \mathit{variable}\ \mathit{relation}\ \mathit{rational\_const} \\
\mathit{variable} & ::= & \mathit{real\_variable} \mid \mathit{boolean\_variable} \\
\mathit{relation} & ::= & < \mid \leq \mid = \mid \geq \mid >
\end{array}$$

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

$$
\begin{array}{rcl}
\mathit{equation} & ::= & \mathit{triplet} \mid \mathit{pair} \\
\mathit{triplet} & ::= & \mathit{real\_variable} = \mathit{real\_variable}\ \mathit{bop}\ \mathit{real\_variable} \\
\mathit{pair} & ::= & \mathit{real\_variable} = \mathit{uop}\ \mathit{real\_variable}
\end{array}
$$

where $\mathit{bop}, \mathit{uop}$ are binary and unary operation symbols, including $+$, $-$, $\times$, sin, etc., and $\mathit{rational\_const}$ ranges over the rational constants.

Such constraint formulae are interpreted over valuations $\sigma \in (BV \xrightarrow{\text{total}} \mathbb{B}) \times (RV \xrightarrow{\text{total}} \mathbb{R})$, where BV is the set of Boolean and RV the set of real-valued variables. $\mathbb{B}$ is identified with the subset $\{0,1\}$ of $\mathbb{R}$, so that literals v and $\neg v$ can be encoded by appropriate rational-valued bounds, e.g. $v \geq 1$ or $v \leq 0$. The definition of satisfaction is standard: a constraint formula ϕ is satisfied by a valuation 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. In order to make all arithmetic operators total, we extend their codomain (as well as, for compositionality, their domain) with a special value $\mho \notin \mathbb{R}$ such that the operators manipulate values in $\mathbb{R}^{\mho} = \mathbb{R} \cup \{\mho\}$. The comparison operations on $\mathbb{R}$ are extended to $\mathbb{R}^{\mho}$ in such a way that $\mho$ is incomparable to any real number, that is, $c \not\sim \mho$ and $\mho \not\sim c$ for any $c \in \mathbb{R}$ and any relation $\sim \in \{<, \leq, =, \geq, >\}$.

Instead of real-valued valuations of variables, our constraint solving algorithm manipulates interval-valued valuations $\rho \in (BV \xrightarrow{\text{total}} \mathbb{I}_{\mathbb{B}}) \times (RV \xrightarrow{\text{total}} \mathbb{I}_{\mathbb{R}})$, where $\mathbb{I}_{\mathbb{B}} = 2^{\mathbb{B}}$ and $\mathbb{I}_{\mathbb{R}}$ is the set of convex subsets of $\mathbb{R}^{\mho}$.[2] Slightly abusing notation, we write $\rho(l)$ for $\rho_{\mathbb{I}_{\mathbb{B}}}(l)$ when $\rho = (\rho_{\mathbb{I}_{\mathbb{B}}}, \rho_{\mathbb{I}_{\mathbb{R}}})$ and $l \in BV$, and similarly $\rho(x)$ for $\rho_{\mathbb{I}_{\mathbb{R}}}(x)$ when $x \in RV$. In the following, we occasionally use the term *box* synonymously for interval valuation. If both σ and η are interval valuations then σ is called a *refinement* of η iff $\sigma(v) \subseteq \eta(v)$ for each $v \in BV \cup RV$.

In order to lift a binary operation $\circ$ and its partial inverses to sets, we define

$$
\begin{array}{rcl}
m \bullet_1 n & = & \{x \mid \exists y \in m, z \in n : x = y \circ z\}, \\
m \bullet_2 n & = & \{y \mid \exists x \in m, z \in n : (x = y \circ z \vee \forall y' \in \mathbb{R} : (x \neq y' \circ z \wedge y = \mho))\}, \\
m \bullet_3 n & = & \{z \mid \exists x \in m, y \in n : (x = y \circ z \vee \forall z' \in \mathbb{R} : (x \neq y \circ z' \wedge z = \mho))\},
\end{array}
$$

and similarly for unary $\circ$:

$$
\begin{array}{rcl}
\bullet_1 m & = & \{x \mid \exists y \in m : x = \circ y\}, \\
\bullet_2 m & = & \{y \mid \exists x \in m : (x = \circ y \vee \forall y' \in \mathbb{R} : (x \neq \circ y' \wedge y = \mho))\}.
\end{array}
$$

Note that these are essentially the images of the argument sets under the relation $\{(x,y,z) \mid x = y \circ z\}$ (or $\{(x,y) \mid x = \circ y\}$, resp.) when substituting the respective arguments. We lift these set-valued operators to (computer-representable) intervals by assigning to each set-valued operation $\bullet$ a conservative interval approximation $\hat{\bullet}$ which satisfies $i_1 \hat{\bullet} i_2 \in \mathbb{I}_{\mathbb{R}}$ and $i_1 \hat{\bullet} i_2 \supseteq i_1 \bullet i_2$ for all intervals i_1 and i_2 [Moo66]. Note that the definition of an interval extension does not specify how to exactly lift a set operation $\bullet$ to intervals, but leaves some design choice by permitting arbitrary overapproximations. For the sake of reasoning power, $i_1 \hat{\bullet} i_2$ should be chosen such that it provides an as tight as possible overapproximation of $i_1 \bullet i_2$. Thich means that in practice $i_1 \hat{\bullet} i_2$ is the *interval hull* of $i_1 \bullet i_2$,

[2] Note that this definition covers the open, half-open, and closed intervals over $\mathbb{R}$, including unbounded intervals, as well as the union of such intervals with $\{\mho\}$.

that is, $\bigcap_{i\in\mathbb{I}_\mathbb{R}, i\supseteq i_1 \bullet i_2} i$ extended by some outward rounding to compensate for the imprecision of computer arithmetic and the finiteness of the set of floating-point numbers.

For the manipulated interval valuations we adapt the common notion of *hull consistency* (cf. [BG06]) from interval constraint propagation (cf. Sect. 3) which our algorithm will try to enforce by reasoning steps. We call an interval valuation ρ *hull consistently satisfying* for a constraint formula ϕ, denoted $\rho \models_{\mathrm{hc}} \phi$, iff each clause of ϕ contains at least one hull consistently satisfied atom. *Hull consistent satisfaction* of atoms is defined as follows:

$$\begin{array}{llll}
\rho \models_{\mathrm{hc}} x \sim c & \text{iff} & \rho(x) \subseteq \{u \mid u \in \mathbb{R}, u \sim c\} & \text{for} \quad x \in RV \cup BV, c \in \mathbb{Q} \\
\rho \models_{\mathrm{hc}} x = y \circ z & \text{iff} & \rho(x) \supseteq \rho(y) \hat{\bullet}_1 \rho(z), & \\
 & & \rho(y) \supseteq \rho(x) \hat{\bullet}_2 \rho(z), & \\
 & & \rho(z) \supseteq \rho(x) \hat{\bullet}_3 \rho(y) & \text{for} \quad x, y, z \in RV, \circ \in bop \\
\rho \models_{\mathrm{hc}} x = \circ y & \text{iff} & \rho(x) \supseteq \hat{\bullet}_1 \rho(y), & \\
 & & \rho(y) \supseteq \hat{\bullet}_2 \rho(x) & \text{for} \quad x, y \in RV, \circ \in uop.
\end{array}$$

We call a formula ϕ *hull consistently satisfiable*, denoted $\mathrm{hcsat}\,(\phi)$, iff there is an interval valuation ρ with $\rho \models_{\mathrm{hc}} \phi$ and $\rho(v) \neq \emptyset$ for all $v \in BV \cup RV$. Note that hull consistent satisfiability is a necessary, yet not sufficient condition for real-valued satisfiability. It is in general not sufficient, as can be seen from the example $(x = x \cdot x) \wedge (x > 0) \wedge (x < 1)$, which is hull consistently satisfied by $\rho(x) = (0, 1)$, yet not satisfiable over the reals.

When solving satisfiability problems of formulae with Davis-Putnam-like procedures, we will build interval valuations incrementally by successively contracting intervals through constraint propagation and branching. This may lead to situations where an interval valuation does no longer contain any solution, in which case we have to revert some branching decisions previously taken. In order to detect this —in general undecidable— situation, we define a sufficient criterion for non-existence of a solution within the interval valuation: We say that an interval valuation ρ is *inconsistent with an atom* a, denoted $\rho \sharp a$, iff the left- and right-hand sides of the atom have disjoint valuations under ρ, i.e.

$$\begin{array}{lll}
\rho \sharp x \sim c & \text{iff} \;\; \rho(x) \cap \{u \mid u \in \mathbb{R}, u \sim c\} = \emptyset & \text{for } x \in RV \cup BV, c \in \mathbb{Q} \\
\rho \sharp x = y \circ z & \text{iff} \;\; \rho(x) \cap \rho(y) \hat{\bullet}_1 \rho(z) = \emptyset & \text{for } x, y, z \in RV, \circ \in bop \\
\rho \sharp x = \circ y & \text{iff} \;\; \rho(x) \cap \hat{\bullet}_1 \rho(y) = \emptyset & \text{for } x, y \in RV, \circ \in uop
\end{array}$$

Note that deciding inconsistency of an interval valuation with an atom (and hence, with a clause or a formula) is straightforward, as is deciding hull consistent satisfaction of an atom (clause, formula) by an interval valuation. If ρ is neither hull consistently satisfying for ϕ nor inconsistent with ϕ then we call ϕ *inconclusive on* ρ, which is again decidable.

3. Algorithmic basis

Our constraint solving approach builds upon the well-known techniques of interval constraint propagation (ICP) and of propositional SAT solving by the DPLL procedure plus its more recent algorithmic enhancements.

Interval constraint propagation is one of the sub-topics of the area of constraint programming where constraint propagation techniques are studied in various, and often discrete, domains. For the domain of the real numbers, given a constraint ϕ and a floating-point box B, so-called *contractors* compute another floating-point box $C(\phi, B)$ such that

$C(\phi, B) \subseteq B$ and such that $C(\phi, B)$ contains all solutions of ϕ in B (cf. the notion of *narrowing operator* [BMVH94, Ben96]).

There are several methods for implementing such contractors. The most basic method [Dav87, Cle87] decomposes all atomic constraints (i.e., constraints of the form $t \geq 0$ or $t = 0$, where t is a term) into conjunctions of so-called primitive constraints resembling three-address code (i.e., constraints such as $x + y = z$, $xy = z$, $z \in [\underline{a}, \overline{a}]$, or $z \geq 0$) by introducing additional auxiliary variables (e.g., decomposing $x + \sin y \geq 0$ to $\sin y = v_1 \wedge x + v_1 = v_2 \wedge v_2 \geq 0$). Then it applies a contractor for these primitive constraints until a fixpoint is reached.

We illustrate contractors for primitive constraints using the example of a primitive constraint $x + y = z$ with the intervals $[1, 4]$, $[2, 3]$, and $[0, 5]$ for x, y, and z, respectively: 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 $[3, 5]$. Contractors for other primitive constraints can be based on interval arithmetic in a similar way. 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.

There are several variants, alternatives and improvements of the approach described above (cf. [BG06] for a survey of the literature). These do in particular deal with stronger contractors based on non-decomposed constraints. While such could easily be included into our approach, the description in the remainder will concentrate on the simple contractors available on the decomposed form. The reasons for doing so stem from our application context: while in merely conjunctive constraint systems, non-decomposed constraints are clearly better due to their stronger contractors, completely different aspects become dominant in large and complex-structured Boolean combinations of arithmetic constraints. Here, pruning of the intervals is no longer the only forward inference mechanism, but pruning of the search space originating from the Boolean structure based on inferences from the theory side becomes at least equally important. There, decomposed constraints have their advantage, as they permit generation of more concise reasons (cf. Sect. 4.3). It would, however, be feasible to have both the decomposed and non-decomposed forms of the constraints and their respective contractors coexist in our system, joining their virtues.

Dealing with partial operations, our implementation associates to each constraint $x = y \circ z$ in the three-address decomposition the contractor $\rho'(x) := \rho(x) \cap \rho(y) \hat{\bullet}_1 \rho(z)$ as well as all the related solved-form contractors $\rho'(y) := \rho(y) \cap \rho(x) \hat{\bullet}_2 \rho(z)$ and $\rho'(z) := \rho(z) \cap \rho(x) \hat{\bullet}_3 \rho(y)$. Together, this set of contractors is essentially equivalent to the usual contractor for primitive constraints [Cle87], yet we take the different solved forms as being independent contractors in order to be able to trace the reasons for contractions within conflict diagnosis. Note that each individual contraction $B' = C(e, B)$ can be decomposed into a set of individual contractions affecting just one face of B each, and each having a subset of the bounds describing the faces of B as a reason. E.g., in the above example, the first interval contraction derives the new bound $x \leq 3$ from the reasons $y \geq 2$ and $z \leq 5$, using equation $x + y = z$ in its solved form $x = z - y$. In the sequel, we denote such an atomic derivation of ICP by $(y \geq 2, z \leq 5) \overset{x+y=z}{\leadsto} (x \leq 3)$.

Propositional SAT solving. The *Propositional Satisfiability Problem* (SAT) is a well-known NP-complete problem, with extensive applications in various fields of computer science and engineering. In recent years a lot of developments in creating powerful SAT algorithms have been made, leading to state-of-the-art approaches like BerkMin [GN02], MiniSat [ES03], MiraXT [LSB07], and zChaff [MMZ+01]. All of them are enhanced variants of the classical backtrack search DPLL procedure [DLL62].

Given a Boolean formula ϕ in *Conjunctive Normal Form* (CNF) and a partial valuation ρ, which is empty at the beginning of the search process, a backtrack search algorithm incrementally extends ρ until either $\rho \models \phi$ holds or ρ turns out to be inconsistent for ϕ. In the latter case another extension of ρ is tried through backtracking. Each decision step is followed by the *deduction phase*, involving the search for *unit clauses*, i.e. clauses that have only one unassigned literal left while all other literals are assigned incorrectly in the current valuation ρ. Obviously, unit clauses require certain assignments in order to preserve their satisfiability, where the execution of the implied assignments itself might force further assignments. In the context of SAT solving such necessary assignments are also referred to as *implications*. To perform the deduction phase in an efficient manner, zChaff introduced a *lazy clause evaluation* technique based on *Watched Literals* (WL): for each clause two literals are selected in such a way that they either are both unassigned or at least one of them is satisfying the clause. So, if at some point during the search one of the WLs is getting assigned incorrectly, a new WL for the corresponding clause has to be found. If such a literal does not exist and the second WL is still unassigned, the clause is forcing an implication. As a consequence of this method there is no need to check all clauses after making a decision step, but only those for which a WL is getting assigned incorrectly.

However, deduction may also yield a *conflicting clause* which has all its literals assigned false, indicating the need for backtracking. To avoid repeated conflicts due to the same reason, modern SAT algorithms incorporate *conflict-driven learning* to derive a sufficiently general explanation (a combination of variable assignments) for the actual conflict. Based on that (ideally minimal) set of assignments that triggered the particular conflict, a *conflict clause* is generated and added to the clause set to guide the subsequent search. The conflict clause is also used to compute the backtrack level, which is defined as the maximum level the SAT algorithm has to backtrack to in order to solve the conflict. This approach leads to a *non-chronological backtracking* operation, often jumping back more than just one level, thus making conflict-driven learning combined with non-chronological backtracking a powerful mechanism to prune large parts of the search space [ZMMM01].

4. Integrating interval constraint propagation and SAT

As can be seen from Table 1, branch-and-prune algorithms based on interval constraint propagation (ICP) with interval splitting and the core algorithm of DPPL SAT solving share a similar structure. This similarity motivates a tighter integration of propositional

		Interval Constraint Solving	**DPLL SAT**
Given:		Constraint set $C = \{c_1, \ldots, c_n\}$, initial box $B \subseteq \mathbb{R}^{\lvert \text{free}(C) \rvert}$	Clause set $C = \{c_1, \ldots, c_n\}$, initial box $B \subseteq \mathbb{B}^{\lvert \text{free}(C) \rvert}$
Goal:		Find box $B' \subseteq B$ containing satisfying valuations throughout *or* show non-existence of such B'.	
Alg.:	1.	$L := \{B\}$	
	2.	If $L \neq \emptyset$ then take	
		some vs.	*most recently added*
		box $b :\in L$, otherwise report "unsatisfiable" and stop.	
	3.	To determine subbox $b' \subseteq b$ containing all solutions in b, use	
		contractor C vs.	*unit propagation.*
	4.	If $b' = \emptyset$ then set $L := L \setminus \{b\}$, goto 2.	
	5.	Check whether b' satisfies all	
		constraints vs.	*clauses*
		in C; if so then report b' as satisfying and stop.	
	6.	If $b' \subset b$ then set $L := L \setminus \{b\} \cup \{b'\}$, goto 2.	
	7.	Split b into subintervals b_1 and b_2, set $L := L \setminus \{b\} \cup \{b_1, b_2\}$, goto 2.	

Table 1. Schematic representation of the algorithms underlying interval constraint solving (left) vs. basic DPLL SAT (right). The close analogy suggests a tight integration into a DPLL-style algorithm manipulating large Boolean combinations of arithmetic formulae via a homogeneous treatment of Boolean and arithmetic parts.

SAT and arithmetic reasoning than in classical lazy theorem proving. This tight integration shares the common algorithmic parts, thereby providing the SAT solver with full control over and full introspection into the ICP process. This way, recent algorithmic enhancements of propositional SAT solving, like lazy clause evaluation, conflict-driven learning, and non-chronological backjumping carry over to ICP-based arithmetic constraint solving. In particular, we are able to learn forms of conflicts that are considerably more general than classical as well as generalized nogoods [KB03] in search procedures for constraint solving.

4.1 Introductory example

Before providing a formal exposition of our satisfiability solving algorithm in the next section, we explain it by means of an example. Given the formula $\psi = (x = \cos(y) \vee y = \sin(x)) \wedge (x < 0 \vee x^2 = -3 \cdot y)$, where x and y are real-valued variables with ranges $x \in [-10, 30]$ and $y \in [-8, 25]$, the algorithm first rewrites ψ into an equisatisfiable CNF

$$\varphi = (x = \cos(y) \vee y = \sin(x)) \wedge (x < 0 \vee b) \wedge (\neg b \vee h = -3 \cdot y) \wedge (\neg b \vee h = x^2)$$

of linear size (cf. Sect. 2), where $h \in \mathbb{R}$ and $b \in \mathbb{B}$ are fresh auxiliary variables. Note that b and $\neg b$ are just abbreviations for the numeric constraints $b \geq 1$ and $b \leq 0$, respectively.

Each atom of the formula is inconclusive under the initial interval valuation ρ of the variables given by $\rho(x) = [-10, 30]$, $\rho(y) = (-8, 25]$, $\rho(h) = \mathbb{R}$, and $\rho(b) = \mathbb{B}$. Thus, the whole formula φ is inconclusive. To start reasoning, we branch the search space, as known from DPLL and ICP: we split the interval of y and decide $y > 1$, thus opening decision level 1. Under this new interval assignment ρ_1, where $\rho_1(y) = (1, 25]$ and $\rho_1(v) = \rho(v)$ for

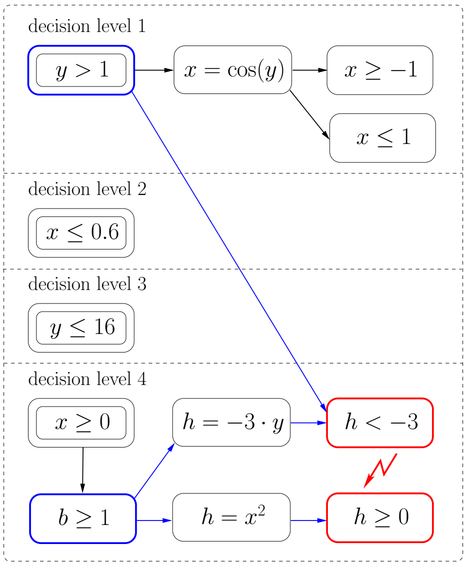

Figure 1. Decided and implied bounds depicted as a graph structure. Bounds in the blue boxes imply the conflict (red boxes).

$v \neq y$, the atom $y = \sin(x)$ becomes inconsistent since

$$\begin{array}{rlcl} & \rho_1(y) & \cap & \sin(\rho_1(x)) \\ = & (1, 25] & \cap & [-1, 1] \\ = & \emptyset & & \end{array} .$$

Therefore, the first clause becomes unit in analogy to DPLL-style SAT solving, as all atoms but one in this clause are inconsistent. To satisfy the whole formula φ under (a refinement of) ρ_1, the atom $x = \cos(y)$ thus has to hold. From $x = \cos(y)$ we conclude $x \geq -1$ and $x \leq 1$, which leads to a refined interval valuation ρ_2 with $\rho_2(x) = [-1, 1]$. No further deductions are possible.

Thus, another split of an interval is performed by deciding $x \leq 0.6$ (decision level 2). This split does not cause any deduction. The same holds for the next decision $y \leq 16$ at decision level 3.

After deciding $x \geq 0$ (opening decision level 4), we obtain the interval valuation ρ_3 with $\rho_3(x) = [0, 0.6], \rho_3(y) = (1, 16]$. The atom $x < 0$ in the second clause becomes inconsistent under ρ_3. Thus b is implied. From $b \geq 1$ and the third and fourth clause becoming unit, it follows that the equations $h = -3 \cdot y$ and $h = x^2$ have to be satisfied. Thus, we can derive new interval borders for the variables from these equations by ICP. We do not call a subordinate interval constraint solver for this issue, but instead apply the corresponding contractor for each equation (cf. Sect. 3) locally in order to be able to combine interval constraint propagations and unit propagations in a single implication queue and implication graph, permitting their uniform treatment within conflict detection and conflict-driven learning.

The atoms $y > 1$ and $h = -3 \cdot y$ together imply $h < -3$ by interval constraint propagation. The corresponding bound $h < -3$ is thus asserted and ρ_3 narrowed accordingly. On the other hand, $h = x^2$ yields $h \geq 0$, also by ICP. Performing the corresponding narrowing, we encounter an empty interval valuation, as the models of $h < -3$ and of $h \geq 0$ have an empty intersection. As in propositional SAT solving, we analyze that conflict by scanning its reasons. When we trace back the reasons for this conflict situation, we find that our first decision $y > 1$ and the bound $b \geq 1$ deduced on decision level 4 are reasons for the conflict (as illustrated in Fig. 1). (More formally, determining the reasons for a conflict is done by conflict analysis involving an implication graph like in DPLL SAT solvers, cf. Sect. 4.3.) In order to investigate another part of the search space not considered so far, we jump back to the second largest decision level d contributing to the conflict and undo all decisions and deductions up to there (excluding d). In our example we jump back to decision level 1, and after undoing the corresponding decisions and deductions, the current interval valuation again is ρ_2.

Moreover, similar to DPLL learning mechanisms we add a disjunction of the negated reasons as a conflict clause to the formula, in order to avoid visiting the same branch again. Hence, we add the conflict clause $(y \leq 1 \vee \neg b)$. We use an approach guaranteeing that the conflict clause is always unit after undoing decisions and deductions as mentioned above, e.g. the unique implication point technique from [ZMMM01] (cf. Sect. 4.3). Hence, we deduce the Boolean literal $\neg b$ on decision level 1 based on our first decision $y > 1$. The bound $b \leq 0$ implies that the second clause becomes unit and propagates $x < 0$. From $x < 0$, $y > 1$, and $x = \cos(y)$ ICP deduces $y > 1.57$. The current interval valuation ρ_4 thus is $\rho_4(x) = [-1, 0)$, $\rho_4(y) = (1.57, 25]$, $\rho_4(b) = [0, 0]$, and $\rho_4(h) = \mathbb{R}$.

Assume that by further decisions and deductions the intervals of x and y were narrowed down to $[-1, -0.65]$ and $[3, 4]$, resp., while all other intervals remaining unchanged. Let ρ_5 be the according interval valuation. We reached a state of the search space where we know that clauses two, three, and four are definitely satisfiable because at least one atom of each of these clauses is definitely satisfiable. The latter is true as each value of the current interval $\rho_5(x) = [-1, -0.65)$ assigned to x (resp. $\rho_5(b) = [0, 0]$ assigned to b) satisfies the bound $x < 0$ (resp. $\neg b$). However, the same does not hold for $x = \cos(y)$. In general, we can only deduce from an interval valuation that equation e is satisfiable over the reals if all intervals of the variables occurring in e are point intervals in the interval valuation. Reaching point intervals cannot be expected by naive splitting and ICP. Therefore, in general, our algorithm may output the result "unknown" in addition to "satisfiable" and "unsatisfiable".

In section 4.5 we discuss approaches for finding satisfying valuations. In our example, we know that for each value $v_y \in \rho_5(y) = [3, 4]$ there is a value $v_x \in \rho_5(x) = [-1, -0.65]$ s.t. $v_x = \cos(v_y)$. The value for x of a solution depends on the value for y and the equation $x = \cos(y)$. However, the value for x does not depend on any other values or equations. Hence, we can assert that φ, and thus ψ, is satisfiable.

4.2 Interval constraint solving as a multi-valued SAT problem.

The underlying idea of our algorithm is that the two central operations of ICP-based arithmetic constraint solving —interval contraction by constraint propagation and by interval splitting— correspond to asserting bounds on real-valued variables $v \sim c$ with $v \in RV$, $\sim \in \{<, \leq, \geq, >\}$ and $c \in \mathbb{Q}$. Likewise, the decision steps and unit propagations in DPLL proof search correspond to asserting literals. A unified DPLL- and ICP-based proof search on a formula ϕ from the formula language of Sect. 2 can thus be based on asserting or retracting atoms of the formula language, thereby in lockstep refining or widening an interval valuation ρ that represents the current set of candidate solutions.

In our algorithm we use the letter M to denote the list of asserted atoms. In order to allow backtracking on this data-structure, in addition, we intersperse a special marker symbol $|$ into this list M. Since M may contain several atoms with bounds on the same variable, it is costly to re-construct from M the tightest of these bounds for a given variable. Hence, in addition to M, we maintain a stack of interval assignments Σ, where each element of Σ stores the collected information about the bounds in M up to a marker $|$ and the top element ρ of Σ stores the information about all the bounds currently in M. The algorithm thus maintains a 3-tuple (Σ, M, ϕ) as its proof state, where Σ is a stack of interval assignments, M a list of asserted atoms [3] cut by a special marker symbol $|$, and ϕ a formula. For the basic procedure, ϕ will remain constant and always be equal to the formula to be solved. It is not before introducing conflict-driven learning that we will see changes to ϕ.

The procedure searching for satisfying valuations then proceeds as follows:

Step 1. Proof search on the input formula ϕ starts with an empty set of asserted atoms and the stack Σ of interval valuations just containing a single interval valuation ρ being the minimal element $\rho_\perp$ wrt. the refinement relation on interval valuations, that is, all intervals being maximal wrt. the value domains of the individual variables. I.e.,

$$\rho_\perp(v) = \begin{cases} \mathbb{B} & \text{if } v \in BV, \\ \mathbb{R} \cup \{\mho\} & \text{if } v \in RV \text{ is an auxiliary variable introduced by} \\ & \text{rewriting to three-address form,} \\ X & \text{if } v \in RV \text{ is a problem variable with range } X \in \mathbb{I}_\mathbb{R}. \end{cases}$$

The start state of the search procedure thus is $(\langle \rho_\perp \rangle, \varepsilon, \phi)$

Step 2. Proof search continues with searching for *unit clauses* in ϕ, that is, clauses that have all but one atoms being inconsistent with the current interval valuation ρ. If such a clause is found then the remaining atom is asserted:

$$(1) \qquad \frac{\phi = \phi' \wedge (a_1 \vee \ldots \vee a_n), j \in \mathbb{N}_{\leq n}, \forall i \in \mathbb{N}_{\leq n} \,.\, (i \neq j \;\Rightarrow\; (\rho \sharp a_i)), a_j \notin M}{(\Sigma \cdot \langle \rho \rangle, M, \phi) \longrightarrow (\Sigma \cdot \langle \rho \rangle, M \cdot \langle a_j \rangle, \phi)}$$

Step 2 is repeated until all unit clauses have been processed.

Step 3. If there is an asserted atom a that yields contractions narrowing the current interval valuation ρ, then the contractors corresponding to a are applied to ρ. In the case where the asserted atom is a bound, this is straightforward.

$$(2) \qquad \frac{(v \sim c) \in M, \rho \not\models_{\mathrm{hc}} v \sim c}{(\Sigma \cdot \langle \rho \rangle, M, \phi) \longrightarrow (\Sigma \cdot \langle \mathit{update}_\rho(v \sim c) \rangle, M, \phi)}$$

where $\mathit{update}_\rho(v \sim c)(v') = \rho(v) \cap \{x \mid x \sim c\}$, if $v' = v$, and $\rho(v')$, otherwise.

In the case of triplets and pairs, these contractors are the usual contractors for the primitive

[3] Note that we are allowing arbitrary theory atoms here, whereas SMT maintains a list of asserted Boolean literals some of which abbreviate theory atoms occurring in the formula. This generalization permits asserting freshly generated theory atoms at any time, without the need to generate fresh clauses introducing fresh theory atoms, as necessary in direct extensions of DPLL(T) supporting generation of fresh arithmetic atoms on demand [BNOT06].

constraints of ICP, as explained in Sect. 3.[4] Beyond contracting ρ, the contractions obtained from triplets and pairs are in turn asserted as bounds (this is redundant for contractions stemming from bounds, as the asserted atoms would be equal to the already asserted bound which effected the contraction).

$$\frac{e, b_1, \ldots, b_n \in M, (b_1, \ldots, b_n) \overset{e}{\rightsquigarrow} (v \sim c)\,, \rho \not\models_{\mathrm{hc}} v \sim c}{(\Sigma \cdot \langle \rho \rangle, M, \phi) \longrightarrow (\Sigma \cdot \langle \mathit{update}_{\rho}(v \sim c) \rangle, M \cdot \langle v \sim c \rangle, \phi)} \tag{3}$$

where e is an equation and the b_i are bounds. For efficiency reasons, b_1 to b_n are in practice inferred from ρ, which when viewed as a polytope has faces reflecting these bounds, rather than retrieving them from the list M.

Note that rule (3) is different in spirit from theory-related rules in lazy theorem proving and DPLL(T), as it does neither analyze consistency of the currently asserted set of theory atoms (as in lazy TP and DPLL(T)) nor implicative relations between these and other theory atoms occurring in the input formula (as in DPLL(T)). Instead, it applies purely local reasoning with respect to the single theory atom e and the bounds (i.e., domain restrictions) b_1 to b_n, generating a fresh bound atom $v \sim c$ not present in the original formula. Consistency of asserted theory atoms is never tested on the full set of (non-linear) theory atoms, but only within the extremely simple sub-theory of bound atoms through rules (5) and (6), based on the bookkeeping pursued by rule (2). The massive generation of fresh bound atoms not occurring in the input formula is crucial to this process, providing its delayed consistency check in the theory of bounds with the full power of ICP-based consistency checks, but more fine-granular conflict analysis (cf. rule (7)).

Step 3 is repeated until contraction detects a conflict in the sense of some interval $\rho(v)$ becoming empty, which is handled by continuing at step 5, or until no further contraction is obtained.[5] In the latter case, the algorithm checks if contraction of ρ has yielded new unit clauses, in which case it re-enters step 2.

Step 4. Otherwise, it applies a *splitting step*: it selects a variable $v \in BV \cup RV$ that is interpreted by a non-point interval (i.e., $|\rho(v) \cap \mathbb{R}| > 1$) and splits its interval $\rho(v)$ by asserting a bound that contains v as a free variable and which is inconclusive on ρ. Note that the complement of such an assertion also is a bound and is inconclusive on ρ too. In our current implementation, we use the usual strategy of bisection, i.e., the choice of c as the midpoint of $\rho(v)$.

$$\frac{v \sim c \text{ inconclusive on } \rho, v \text{ occurs in } \phi}{(\Sigma \cdot \langle \rho \rangle, M, \phi) \longrightarrow (\Sigma \cdot \langle \rho, \mathit{update}_{\rho}(v \sim c) \rangle, M \cdot \langle |, v \sim c \rangle, \phi)} \tag{4}$$

Note that the topmost environment ρ in the stack is duplicated before splitting s.t. ρ can easily be retrieved upon backtracking. After the split, the algorithm continues at 2.

Step 5. In case of a conflict, some previous splits (cf. step 4) have to be reverted, which is achieved by backtracking —thereby undoing all assertions being consequences of the split via retrieval of the previous interval valuation in the stack— and by asserting the complement

[4] Note that these contractors are defined on floating point intervals, ensuring correctness due to outward rounding. Hence we also use floating point intervals throughout the algorithm.

[5] In practice, one stops as soon as the changes become negligible.

of the previous split. In M, the split is marked by the special symbol | preceding the atom asserted by the split.

$$\text{(5)} \qquad \frac{\rho'(w) = \emptyset \text{ for some } w \in BV \cup RV, | \notin M'}{(\Sigma \cdot \langle \rho, \rho' \rangle, M \cdot \langle |, v \sim c \rangle \cdot M', \phi) \longrightarrow (\Sigma \cdot \langle \mathit{update}_{\rho}(v \not\sim c) \rangle, M \cdot \langle v \not\sim c \rangle, \phi)}$$

Later (Sect. 4.3 below) we will see that, beyond backtracking, information about the reason for the conflict can be recorded in the formula ϕ, thus pruning the remaining search space. If rule (5) is applicable, i.e. if there is an open backtracking point marked by the split marker "|" in the list of asserted atoms, then we apply the rule and proceed to step 2. Otherwise, i.e. if there is no previous split with a yet unexplored alternative, then the algorithm stops with result "unsatisfiable".

$$\text{(6)} \qquad \frac{\rho(w) = \emptyset \text{ for some } w \in BV \cup RV, | \notin M}{(\Sigma \cdot \langle \rho \rangle, M, \phi) \longrightarrow \text{unsat}}$$

The correctness of the algorithm rests on the following two invariance properties preserved by rules (1) to (5):

Lemma 1. *Assume* $(\langle \rho_{\perp} \rangle, \varepsilon, \phi) \longrightarrow^{*} (\Sigma \cdot \langle \rho \rangle, M, \phi) \longrightarrow (\Sigma', M', \phi)$. *Then*

1. $\eta \models_{\text{hc}} \bigwedge_{a \in M} a$ *implies that* η *is a (not necessarily proper) refinement of* ρ,
2. $\text{hcsat}\left(\phi \wedge \left(\bigwedge_{a \in M} a \vee \neg \bigwedge_{b \in C_M} b\right)\right)$ *implies* $\text{hcsat}\left(\phi \wedge \left(\bigwedge_{a' \in M'} a' \vee \neg \bigwedge_{b' \in C_{M'}} b'\right)\right)$

hold, where $C_N = \{x \sim c \mid N = N_1 \cdot \langle |, x \sim c \rangle \cdot N_2\}$ *is the set of choice points in* N.

Proof. Property 1 follows from the fact that for each contraction applied to ρ, a corresponding bound can be found in M (including bounds added within the same step that the contraction occurs). Hence, if $\rho(v) = [a, b]$ and if $[a, b]$ is a proper subset of the original range of v then there is a bound $x \geq a \in M$ and a bound $x \leq b \in M$, and analogously for half-open intervals $\rho(v) = (a, b]$ or $\rho(v) = [a, b)$ or open intervals $\rho(v) = (a, b)$. Consequently, $\eta \models_{\text{hc}} M$ implies $\eta(v) \subseteq [a, b] = \rho(v)$ in case of $\rho(v) = [a, b]$, and similarly for the other cases. I.e., η is a refinement of ρ whenever $\eta \models_{\text{hc}} M$.

Property 2 requires a case analysis wrt. the changes applied to M: Within rules (1) to (3), M is expanded by deduced atoms a' that the different contractors (unit propagation, interval contraction) permit to be drawn from $\phi \wedge \bigwedge_{a \in M} a$ with respect to refinements of ρ. Each such atom a' satisfies the contractor soundness condition $\eta \models_{\text{hc}} \phi \wedge \bigwedge_{a \in M} a$ iff $\eta \models_{\text{hc}} \phi \wedge \bigwedge_{a \in M} a \wedge a'$ for each refinement η of ρ. As property 1 shows that $\phi \wedge \bigwedge_{a \in M} a$ can only be satisfied by refinements of ρ, the conjectured implication follows for rules (1) to (3) from the fact that $M' = M \cup \{a'\}$ and $C_{M'} = C_M$. The splitting rule (4) adds a split bound $x \sim c$ which occurs in both M' and $C_{M'}$ such that the conjectured implication holds due to absorption. For rule (5) we observe that due to the premise $\rho(w) = \emptyset$ of the rule, property 1 of the Lemma gives $\eta(w) = \emptyset$ for each interval valuation η with $\eta \models_{\text{hc}} \bigwedge_{a \in M} a$. I.e., $\bigwedge_{a \in M} a$ is not hull consistently satisfiable. Consequently, either $\phi \wedge \left(\bigwedge_{a \in M} a \vee \neg \bigwedge_{b \in C_M} b\right)$ is not hull consistently satisfiable (in which case the implication trivially follows), or $\phi \wedge \neg \bigwedge_{b \in C_M} b$ is hull consistently satisfiable. The latter implies hull consistent satisfiability of $\phi \wedge \left(\bigwedge_{a' \in M'} a' \vee \neg \bigwedge_{b' \in C_{M'}} b'\right)$, as $C_{M'} = C_M \setminus \{v \sim c\}$ and $v \not\sim c \in M'$ for some bound $v \sim c$. Rule (6), finally, does not yield a configuration of the form (Σ', M', ϕ) such that it trivially satisfies the conjecture. □

Corollary 1. *If* $(\langle \rho_\perp \rangle, \varepsilon, \phi) \longrightarrow^* (\Sigma \cdot \langle \rho \rangle, M, \phi) \longrightarrow \text{unsat}$ *then* ϕ *is unsatisfiable over* $\mathbb{R}$.

Proof. We assume that $(\langle \rho_\perp \rangle, \varepsilon, \phi) \longrightarrow^* (\Sigma \cdot \langle \rho \rangle, M, \phi) \longrightarrow \text{unsat}$. Then, according to the premises of rule (6), $\rho(w) = \emptyset$ for some $w \in BV \cup RV$ and, furthermore, $| \not\in M$. By induction over the length of the derivation sequence and Lemma 1, property 2, we obtain that $\text{hcsat}\,(\phi)$ implies $\text{hcsat}\,\left(\phi \wedge \left(\bigwedge_{a \in M} a \vee \neg \bigwedge_{b \in C_M} b\right)\right)$. As $| \not\in M$ and thus $C_M = \emptyset$, this in turn gives: $\text{hcsat}\,(\phi)$ implies $\text{hcsat}\,\left(\phi \wedge \bigwedge_{a \in M} a\right)$. According to Lemma 1, property 1, any interval valuation η with $\eta \models_{\text{hc}} \phi \wedge \bigwedge_{a \in M} a$ is a refinement of ρ, i.e. has $\eta(w) = \emptyset$. Thus, $\phi \wedge \bigwedge_{a \in M} a$ is not hull consistently satisfiable, which implies that ϕ is not hull consistently satisfiable. As hull consistent satisfiability is a necessary condition for satisfiability over the reals, it follows that ϕ is unsatisfiable. □

4.3 Algorithmic enhancements

By its similarity to DPLL algorithms, this base algorithms lends itself to all the algorithmic enhancements and sophisticated data structures that were instrumental to the impressive recent gains in propositional SAT solver performance.

Lazy clause evaluation. In order to save costly visits to and evaluations of disjunctive clauses, we extend the lazy clause evaluation scheme of zChaff [MMZ+01] to our more general class of atoms: within each clause, we select two atoms which are inconclusive wrt. the current valuation ρ, called the "watched atoms" of the clause. Instead of scanning the whole clause set for unit clauses in step 2 of the base algorithm, we only visit the clause if a free variable of one of its two watched atoms is contracted, that is, a tighter bound than recorded in $\text{top}(\Sigma)$ is asserted. In this case, we evaluate the atom's truth value. If found to be inconsistent wrt. the new interval assignment, the algorithm tries to substitute the atom by a currently unwatched and not yet inconsistent atom to watch in the future. If this substitution fails due to all remaining atoms in the clause being inconsistent, the clause has become unit and the second watched atom has to be propagated using rule (1).

Maintaining an implication graph. In order to be able to tell reasons for conflicts (i.e., empty interval valuations) encountered, our solver maintains an implication graph IG akin to that known from propositional SAT solving [ZMMM01]. As all asserted atoms are recorded in the stack-like data structure M (cf. Sect. 4.2), the implication graph is implemented by way of pointers providing backward references within M. Each asserted atom in M then has a set of pointers naming the reasons (if any) for its assertion. That is, after application of rule (1), the entry a_j in M is decorated with pointers to the *reasons* for the entries a_i with $i \neq j$ being inconsistent. These reasons are bounds already asserted in M: if a_i in rule (1) is a bound $v \sim c$ then M contains another bound $v \sim' c'$ with $v \sim c \wedge v \sim' c$ being unsatisfiable, in which case $v \sim' c'$ can serve as a reason. If a_i is an equation $x = y \circ z$ then M contains a set of at most 6 bounds for x, y, and z that shows inconsistency of the equation; the bounds in this set then constitute the reasons for a_i being inconsistent. When applying rule (3), the reasons are apparent from the contraction enforced, as explained in Sect. 3: in the contraction $(b_1, \ldots, b_n) \overset{e}{\leadsto} (v \sim c)$, $b_1, \ldots, b_n$ are the reasons for the bound $v \sim c$. The other rules do not record reasons because they either do not assert atoms, as in rule (2), or as the asserted atoms originate in choices (rules 4 and 5), which is recorded by attaching an empty set of reasons to the asserted atom. Note how the homogeneous treatment of Boolean and theory-related reasoning in our framework simplifies extraction of the implication graph: as both Boolean constraint propagations and theory-related constraint propagations are bound assignments in the same list M of asserted atoms, rather than deferring the theory reasoning to a subordinate theory solver checking theory consistency, the implication graph can be maintained via links in M only.

The aforementioned pointer structure is in one-to-one correspondence to an *implication graph* $IG \subset A_M \times A_M$ relating reasons to consequences, where A_M is the set of atoms in M. IG collects all references to reasons occurring in M as follows: $(a, a') \in IG$ iff the occurrence of a' in M mentions a as its reason. Figure 2 provides an example. Given the implication graph IG, the set $R_{IG}(a)$ of sufficient reasons for an atom a in M is defined inductively as the smallest set satisfying the following three conditions.

1. $\{a\} \in R_{IG}(a)$.
2. Let $r \in R \in R_{IG}(a)$ and $S = \{q \mid (q, r) \in IG\}$. If $S \neq \emptyset$ then $(R \setminus \{r\}) \cup S \in R_{IG}(a)$.
3. If $R \in R_{IG}(a)$ and $S \supset R$ then $S \in R_{IG}(a)$.

The rationale of this definition is that (1.) a itself is a sufficient reason for a being true, (2.) a sufficient reason of a can be obtained by replacing any reason r of a with a sufficient reason for r, (3.) any superset of a sufficient reason of a is a sufficient reason of a.

Conflict-driven learning and non-chronological backtracking. In case of a conflict encountered during the search, we can record a reason for the conflict preventing us from constructing other interval valuations provoking a similar conflict. Therefore, we traverse the implication graph IG to derive a reason for the conflict encountered, and add this reason in negated form to the input formula. We use the unique implication point technique [ZMMM01] to derive a conflict clause which is general in that it contains few atoms. This clause becomes asserting upon backjumping to the second largest decision level contributing to the conflict, i.e. upon undoing all decisions and constraint propagations younger than the chronologically youngest but one decision among the antecedents of the conflict.

$$(7) \quad \frac{\begin{array}{c}\rho'(v) = \emptyset, M = M' \cdot \langle | \rangle \cdot M'', b, b' \text{ are bounds}, b, b' \in M, \models \neg(b \wedge b'), |\Sigma| = \|M'\|,\\ a_1, \ldots, a_n \text{ are bounds}, a_1 \in M'', a_2, \ldots, a_n \in M', \{a_1, \ldots, a_n\} \in R_{IG}(b) \cap R_{IG}(b')\end{array}}{(\Sigma \cdot \langle \rho \rangle \cdot \Sigma' \cdot \langle \rho' \rangle, M, \phi) \longrightarrow (\Sigma \cdot \langle \mathit{update}_\rho(\neg a_1) \rangle, M' \cdot \langle \neg a_1 \rangle, \phi \wedge (\neg a_1 \vee \ldots \vee \neg a_n))}$$

where $|\Sigma|$ gives the length of sequence Σ and $\|M\|$ the number of markers $|$ in M. Note that the application conditions of the rule are always satisfied when the conditions of the backtrack rule (5) apply, as $\rho'(v) = \emptyset$ can only arise if there are two contradicting bounds $b = v \sim c$ and $b' = v \sim' c'$ in M, and as deduction sequences can, by following IG long enough, always be traced back to bounds as reasons. Hence, the learning rule (7) can fully replace the backtrack rule (5). An application of the rule is shown in Fig. 2.

Note that, while adopting the conflict detection techniques from propositional SAT solving, our conflict clauses are more general than those generated in propositional SAT

$$\begin{array}{ll} c_1: & (x > 4 \;\vee\; y \leq 0 \;\vee\; b) \\ c_2: & \wedge\; (\neg b \;\vee\; h_1 = x \cdot x) \\ c_3: & \wedge\; (\neg b \;\vee\; h_2 = -2 \cdot y) \\ c_4: & \wedge\; (\neg b \;\vee\; h_3 = h_1 + h_2) \\ c_5: & \wedge\; (\neg b \;\vee\; h_3 \geq 6.2) \end{array}$$

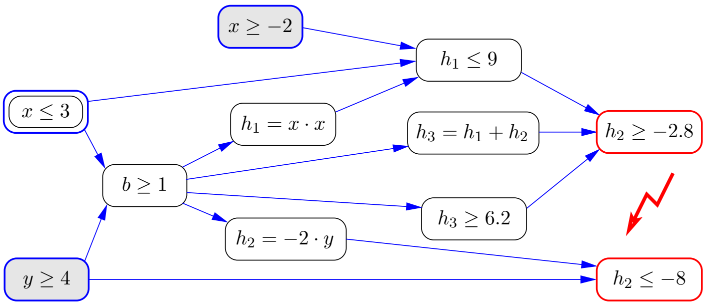

Figure 2. Conflict analysis: Let $(x > 4 \vee y \leq 0 \vee x^2 - 2y \geq 6.2)$ be a fragment of a formula to be solved. Rewriting this fragment into our internal syntax yields the clauses $c_1, \ldots, c_5$ to the left. Assume $x \geq -2$ and $y \geq 4$ have been asserted on decision levels k_1 and k_2, resp., and another decision level is opened by asserting $x \leq 3$. Clause c_1 becomes unit since the atoms $x > 4$ and $y \leq 0$ are inconsistent. By rule 1, we assert $b \geq 1$ due to the atoms $x \leq 3$ and $y \geq 4$. By $b \geq 1$ the clauses $c_2, \ldots, c_5$ become unit, implying the equations $h_1 = x \cdot x$, $h_2 = -2 \cdot y$, $h_3 = h_1 + h_2$, and the bound $h_3 \geq 6.2$. By equation $h_2 = -2 \cdot y$ and $y \geq 4$ we compute the new upper bound $h_2 \leq -8$. The remaining deduction process is indicated by the resulting implication graph on the right, ending in a conflict on h_2 (red boxes). Edges relate implications to their antecedents. Following the implication chains (blue edges) from the conflict yields the bounds causing the conflict (blue boxes). The added conflict clause $\neg(x \geq -2) \vee \neg(x \leq 3) \vee \neg(y \geq 4)$ becomes unit after backjumping to decision level $max(k_1, k_2)$, propagating $x > 3$.

solving: as the antecedents of a contraction may involve arbitrary arithmetic bounds, so do the conflict clauses. Furthermore, in contrast to nogood learning in constraint propagation, we are not confined to learning forbidden combinations of value assignments in the search space, which here would amount to learning disjunctions of interval disequations $x \notin I$ with x being a problem variable and I an interval. Instead, our algorithm may learn arbitrary combinations of atoms $x \sim c$, which provides stronger pruning of the search space: while a nogood $x \notin I$ would only prevent a future visit to any subinterval of I, a bound $x \geq c$, for example, blocks visits to any interval whose left endpoint is at least c, no matter how it is otherwise located relative to the current interval valuation.

4.4 Enforcing progress and termination

The naive base algorithm described above would apply unbounded splitting, thus risking non-termination due to the density of the order on $\mathbb{R}$. It traverses the search tree until either no further splits are possible due to the search space being fully covered by conflict clauses or until a solution of the problem is witnessed (cf. Sect. 4.5). In contrast to purely propositional SAT solving, where the split depth is bounded by the number of variables in the SAT problem, this entails the risk of non-termination due to infinite sequences of splits being possible on each real-valued interval. Even worse, by pursuing depth-first search, the algorithm risks infinite descent into a branch of the search tree even if other branches may yield definite, satisfying results.

We tackle this problem by selecting a heuristics for application of the rules which guarantees a certain progress with respect to decided and deduced bounds. Therefore, we fix a progress bound $\varepsilon > 0$ (to be refined iteratively later on) and demand that the rule applications satisfy the following condition.

Condition 1. *Rules (3), (5), and (7) are only applied if their asserted bound $v \sim c$ narrows the remaining range of v by at least ε, i.e. if $\forall c' \in \mathbb{R} : (v \sim c') \in N \Rightarrow |c - c'| \geq \varepsilon$, where $N = M$ for rules (3) and (5) and $N = M'$ for rule (7).*

Rule (4) is only applied if both the split bound $v \sim c$ and its negation $v \not\sim c$ narrow the remaining range of v by at least ε.

We now define a strict partial ordering $\succ$ on the list of asserted atoms M. For that purpose we denote by $B(M)$ the largest interval box satisfying each bound in M. Let $M = M_1 | \ldots | M_n$ and $M' = M'_1 | \ldots | M'_m$ s.t. $| \notin \left(\bigcup_{i=1}^{n} M_i \cup \bigcup_{j=1}^{m} M'_j \right)$. Then $M \succ M'$ if $\exists k : 1 \leq k \leq n, m : B(M_1) = B(M'_1), \ldots, B(M_{k-1}) = B(M'_{k-1})$, and $B(M'_k)$ is a proper refinement of $B(M_k)$, i.e. $B(M'_k) \subset B(M_k)$.

Lemma 2. *If $(\Sigma, M, \phi) \longrightarrow (\Sigma', M', \phi')$ then the following propositions hold.*

1. *$M \succ M'$ or $B(M) = B(M')$ if rule (1) was applied.*
2. *Rule (2) does not change the list of asserted atoms, i.e. $B(M) = B(M')$.*
3. *$M \succ M'$ if rule (3), (4), (5), or (7) was applied.*

One can easily check that propositions 2 and 3 of lemma 2 hold. For proposition 1 note that if a bound $v \sim c$ where $M \not\models_{\mathrm{hc}} v \sim c$ is added to the list of atoms then $M \succ M'$. Otherwise, $B(M) = B(M')$ holds by adding a bound $v \sim c$ with $M \models_{\mathrm{hc}} v \sim c$ or an equation.

By Lemma 2, we are able to prove termination of our algorithm when started with the initial intervals of both auxiliary and problem variables being bounded (where the bounded intervals may contain the special value $\mho$). Let $\mathbb{I}$ be an interval. If $\mathbb{I}$ is bounded, i.e. $\exists lb, ub \in \mathbb{R} : \inf(\mathbb{I}) = lb \wedge \sup(\mathbb{I}) = ub$, then the width of $\mathbb{I}$ is defined as $width(\mathbb{I}) = ub - lb$, else $width(\mathbb{I}) = \infty$. Note that the bounds may well be the biggest and smallest numbers representable on our computer.

Lemma 3 (Termination). *Let $(\langle \rho_\perp \rangle, M, \phi)$ be the initial state of the algorithm where $| \notin M$ and $\forall v \in BV \cup RV : width(M(v)) \neq \infty$. Then the algorithm reaches a state S after finitely many applications of the rules (1) to (7) subject to condition 1 s.t.*

1. *$S = \mathrm{unsat}$, or*
2. *$S = (\Sigma, M', \phi')$ where $\forall v \in BV \cup RV : width(M'(v)) \leq 2 \cdot \varepsilon$ and ε is the constant progress parameter of condition 1.*

Proof. If the state unsat will eventually be reached the algorithm stops. Therefore assume that state unsat will not be reached. Observe that rules (1) and (2) can be applied only finitely often in consecution. Therefore, each possible infinite execution path of the algorithm applies the rules (3), (4), (5), or (7) infinftely often, yielding an infinitely decreasing (wrt. $\succ$) sequence of $(M_i)_{i\in\mathbb{N}}$ according to Lemma 2. Let $(N_j)_{j\in\mathbb{N}}$ be its infinite subsequence of M_i originating from application of rules (3), (4), (5), or (7). Due to condition 1, the length of this subsequence is bounded by $\mathcal{O}\left(\prod_{v\in RV\cup BV}\frac{width(M(v))}{\varepsilon}\right)<\infty$, yielding a contradiction.

Observe, that the rules 5 and 7 are always applicable: For rule 5, $v\sim c$ was added by rule 4 and, thus, $v\not\sim c$ narrows the remaining range of v by at least ε (condition 1). Concerning rule 7, we are always able to find a bound $a_1\in M''$ s.t. $\neg a_1$ narrows the remaining range by at least ε. (The rightmost bound in M added by rule 4 is always adequate.) The condition for M' then follows from the fact that application of rule 4 failed. □

Thus, given a progress bound $\varepsilon>0$, the algorithm will always terminate. Termination could, however, be enforced by the progress bound before a conclusive result has been found. We compensate for that by performing restarts with refined progress bound $\varepsilon'>0$. Note that such an iterative refinement of the progress bound is considerably different from not using a progress bound, as it still prevents infinite digression into a single branch of the search space, adding some breadth-first flavor.

Achieving almost-completeness through restarts. With a given progress parameter ε of condition 1, the above procedure may terminate with inconclusive result: it may fail to terminate with an "unsat" result, yet undecided branches in the search space — corresponding to inconclusive interval interpretations— remain. In this case, the solver simply is restarted with a smaller progress parameter. As all the conflict clauses are preserved from the previous run, the new run essentially only visits those interval interpretations that were previously left in an inconclusive state, and it extends the proof tree precisely at these inconclusive leaves.

By iterating this scheme for incrementally smaller progress parameter converging to zero, we obtain an "almost complete" procedure being able to refute (and, with the extensions of Sect. 4.5, verify) all *robustly* unsatisfiable (robustly satisfiable, resp.) formulae, where robustness here means that the corresponding property is stable under small perturbation of the constants in the problem.

4.5 Finding satisfying valuations

The notion of hull consistency, while being a necessary condition for real-valued satisfiability, fails to be a sufficient condition for real-valued satisfiability. Current interval based solvers usually take the mid-point of a (e.g., hull consistent) valuation as a starting point for an iterative local search method (e.g., Newton's method). If the search converges, the resulting point contains an approximate solution of the input constraints. For a mathematically correct proof of satisfiability one then usually generates boxes around the approximate solution (cf. ϵ-inflation [May94]), and verifies the existence of a solution to the equality constraints within the box using fixpoint arguments (cf. Newton operator, Miranda's theorem [Neu90]). The inequality constraints can be verified using simple interval arithmetic. For the whole process the original, un-decomposed input constraints are used.

In our field of application, we can actually do much better, as the formulae tend to have a specific structure facilitating generation of witnesses from tight enough interval valuations.

In the formulae generated from hybrid transition systems, equations usually stem from two sources only: either they are introduced by rewriting complex expressions to three-address form, or they stem from translating assignment operations associated to transitions of the hybrid system into constraints. In the first case, the left-hand sides of the equations are auxiliary variables which occur exactly once on a left-hand side of an equation, and those equations form an acyclic graph. In the latter case, multiple left-hand occurrences of the same variable may appear in the constraint system, but these stem from mutually exclusive transitions such that satisfying the overall formula does not require to satisfy the conjunction of two or more equations with the same left-hand side. Under these circumstances, a certain form of interval satisfiability by tight intervals is sufficient for real-valued satisfiability.

We call an interval valuation ρ *strongly satisfying* for ϕ, denoted $\rho \models_{\mathrm{s}} \phi$, iff there is a finite set $A = \{a_1, \ldots, a_n\}$ of atoms such that the following four conditions hold:

1. Each clause in ϕ contains at least one atom $a \in A$.

2. If a_i is an equation $x = y \circ z$ or $x = \circ y$ then x is interpreted by a point interval (i.e., $|\rho(x)| = 1$), or x does neither occur in any a_j with $j > i$ nor on the right-hand side of a_i (i.e., $x \neq y$ and $x \neq z$).

3. $\check{\rho}$ assigns purely real-valued intervals to all variables $v \in RV$, i.e. $\mho \notin \check{\rho}(v)$, where $\check{\rho}$ is the smallest interval valuation satisfying

   (a) $\forall v \in BV \cup RV : \rho(v) \subseteq \check{\rho}(v)$,

   (b) for each equation $(x = y \circ z) \in A$ or $(x = \circ y) \in A$ with $|\rho(x)| > 1$, the inclusion $\check{\rho}(x) \supseteq \check{\rho}(y) \hat{\bullet}_1 \check{\rho}(z)$ or $\check{\rho}(x) \supseteq \hat{\bullet}_1 \check{\rho}(y)$, resp., holds.[6.]

4. All atoms $a \in A$ are interval satisfied by $\check{\rho}$ in the following sense:

$$\begin{array}{lll} a = (x \sim c) & \text{implies} & \check{\rho}(x) \subseteq \{u \mid u \in \mathbb{R}, u \sim c\}, \\ a = (x = y \circ z) & \text{implies} & \check{\rho}(x) \supseteq \check{\rho}(y) \hat{\bullet}_1 \check{\rho}(z), \\ a = (x = \circ y) & \text{implies} & \check{\rho}(x) \supseteq \hat{\bullet}_1 \check{\rho}(y). \end{array}$$

Due to the ordering condition on equality constraints that are satisfied by non-point intervals, we obtain the following tight correspondence between strong interval satisfiability and real-valued satisfiability:

Lemma 4. *If $\rho \models_{\mathrm{s}} \phi$ then there exists a real-valued valuation σ such that $\sigma \models \phi$.*

Proof. If $\rho \models_{\mathrm{s}} \phi$ then we can recursively define a real-valued valuation σ exploiting the structure of the family of witnesses $a_i \in A$ as follows:

1. For each $v \in BV$ and for each $v \in RV$ not occurring on the left-hand side of any equation in $\{a_1, \ldots, a_n\}$, select $\sigma(v) \in \check{\rho}(v)$ arbitrarily;

[6.] Note that the interval valuation $\check{\rho}$ is well-defined due to condition (2). Furthermore, $\check{\rho}$ is easily computable by a recursive procedure that sets $\check{\rho}(x) = \rho(x)$ for all x with $|\rho(x)| = 1$ or which do not occur on the left hand side of some a_i. The remaining values are then computed by solving the equations $\check{\rho}(x) = \check{\rho}(y) \hat{\bullet}_1 \check{\rho}(z)$ or $\check{\rho}(x) = \hat{\bullet}_1 \check{\rho}(y)$, respectively.

2. For $i = n$ down to 1, process the constraints a_n to a_1 in reverse sequence as follows: if a_i is an equation $v = x \circ y$ or $v = \circ x$ then take $\sigma(v) = \sigma(x) \circ \sigma(y)$ or $\sigma(v) = \circ\sigma(x)$, respectively.

Note that solutions to the equation system in (2.) exist because the hierarchical order of variable dependencies in a_1 to a_n enforces that each $\sigma(v)$ either is subject to at most one defining equation or is picked from a point interval $\check{\rho}(v) \supseteq \check{\rho}(x)\hat{\bullet}_1\check{\rho}(y)$ or $\check{\rho}(v) \supseteq \hat{\bullet}_1\check{\rho}(y)$, respectively. Furthermore, $\sigma(v) \neq \mho$ as $\mho \notin \check{\rho}(v) \supseteq \check{\rho}(x)\hat{\bullet}_1\check{\rho}(y) \ni \sigma(x) \circ \sigma(y)$ and $\mho \notin \check{\rho}(v) \supseteq \hat{\bullet}_1\check{\rho}(y) \ni \circ\sigma(y)$, respectively. It is straightforward to check that $\sigma \models \phi$. □

Example. Let

$$\begin{array}{rcl}\phi & = & (x > 0 \vee y \geq 0 \vee c < 0) \wedge (x > 4 \vee y \leq -3 \vee b) \wedge (\neg b \vee x = y + z) \wedge \\ & & (\neg b \vee y = \sin(c)) \wedge (\neg b \vee c = 2 \cdot z)\end{array}$$

be a formula, x, y, z, c be real-valued variables and b a Boolean variable. Let ρ be the current interval valuation, where $\rho(x) = (1.54, 1.65), \rho(y) = (-0.06, 0.05), \rho(z) = [1.55, 1.6], \rho(c) = [3.1, 3.2], \rho(b) = [1, 1](= \{\mathtt{true}\})$.
Let $A = \{a_1 = (x > 0), a_2 = (b), a_3 = (x = y + z), a_4 = (y = \sin(c)), a_5 = (c = 2 \cdot z)\}$ be a set of atoms. Obviously, conditions 1 and 2 are satisfied by A for ϕ. Condition 3 leads to $\check{\rho}$, where

$$\begin{array}{rclcl}\check{\rho}(z) & = & \rho(z) & = & [1.55, 1.6], \\ \check{\rho}(c) & = & 2 \mathbin{\hat{\cdot}} \check{\rho}(z) & = & [3.1, 3.2], \\ \check{\rho}(y) & = & \widehat{\sin}(\check{\rho}(c)) & = & (-0.06, 0.05), \\ \check{\rho}(x) & = & \check{\rho}(y) \mathbin{\hat{+}} \check{\rho}(z) & = & (1.49, 1.65), \\ \check{\rho}(b) & = & \rho(b) & = & [1, 1].\end{array}$$

By an easy check, condition 4 can be verified. Thus, ρ strongly satisfies ϕ which implies that there is a real-valued solution of ϕ, e.g. $z = 1.6$, $c = 3.2$, $y = \sin(3.2)$, $x = \sin(3.2) + 1.6$, and $b = \mathtt{true}$. □

This motivates extending the base algorithm from Sect. 4 with the following additional rule for termination:

$$\frac{\rho \models_{\mathrm{s}} \phi}{(\Sigma \cdot \langle\rho\rangle, M, \phi) \longrightarrow \mathrm{sat}} \tag{8}$$

Note that $\rho \models_{\mathrm{s}} \phi$ is decidable. For the sake of efficiency, our implementation checks applicability of rule (8) only if all clauses are unit (in the sense of containing at most one atom that is not yet inconsistent) and $\mathit{width}(\rho(v)) \leq 2 \cdot \varepsilon$ for each $v \in \mathit{RV}$, where ε is the the progress parameter of condition 1.

The correctness of rule (8) wrt. real-valued satisfiability follows directly from Lemma 4, providing the following correctness property for the extended algorithm.

Corollary 2. *If* $(\langle\rho_\bot\rangle, \varepsilon, \phi) \longrightarrow^* \mathrm{sat}$ *then* ϕ *is satisfiable over* $\mathbb{R}$. □

We conjecture that our algorithm enhanced by rule (8) will eventually, after sufficiently refining the progress parameter ε of condition 1, find a satisfying solution to each satisfiable formula of the aforementioned syntactic structure arising in our domain, provided the formula ϕ has at least one *non-isolated* model. Here, a model σ of ϕ is non-isolated iff it satisfies a set $\{a_1, \ldots, a_n\} = A$ of atoms such that each clause in ϕ contains at least one a_i and for the vector $(x_1, \ldots, x_m)$ of variables not occurring on the left-hand side of an equation $e \in A$ there exists a neighborhood $U_{\vec{x}}$ of $(\sigma(x_1), \ldots, \sigma(x_m))$ such that there is a model σ' with $\sigma'(x_i) = u_i$ for each $\vec{u} \in U_{\vec{x}}$. Confer [Rat06, Rat02] for a more complete discussion of such robustness issues.

5. Benchmark results

In this section we provide experimental results obtained from benchmarking our tool iSAT.[7] The benchmarks mentioned in sections 5.1 and 5.2 were performed on a 2.5 GHz AMD Opteron machine with 4 GByte physical memory while those of Sect. 5.3 were pursued on a 1.83 GHz Intel Core 2 Duo machine with 1 GByte physical memory, both running Linux.

5.1 Impact of conflict-driven learning

In order to demonstrate the potential of our approach, in particular the benefit of conflict-driven learning adapted to interval constraint solving, we compare the performance of iSAT to a stripped version thereof, where learning and backjumping are disabled (but the optimized data structures, in particular watched atoms, remain functional).

We considered bounded model checking problems, that is, proving a property of a hybrid discrete-continuous transition system for a fixed unwinding depth k. Without learning, the interval constraint solving system failed on every moderately interesting hybrid system due to complexity problems exhausting memory and runtime. This could be expected because the *expected* number of boxes to be visited grows exponentially in the number of variables in the constraint formula, which in turn grows linearly in both the number of problem variables in the hybrid system and in the unwinding depth k. When checking a model of an *elastic approach to train distance control* [FH06], the version without learning exhausts the runtime limit of 3 days already on unwinding depth 1, where formula size is 140 variables and 30 constraints. In contrast, the version with conflict-driven learning solves all instances up to depth 10 in less than 3 minutes, thereby handling instances with more than 1100 variables, a corresponding number of triplets and pairs, and 250 inequality constraints. For simpler hybrid systems, like the model of a *bouncing ball* falling in a gravity field and subject to non-ideal bouncing on the surface, the learning-free solver works due to the deterministic nature of the system. Nevertheless, it fails for unwinding depths > 11, essentially due to enormous numbers of conflicting assignments being constructed (e.g., $> 348 \cdot 10^6$ conflicts for $k = 10$), whereas learning prevents visits to most of these assignments (only 68 conflicts remain for $k = 10$ when conflict-driven learning is pursued). Consequently, the learning-enhanced solver traverses these problems in fractions of a second; it is only from depth 40 that our solver needs more than one minute to solve the bouncing ball problem (2400 variables, 500 constraints). Similar effects were observed on chaotic real-valued maps, like the *gingerbread map*. Without conflict-driven learning, the solver ran into approx. $43 \cdot 10^6$, $291 \cdot 10^6$, and $482 \cdot 10^6$ conflicts for $k = 9$ to 11, whereas only 253, 178, and 155 conflicts were encountered in the conflict-driven approach, respectively. This clearly demonstrates that

[7] The benchmarks and an iSAT executable can be found on `http://hysat.informatik.uni-oldenburg.de`

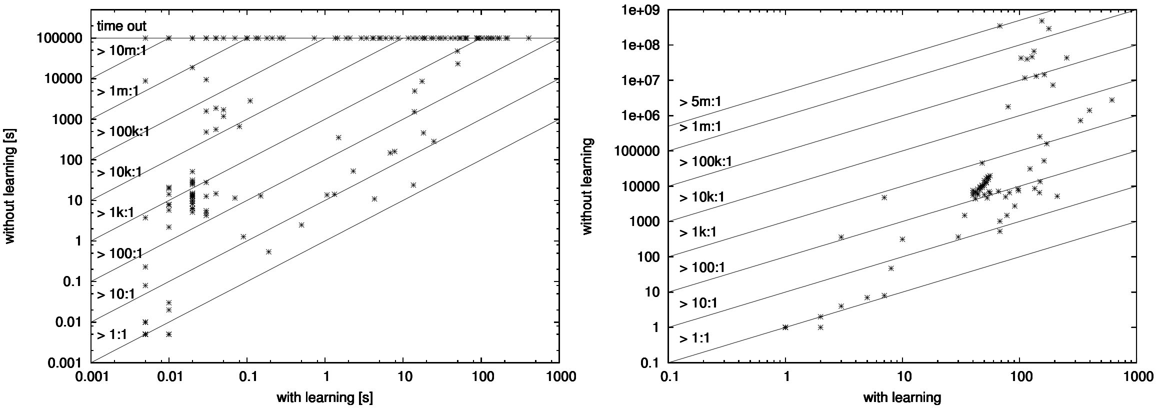

Figure 3. Performance impact of conflict-driven learning and non-chronological backtracking: runtime in seconds (left) and number of conflicts encountered (right)

conflict-driven learning is effective within interval constraint solving: it dramatically prunes the search space, as witnessed by the drastic reduction in conflict situations encountered and by the frequency of backjumps of non-trivial depth, where depths of 47 and 55 decision levels were observed on the gingerbread and bouncing ball model, respectively. Similar effects were observed on two further groups of benchmark examples: an oscillatory *logistic map* and some geometric decision problems dealing with the intersection of n-dimensional geometric objects. On random formulae, we even obtained backjump distances of more than 70000 levels. The results of the aforementioned benchmarks, excluding the random formulae, are presented in Fig. 3.

5.2 Phase transition

As our algorithm performs a backtrack search, we also investigated the phase transition behavior on scalable problems featuring a satisfiability threshold (cf. Fig. 4). Within these experiments, we have unwound bounded model-checking problems over successively larger depths such that the problem undergoes a phase transition from being unsatisfiable to satisfiability, forcing our algorithm to adapt its search strategy from refutation to construction of satisfying assignments. The strong satisfaction test (rule (8)) in our implementation tries to construct a (point) solution as mentiond in the proof of lemma 4 and, if successfull, returns this solution.

The models where three small to medium-sized hybrid systems (two bouncing balls and a high-lift system in avionics), featuring a relatively low amount of nondeterminism in the transition selection, as well as the chaotic Duffing map extended by some target region to be reached through iterated application of the deterministic map. While we had expected an extremely high computational cost of actually finding satisfying valuations due to deep digression into the search lattice, at least for these relatively deterministic systems, the computational cost of actually finding satisfying valuations over the reals turned out to be comparable to that of refuting the unsatisfiable cases. The sizes here were up to 18119 variables, encountered on the satisfiable unwinding of depth 78 of the high-lift model. The

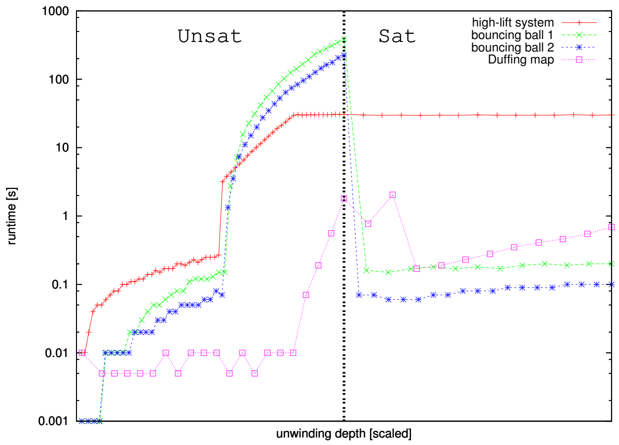

Figure 4. Transition between refutation algorithm and construction of satisfying valuations: runtime of bounded model-checking problems when unwound across the satisfiability threshold

other models were considerably smaller, with the Duffing map being by far the smallest with up to 714 variables.

5.3 Comparison to ABsolver

Finally, we provide a comparison to ABsolver [BPT07], which, to the best of our knowledge, is the only other SMT-based solver addressing the domain of large Boolean combinations of non-linear arithmetic constraints over the reals. The currently reported implementation [BPT07] uses the numerical optimization tool IPOPT [WB06] for solving the non-linear constraints. IPOPT is a highly efficient tool for numerical local optimization. However, in contrast to global optimization methods, it only searches for local solutions, and hence may incorrectly claim a satisfiable set of constraints inconsistent. Moreover, IPOPT may also produce incorrect results due to rounding errors. Note that solving non-linear constraints globally and without rounding errors is considered a problem that is harder to solve by orders of magnitude.

The current implementation of ABsolver[8.] supports the arithmetic operations of addition (and subtraction) and multiplication (and division) only. Therefore, the respective benchmarks are restricted to polynomial arithmetic. We performed the experiments on ABsolver with options `sat:zchafflib`, `l:coin`, `nl:ipopt`. Table 2 lists the experimental results. Its second column states the unwinding depth of bounded model checking problems, the third column the number of arithmetic operators in the benchmark.

The first benchmark is described in [JL98, p. 5]. Benchmarks 2 to 4 can be found in [BPT07] as well as on ABsolver's web page. The industrial case study of a mixed-signal circuit in a car-steering control system (benchmark 5) is described in [BPT07], yet is not publicly available due to protection of intellectual property. The remaining benchmarks are bounded model checking problems of hybrid systems and of iterated chaotic maps. Except for the car-steering control system, all benchmarks are available from the iSAT web site.

[8.] Available from http://absolver.sourceforge.net/

Benchmark	BMC depth	♯arith_op	iSAT	ABSOLVER	speedup
nonlinear_CSP	—	44	0m0.032s	0m0.072s	2.2
esat_n11_m8_nonlinear	—	7	0m0.028s	0m0.012s	0.4
nonlinear_unsat	—	2	0m0.004s	0m0.032s	8.0
div_operator	—	1	0m0.004s	0m0.028s	7.0
car_steering	—	138	0m0.268s	2m11.032s	488.9
h3_train	2	102	0m0.244s	0m2.968s	12.2
h3_train	3	153	0m0.304s	0m6.480s	21.3
h3_train	4	204	0m0.344s	0m10.401s	30.2
h3_train	5	255	0m0.348s	0m15.981s	45.9
h3_train	17	867	0m30.718s	3m22.769s	6.6
h3_train	18	918	0m33.346s	3m39.374s	6.6
h3_train	30	1530	0m46.519s	10m55.965s	14.1
renault_clio	2	132	0m0.020s	0m0.764s	38.2
renault_clio	3	198	0m0.024s	0m1.628s	67.8
renault_clio	30	1980	0m0.300s	3m48.158s	760.5
renault_clio	31	2046	0m0.344s	4m9.528s	725.4
aircraft	5	132	0m0.044s	2m30.113s	3,411.7
aircraft	6	157	0m0.056s	5m47.182s	6,199.7
aircraft	10	257	0m1.496s	50m43.594s	2,034.5
duffing_map	3	21	0m0.004s	0m4.904s	1,226.0
duffing_map	4	28	0m0.004s	1m58.571s	29,642.7
duffing_map	5	35	0m0.004s	3m35.117s	53,779.2
duffing_map	6	42	0m0.004s	5m52.822s	88,205.5
duffing_map	7	49	0m0.001s	0m22.313s	22,313.0
duffing_map	8	56	0m0.004s	1m16.849s	19,212.2
duffing_map	20	140	0m0.008s	6m15.675s	46,959.4
duffing_map	30	210	0m0.012s	> 60m	> 300,000
tinkerbell_map	1	22	0m0.004s	0m1.292s	323.0
tinkerbell_map	2	38	0m0.008s	3m12.052s	24,006.5
tinkerbell_map	3	54	0m0.048s	7m58.210s	9,962.7
tinkerbell_map	4	70	0m0.676s	15m49.495s	1,404.6
tinkerbell_map	5	86	0m1.080s	27m37.548s	1,534.8
tinkerbell_map	6	102	0m0.644s	46m42.739s	4,352.1
nonlinear_ball	2	52	0m0.008s	9m8.538s	68,567.2
nonlinear_ball	3	78	0m0.008s	> 60m	> 450,000
nonlinear_ball	4	104	0m0.012s	> 60m	> 300,000

Table 2. Performance of iSAT relative to ABSOLVER.

For the first 4 benchmarks, which are very small numeric CSPs without complex Boolean structure, the runtimes are almost equal. For all other benchmarks, iSAT yields orders of magnitude of speedup compared to ABSOLVER, no matter whether the benchmarks feature moderately complex Boolean structure, like the mixed-signal circuit in car-steering, feature extremely complex Boolean structure, as in bounded model-checking of the linear hybrid automata h3_train, renault_clio, and aircraft and the non-linear bouncing ball, or are almost conjunctive (basically, one disjunction per unwinding step), like the iterated duffing and tinkerbell maps. The comparable performance on purely conjunctive problems, which is indicative of the relative performance of the underlying arithmetic reasoning engines, together with the huge performance gap on problems with more complex Boolean structure shows that the tight integration of Boolean and arithmetic constraint propagation pursued in iSAT saves overhead incurred in an SMT approach deferring theory problems to subordinate theory solvers.

6. Discussion

Within this article, we have demonstrated how a tight integration of DPLL-style SAT solving and interval constraint propagation can reconcile the strengths of the SAT-modulo-theory (SMT) approach with those of interval constraint propagation (ICP). The particular strength of SMT in manipulating large and complex-structured Boolean combinations of constraints over a (in general decidable) theory is thus lifted to the undecidable domain of non-linear arithmetic involving transcendental functions. In particular, we were thus able to canonically lift to interval-based arithmetic constraint solving of massively disjunctive constraint problems the crucial algorithmic enhancements of modern propositional SAT solvers, especially lazy clause evaluation, conflict-driven learning, and non-chronological backtracking. Our benchmarks demonstrate significant performance gains up to multiple orders of magnitude compared to a pure backtrack SMT+ICP algorithm. Equally important, the performance gains were consistent throughout our set of benchmarks, with only one trivial instance incurring a negligible performance penalty due to the more complex algorithms. Similar results were observed in comparison with a non-linear SMT solver (ABsolver) employing classical deferring of theory problems to subordinate solvers, which substantiates the argument that tighter integration of DPLL and ICP is beneficial.

Plans for future extensions deal with three major topics: first, we will extend the base engine with specific optimizations for bounded model checking of hybrid systems, akin to the optimizations discussed in [FH06] for the case of linear hybrid automata. Second, we will use linear programming on the linear subset of the asserted atoms, that is, on bounds and linear equations, to obtain stronger forward and backward inferences, including additional size reduction of conflicts to be learned. This would lower the overhead when reasoning over timed and (partially) linear hybrid automata, where polyhedral sets provide a more concise description of state sets than the rectangular regions provided by intervals. Finally, we are currently implementing native support for ordinary differential equations via ICP-based reasoning over safe numerical approximations of the solution in the interval domain, as pursued in [Sta97].

References

[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*, Foundations of Artificial Intelligence, Elsevier, Amsterdam, 2006, pp. 571–603.

[BMVH94] F. Benhamou, D. McAllester, and Pascal Van Hentenryck, *CLP(Intervals) revisited*, Int. Symp. on Logic Progr. (Ithaca, NY, USA), MIT Press, 1994, pp. 124–138.

[BNOT06] C. Barrett, R. Nieuwenhuis, A. Oliveras, and C. Tinelli, *Splitting on Demand in SAT Modulo Theories*, 13h International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR'06 (M. Hermann and A. Voronkov, eds.), LNCS, vol. 4246, Springer, 2006, pp. 512–526.

[BPT07] A. Bauer, M. Pister, and M. Tautschnig, *Tool-support for the analysis of hybrid systems and models*, Proceedings of the 2007 Conference on Design, Automation and Test in Europe (DATE'07) (Los Alamitos, CA), IEEE Computer Society, April 2007.

[Cle87] J. G. Cleary, *Logical arithmetic*, Future Computing Systems **2** (1987), no. 2, 125–149.

[Dav87] E. Davis, *Constraint propagation with interval labels*, Artif. Intell. **32** (1987), no. 3, 281–331.

[DLL62] M. Davis, G. Logemann, and D. Loveland, *A machine program for theorem proving*, Communications of the ACM **5** (1962), 394–397.

[dMOR$^+$04] L. de Moura, S. Owre, H. Ruess, J. Rushby, and N. Shankar, *The ICS decision procedures for embedded deduction*, 2nd International Joint Conference on Automated Reasoning (IJCAR) (Cork, Ireland), LNCS, vol. 3097, Springer-Verlag, July 2004, pp. 218–222.

[ES03] N. Eén and N. Sörensson, *An extensible SAT-solver*, Proceedings of SAT'03, 2003, pp. 502–518.

[FH06] M. Fränzle and C. Herde, *Efficient proof engines for bounded model checking of hybrid systems*, Formal Methods in System Design (2006).

[Gan02] H. Ganzinger, *Shostak light*, Automated Deduction – CADE-18 (A. Voronkov, ed.), LNCS, vol. 2392, Springer-Verlag, 2002, pp. 332–346.

[Gin93] M.L. Ginsberg, *Dynamic backtracking*, Journal of Artificial Intelligence Research **1** (1993), 25–46.

[GN02] E. Goldberg and Y. Novikov, *BerkMin: A Fast and Robust SAT-Solver*, Design, Automation, and Test in Europe, 2002.

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

[JL98] N. Jussien and O. Lhomme, *Dynamic domain splitting for numeric CSPs*, European Conference on Artificial Intelligence, 1998, pp. 224–228.

[KB03] G. Katsirelos and F. Bacchus, *Unrestricted nogood recording in CSP search*, Principles and Practice of Constraint Programming — CP 2003 (Francesca Rossi, ed.), LNCS, vol. 2833, Springer-Verlag, 2003, pp. 873–877.

[LSB07] M. Lewis, T. Schubert, and B. Becker, *Multithreaded SAT Solving*, 12th Asia and South Pacific Design Automation Conference, 2007.

[May94] G. Mayer, *Epsilon-inflation in verification algorithms*, Journal of Computational and Applied Mathematics **60** (1994), 147–169.

[MMZ$^+$01] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik, *Chaff: Engineering an Efficient SAT Solver*, Proc. of the 38th Design Automation Conference (DAC'01), June 2001.

[Moo66] R.E. Moore, *Interval analysis*, Prentice Hall, NJ, 1966.

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

[Rat02] Stefan Ratschan, *Quantified constraints under perturbations*, Journal of Symbolic Computation **33** (2002), no. 4, 493–505.

[Rat06] S. Ratschan, *Efficient solving of quantified inequality constraints over the real numbers*, ACM Transactions on Computational Logic **7** (2006), no. 4, 723–748.

[Sta97] O. Stauning, *Automatic validation of numerical solutions*, Ph.D. thesis, Danmarks Tekniske Universitet, Kgs. Lyngby, Denmark, 1997.

[WB06] A. Wächter and L. T. Biegler, *On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming*, Mathematical Programming (2006), 25–57.

[ZMMM01] L. Zhang, C.F. Madigan, M.W. Moskewicz, and S. Malik, *Efficient Conflict Driven Learning in a Boolean Satisfiability Solver*, IEEE/ACM International Conference on Computer-Aided Design, 2001.