AVACS – Automatic Verification and Analysis of Complex Systems

REPORTS

of SFB/TR 14 AVACS

Editors: Board of SFB/TR 14 AVACS

Conflict Analysis and Restarts in a mixed Boolean and Non-linear Arithmetic Constraint Solver

by

Tino Teige Christian Herde Martin Fränzle

Erika Ábrahám

AVACS Technical Report No. 34
January 2008
ISSN: 1860-9821

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

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

Conflict Analysis and Restarts in a mixed Boolean and Non-linear Arithmetic Constraint Solver

Tino Teige[1], Christian Herde[1], Martin Fränzle[1], and Erika Ábrahám[2]

[1] University of Oldenburg, Germany
{teige|herde|fraenzle}@informatik.uni-oldenburg.de
[2] Forschungszentrum Jülich, Germany
e.abraham@fz-juelich.de

Abstract. The recently presented constraint solver HySAT tackles the in general undecidable problem of solving mixed Boolean and non-linear arithmetic constraint formulae over the reals involving transcendental functions. The algorithmic core of HySAT is the iSAT algorithm, a tight integration of the Davis-Putnam-Logemann-Loveland algorithm with interval constraint propagation enriched by enhancements like conflict-driven clause learning and non-chronological backtracking. However, it was an open question whether HySAT's conflict analysis scheme, an adapted first unique implication point technique as applied in most modern SAT solvers, performs favorably compared to other schemes. In this paper, we compare several conflict analysis heuristics to answer this question. Furthermore, we consider the integration of restarts into the iSAT algorithm and investigate their impact. For our empirical results we use benchmarks from the hybrid systems domain.

1 Introduction

Our tool HySAT[3] is a satisfiability checker for arbitrary Boolean combinations of non-linear arithmetic constraints (involving transcendental functions like sin or cos) over real- and integer-valued variables. It has been specifically designed for bounded model checking of hybrid (discrete-continuous) systems. The algorithmic core of HySAT is the iSAT algorithm [FHR$^+$07], a tight integration of the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [DP60,DLL62] with interval constraint propagation (ICP, cf. [BG06] for an extensive survey).

By the front-end of our constraint solver HySAT, the input constraint formulae are rewritten to linearly sized, equi-satisfiable formulae in conjunctive normal form (CNF), with atomic propositions and arithmetic constraints confined to a form resembling three-address code. This rewriting is based on the standard mechanism of introducing auxiliary variables for the values of arithmetic sub-expressions and of logical sub-formulae. Thus, a formula of the internal syntax is a *conjunction of clauses*, where each clause is a *disjunction of atoms*. In the iSAT setting atomic formulae have a more complex structure than in the propositional case. An *atom* is either a *bound* (one variable, one relational operator, and a rational constant), e.g. $x < 5.71$, or an *arithmetic predicate* of at most 3 variables, one relational, and one arithmetic operator, e.g. $x > y \cdot z$ or $x = \sin(y)$ (cf. [FHR$^+$07][4]). The definition of satisfaction is standard: a constraint formula φ is satisfied by a valuation of its

[3] A HySAT executable, the tool documentation, and benchmarks can be found on http://hysat.informatik.uni-oldenburg.de.

[4] Note that the definition of an arithmetic predicate is slightly extended here compared to [FHR$^+$07]. I.e., we are not limited to *equations* like, e.g., $x = y \cdot z$, but we also allow *inequalities* like, e.g., $x > y \cdot z$. Then, interval deduction via ICP and evaluation of the truth value can be smoothly adapted.

variables iff all its clauses are satisfied, i.e., 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. Instead of real-valued valuations of variables, the iSAT algorithm manipulates interval-valued valuations. Due to this fact a possible result of our solver is `unknown`, in addition to `satisfiable` and `unsatisfiable`. In the `unknown`-case, iSAT also provides a "small" interval valuation ρ of the variables x, i.e. each interval $\rho(x)$ does not exceed a (user-)specified size. Moreover, ρ is *hull consistent* in an adapted way (cf. [FHR$^+$07]).

In contrast to most of the satisfiability modulo theories (SMT) approaches (cf. [NOT06] for a nice survey), iSAT does not employ a subordinate theory solver but all decision, reasoning, and learning steps are done in one core. We showed that our novel approach outperforms by orders of magnitude the classical lazy theorem approach of ABsolver [BPT07], which —to the best of our knowledge— is the only other tool addressing mixed Boolean and non-linear arithmetic satisfiability problems. For a very detailed description of the iSAT algorithm the reader is referred to the original paper, in particular to the example on pages 217–219.

In [FHR$^+$07], we introduced *conflict-driven clause learning* and *non-chronological backtracking* within the framework of solving mixed Boolean and non-linear arithmetic constraint problems, which are undecidable in general. Empirical results witness that learning leads to enormous performance gains of multiple orders of magnitude. Due to its proven success in propositional SAT solving we implemented a generalization of the first unique implication point technique (1UIP) from [ZMMM01]. However, in our more general case it is not clear which clause learning heuristics performs best. Like in propositional DPLL engines, the *propagation* process in HySAT takes a considerable amount of the overall runtime (about 90%). Since we reason about arithmetic expressions by means of (safe) interval computations, the propagation steps are more complex and more expensive than in propositional SAT. One way to speed up the propagation phase itself is the use of more efficient data-structures, e.g. a generalized two-watch scheme (cf. [THF$^+$07]).

However, an open and so far not investigated question was whether other conflict analysis schemes which produce shorter conflict clauses are able to significantly decrease the number of propagations and therefore perform better than faster learning schemes like 1UIP. For the Boolean setting it was shown in [ZMMM01] that contrary to common belief, such learning schemes do not perform as well as 1UIP. Thus, one aim of the paper is to shed light on this question. Moreover, we provide a deeper and more detailed insight into the conflict analysis underlying the iSAT algorithm than in [FHR$^+$07], which we esteem as a meaningful contribution.

Furthermore, this paper considers the integration of *restarts* into the iSAT algorithm. As iSAT works very similar to common DPLL SAT solvers (cf. Fig. 2), the algorithmic extension in order to support restarts is technically not challenging. A more interesting issue is whether restarting is still worthwhile in the more general, mixed Boolean and arithmetic, context. The answer to this question is not that obvious: On the one hand, frequent restarts could slow down the solving process since a lot of deductions have to be recomputed when the solver revisits a former branch (or a similar one) of the search tree after restarting. Deductions in iSAT are –besides Boolean propagations– mainly arithmetic (interval) computations and therefore more expensive. On the other hand, restarting could have a positive impact on the efficiency as in the propositional case. Finally, we investigate the *interplay* of restarts and the different conflict analysis heuristics. It is sometimes the case that heuristics perform well if they are employed independently from each other, while their combination does not. We identify which conflict learning scheme is the most stable one upon our benchmarks when the restart mechanism is enabled.

Structure of the paper. Section 2 gives a short overview of related work from different domains, such as SAT, SMT, and CP. Conflict analysis schemes and restarts in iSAT are considered in Section 3, while the empirical evaluation of these heuristics is presented in Section 4. Section 5 concludes the paper and lists some directions for future work.

2 Related work

Conflict-driven clause learning (or clause learning for short) combined with *non-chronological backtracking* plays a crucial role for the efficiency and stability of state-of-the-art DPLL SAT solvers. The idea of conflict-driven learning is to store knowledge about parts of the search space which contain no solution due to a subset of the variable assignments already violating constraints. The knowledge of such "forbidden" areas of the search space is encoded by so-called *conflict clauses*, which prevent the solver from revisiting the same or similar assignments. Another powerful mechanism within the context of clause learning is non-chronological backtracking. Here, the solver is able to backtrack higher in the search tree than in naive backtracking which has proven to be more efficient. One very important heuristics for clause learning is the *unique implication point* (UIP) technique introduced in [MSS96]. In their landmark paper the authors of [ZMMM01] showed that a refined version of the UIP technique, called the *first* UIP (1UIP) technique, outperforms other clause learning heuristics. For more details on different approaches to conflict-driven learning in SAT, e.g. by resolution, the reader is referred to [BHZ06, Section 7].

Learning approaches are also proposed in the constraint programming (CP) community, there often called *nogood* recording. However, in the CP context there is no unique internal representation of the problems, like, e.g., the clausal representation in SAT, and therefore no widely used learning scheme. This originates in the different point of view in the CP world where high-level glass-box tools are provided. Nevertheless, there are some efforts tackling this issue, like *conflict directed backjumping* [Pro93], *dynamic backtracking* [Gin93] and some recent approaches [KB05,RCOJ06,LSTV07].

Conflict-driven clause learning is successfully employed in the SMT community, where subordinate decision procedures are used for reasoning about the underlying theory. To provide learning, the subordinate procedures should be able to return explanations for each theory propagation (e.g., cf. [BNOT06,NOT06] for details).

In the context of non-linear constraint solving, Jussien's and Lhomme's *dynamic domain splitting* technique for numeric constraint satisfaction problems (numeric CSPs) [JL98] (as an extension of dynamic backtracking) implements conflict-driven learning and non-chronological backtracking within arithmetic constraint solving based on domain splitting and arithmetic constraint propagation. However, the algorithm described in [JL98] is not general enough for our problem domain, as it focuses on *conjunctive* constraint systems. 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 conflict clauses entailing both splitting constraints and arbitrary deduced constraints. This is an important property wrt. efficient clause learning as we will see by our benchmark results in Section 4.

The idea of *restarts* in a SAT solver is to erase all decisions and all deductions (after the first decision) done so far while keeping (most of) the learned conflict clauses. Then, the solving process is initialized again. The hope is that the additional information about the search space (given by the conflict clauses) combined with other decisions visiting new parts of the search space decrease the runtime of the solver.

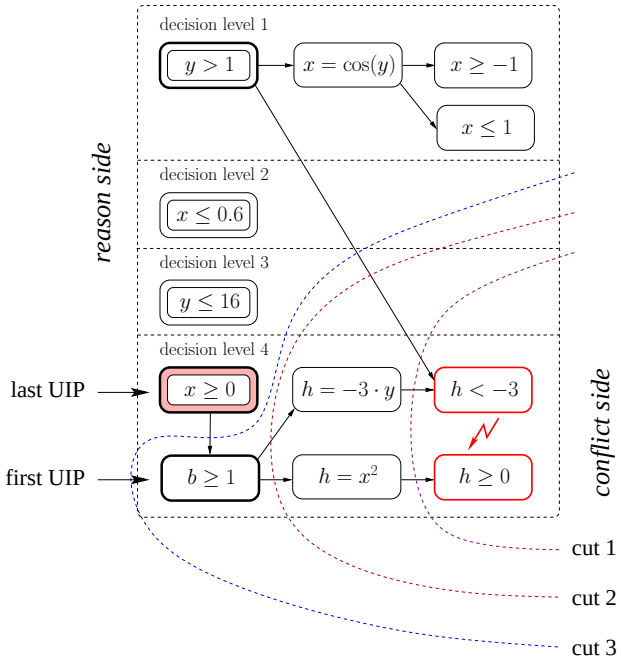

Fig. 1. Implication graph in the iSAT framework

Restart policies are crucial for the success of this heuristics and were extensively studied in the literature. We just mention some of them: fixed-interval policies, e.g. BerkMin [GN02], geometric policies, e.g. MiniSat [ES03] and MiraXT [LSB07], and policies based on a defined sequence as in [LSZ93]. In [Hua07] some important restart policies are compared wrt. their performance. Recent work in the context of CSPs improves the robustness of a CSP solver within the combined nogood recording and restart framework [LSTV07]. More recently, the authors of [WvB07] analytically and empirically investigated the pitfalls of non-universal restart strategies and contributed to the theoretical understanding of universal restart strategies. One focus of their paper is on learning good universal strategies.

3 Conflict analysis and restarts in HySAT

Conflict analysis. The conflict analysis in the iSAT algorithm is best studied by an example. Given the formula

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

of the internal CNF syntax where $x, y, h \in \mathbb{R}$ are real-valued variables and $b \in \mathbb{B} = [0, 1]$ is a Boolean variable. Note that b and $\neg b$ are just abbreviations for the numeric constraints $b \geq 1$ and $b \leq 0$, respectively. The initial interval valuation ρ is defined by $\rho(x) = [-10, 30]$, $\rho(y) = (-8, 25]$, $\rho(h) = \mathbb{R}$, and $\rho(b) = [0, 1]$.

As in modern DPLL SAT solvers, we execute *branching* and *propagation* steps when solving a formula. Branching is done by interval splitting and propagation by evaluating clauses and deducing new intervals. A clause c is called *unit* iff all but one atoms in c are *inconsistent* under the current interval valuation. An atom a is inconsistent under an interval valuation σ iff there are *no* values in the intervals $\sigma(v)$ of the variables v in a s.t. the atom a is satisfied by these values. Otherwise, a is called *consistent*. (Note that a consistent atom a could also be definitely *true*, i.e. all values in the intervals of the corresponding variables satisfy a. However, we

neglect this fact here since it cannot be expected in general that an equation like $x = y \cdot z$ is true under a non-point interval valuation. Reaching point intervals is not guaranteed by pure branching and interval propagation when real-valued variables are involved.) Whenever a clause c becomes *unit*, the remaining consistent atom a in c has to hold for satisfying the formula. Thus, we can propagate a. In case a is an arithmetic equation, we are potentially able to deduce new intervals for the variables in a by interval arithmetic.

Under ρ no clause of φ is unit. To start reasoning, we branch the search space by splitting the interval of y and decide $y > 1$, thus opening decision level 1. Then, the atom $y = \sin(x)$ becomes inconsistent since there are definitely no values for $y \in (1, 25]$ and $x \in [-10, 30]$ which make the atom true. I.e., the first clause becomes unit and the remaining atom $x = \cos(y)$ is asserted. From $x = \cos(y)$ we conclude $x \geq -1$ and $x \leq 1$. Further splits $x \leq 0.6$ and $y \leq 16$ opening decision levels 2 and 3, resp., do not cause any deductions. After deciding $x \geq 0$ (opening decision level 4), the atom $x < 0$ in the second clause becomes inconsistent. 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: The atoms $y > 1$ and $h = -3 \cdot y$ together imply $h < -3$, while $h = x^2$ yields $h \geq 0$. Here, we encounter an empty interval valuation for h, i.e. a *conflict* in the proof search, as the models of $h < -3$ and of $h \geq 0$ have an empty intersection. The bounds $h < -3$ and $h \geq 0$ are referred to as the *conflicting* bounds. To resolve this conflict situation the simplest technique, called *backtracking*, is to undo the current decision level 4 and to deduce the negation $x < 0$ of the last decision bound $x \geq 0$. In contrast to this simple approach, other conflict resolving methods which take the reasons for the failure into account were proven to be beneficial for (real-world) problems with some structure. In [FHR+07] we showed that simple backtracking is outperformed by our conflict-driven learning approach upon our benchmarks from the hybrid systems domain often by orders of magnitude.

Analogously to DPLL SAT solvers, all implication relationships during iSAT's proof search are recorded in an *implication graph* which is a directed acyclic graph. The implication graph for our example is depicted in Fig. 1. Each vertex represents a decided or deduced atom at the specified decision level. All vertices with directed edges to a vertex v, called *antecedent vertices* of v, are the reasons for that atom. (As well-known from the SAT case, for each vertex in the implication graph which was deduced since a clause became unit, our implementation just stores a pointer to this clause. This part of the implication graph can be constructed on demand.)

Conflict-driven clause learning means to add a *conflict clause* which prevents the solver from visiting the same conflict again. More precisely, if some atoms $a_1, \ldots, a_n$ together imply a conflict then at least one of them must not hold in order to find a solution. This fact can be encoded as a clause $(\neg a_1 \vee \ldots \vee \neg a_n)$ and then added to the clause database of the solver. Such a clause can be generated by an analysis of the implication graph. Therefore, the implication graph is bipartitioned into the *reason side* containing all decision bounds and the *conflict side* containing both conflicting bounds. Such a bipartition is called a *cut*. All vertices on the reason side with directed edges to the conflict side are reasons for the conflict and thus imply the conflict. As you see in Fig. 1, different cuts lead to different conflict clauses, i.e. cut 1: $(y \leq 1 \vee h > -3 \cdot y \vee h < -3 \cdot y \vee h > x^2 \vee h < x^2)$, cut 2: $(y \leq 1 \vee \neg b)$, and cut 3: $(y \leq 1 \vee x < 0)$.

A vertex v of the implication graph is called *unique implication point* (UIP) (adapted from [MSS96]) iff each path from the decision bound ($x \geq 0$ in our example) of the current decision level to one of the conflicting bounds ($h < -3$ or $h \geq 0$ in our example) goes through v and v represents an inequality atom. In our example, $b \geq 1$ and $x \geq 0$ are UIPs. (Note that the corresponding decision bound is

always a UIP by definition.) A UIP is a single reason for the conflict at the current decision level. It is desirable to learn a conflict clause c which contains the negation of a UIP, since after backtracking to the second highest decision level contributing to the conflict, the clause c is always unit and the remaining consistent atom a in c is propagated. This technique is called *non-chronological backtracking* or *backjumping*. As an important remark, the negation of a UIP, i.e. the remaining atom a in c, is *one* inequality atom since we define that the UIP is an inequality atom. This ensures that c is indeed unit. In our example, when learning the clause $(y \leq 1 \vee \neg b)$ (cut 2) the solver backjumps to decision level 1 and propagates $b \leq 0$.

In [ZMMM01], the concept of a UIP is generalized for decision levels less than the current one. Adapted to the iSAT case, a vertex v at some decision level dl is called UIP iff each path from the decision bound of decision level dl to one of the conflicting bounds goes through v, or through a vertex v' of a decision level higher than dl and v' is on the reason side. In our example, $y > 1$ is a UIP at decision level 1 while, e.g., $x = \cos(y)$ is *not* a UIP. UIPs can be ordered for each decision level (cf. Fig. 1). Note that the *last* UIP for any decision level is the decision bound at this decision level. For our experiments we consider some promising clause learning heuristics from [ZMMM01]: The 1UIP scheme means that the conflict clause is generated by the cut where the *first* UIP at the current decision level is on the reason side. This cut is closest to the conflicting bounds. For the 2UIP, 3UIP, ..., allUIP schemes we additionally require that for the highest two, highest three, ..., all decision levels contributing to the conflict their first UIPs are on the reason side. It is also of interest to learn very short conflict clauses. On that account, the learning heuristics "minimal" adds the shortest of the clauses produced by 1UIP, 2UIP, ..., allUIP to the database. The clause learning scheme referred to as "decision" simulates the approach in [JL98] where just the decision bounds are taken into account when generating the conflict clause, i.e. "decision" is a modified version of allUIP where the last UIPs are on the reason side.

Note that in our more general case, the interval for the same variable is narrowed down many times during the propagation phase. As a consequence, the reasons for a conflict possibly contain redundant information. Therefore, we normalize the conflict clause, e.g. if $x > 3.51$, $y < -18.9$, and $x \geq 0.12$ are the reasons for a conflict then we add the conflict clause $(x \leq 3.51 \vee y \geq -18.9)$ since $x > 3.51$ implies $x \geq 0.12$ and thus $x > 3.51$ and $y < -18.9$ lead to a conflict. This observation can also be exploited during determining the bipartition of the implication graph. If, e.g., $x > 3.51$ and $x \geq 0.12$ are currently on the reason side with edges to the conflict side, then we may replace the antecedent vertices of $x \geq 0.12$ by $x > 3.51$, since $x > 3.51$ implies $x \geq 0.12$. Our hope was that this approach cuts off big parts of the implication graph, thus leads to better conflict clauses, and increases the efficiency of the solver. Unfortunately, the performance of this "improved" conflict analysis is very unstable. On some examples it speeds up the solver while on others it slows down the runtime dramatically. We do not have reasons for this behavior yet even after careful profiling.

Restarts. In order to support restarts, we extended the iSAT procedure as shown in Fig. 2. We applied the geometric restart policy from MiraXT [LSB07] with an inner and outer limit. I.e., whenever the restart counter (accumulating the number of conflicts) exceeds the inner limit, a restart is enforced while resetting the restart counter and increasing the inner limit. To circumvent a too fast growth of the inner limit, the latter is reinitialized when it exceeds the outer limit. Then, the outer limit will be increased. The inner and outer limits are initialized with values 500 and 1000, respectively. Thus, restarts are executed after $501, 1252, 1753, 2504, 3630, 5318, 5819, \ldots$ conflicts, i.e. the values of the inner limit are $500, 750, 500, 750, 1125, 1687, 500, \ldots$. (This policy is in some sense similar to [LSZ93] as again and again the inner limit

```
1  iSAT() {
2      restart_limit_inner = 500;
3      restart_limit_outer = 1000;
4      restart_counter = 0;
5      while (true) {
6          while (true) {
7              result = deduce();                  // Deducing.
8              if (result == CONFLICT) {
9                  resolved = analyze_conflict();  // Learning & Backjumping.
10                 if (!resolved) {
11                     return "UNSATISFIABLE";
12                 }
13                 restart_counter++;
14                 if (restart_counter > restart_limit_inner) {
15                     restart_counter = 0;
16                     restart_limit_inner =
17                         restart_limit_inner + (restart_limit_inner >> 1);
18                     if (restart_limit_inner > restart_limit_outer) {
19                         restart_limit_inner = 500;
20                         restart_limit_outer += restart_limit_outer;
21                     }
22                     restart();                  // Restarting.
23                 }
24             }
25             else if (result == SOLUTION) {
26                 return "SATISFIABLE";
27             }
28             else {
29                 break;
30             }
31         }
32         if (!decide_next_branch()) {            // Branching.
33             return "UNKNOWN";
34         }
35     }
36 }
```

Fig. 2. The extended iSAT algorithm supporting restarts

is reset to its initial value 500.) We found out that these initial values are the best choice on average for our experiments. Smaller (≤ 100) and greater (≥ 1000) initial values for the inner limit do not pay off as well on our set of benchmarks.

4 Experimental results

In this section, we present the results of benchmarking the described learning schemes both *with* and *without* restarts. We considered bounded model checking (BMC) problems, i.e., proving a property of a hybrid discrete-continuous transition system for a fixed unwinding depth k. Besides the well-known *bouncing ball* example, these BMC problems are two versions of a controller for the *European Train Control System* (ETCS) Level 3 (e.g., cf. [HEFT08]), an *aircraft collision avoidance protocol* (similar to [TPS98]), and a protocol for an automated *parking maneuver* of a car. To achieve comparable figures, all BMC instances were solved *without* any BMC optimizations as, e.g., discussed in [FH07] and with the same static round-robin variable selection heuristics. The results are listed in Table 1 and 2. The number in parentheses following the benchmark's name is the number of BMC instances to be solved. All instances but the last one (of each benchmark) are *unsatisfiable*. A bold number n in the same line as the benchmark's name means that the corresponding learning scheme was just able to solve the first n instances within the time limit of 30 minutes. The corresponding results (in slanted numbers) then hold for the first n instances. All benchmarks were performed on an 1.83 GHz Intel Core 2 Duo machine with 1 GByte physical memory running Linux.

benchmark	1UIP	2UIP	3UIP	4UIP	5UIP	allUIP	minimal	decision
bouncing ball (57)								
#conflicts (x10^3)	**17.8**	19.6	**16.8**	**17.6**	23.3	26.0	**17.4**	82.0
atoms/conflict clause	**34.0**	38.7	43.5	46.0	45.8	**31.0**	**24.4**	41.7
#decisions (x10^3)	77.3	85.8	71.5	77.8	103.7	113.9	76.4	325.5
#implications (x10^6)	22.9	24.1	20.0	22.6	29.2	32.0	21.4	81.3
runtime (sec)	**50.3**	58.0	**48.3**	56.3	76.2	83.3	57.3	278.6
ETCS3 v1 (33)								**25**
#conflicts (x10^3)	**39.3**	43.2	**34.4**	51.3	**37.7**	**37.4**	**37.9**	*142.0*
atoms/conflict clause	159.5	215.9	243.0	430.1	309.6	171.1	**109.7**	*177.0*
#decisions (x10^3)	135.4	151.6	122.6	175.4	131.0	127.9	138.6	*429.9*
#implications (x10^6)	113.8	134.6	97.9	160.4	109.1	112.5	116.8	*173.6*
runtime (sec)	**419.8**	565.3	**428.7**	953.3	514.0	625.1	760.7	*1510.2*
ETCS3 v2 (17)								**16**
#conflicts (x10^3)	8.4	9.5	7.5	8.2	**5.5**	7.8	11.5	*79.1*
atoms/conflict clause	**91.6**	128.8	131.7	169.9	135.8	**95.9**	**94.0**	*148.4*
#decisions (x10^3)	26.7	29.6	24.1	24.4	18.0	23.9	33.0	*203.8*
#implications (x10^6)	12.7	12.3	13.3	11.4	7.4	11.3	17.1	*62.1*
runtime (sec)	45.7	50.5	50.1	41.8	**24.4**	54.1	105.9	*678.5*
aircraft protocol (3)						**2**		**1**
#conflicts (x10^3)	24.1	**4.8**	14.9	**9.5**	17.3	*3.6*	**4.7**	*< 0.1*
atoms/conflict clause	41.8	**34.3**	40.4	66.9	86.8	*34.4*	**23.9**	*0.0*
#decisions (x10^3)	52.8	12.3	43.6	24.4	42.8	*8.2*	13.7	*0.0*
#implications (x10^6)	15.1	1.5	5.8	4.4	7.6	*1.3*	1.6	*< 0.1*
runtime (sec)	214.6	**7.08**	38.7	**28.4**	64.8	*7.7*	**8.3**	*< 0.1*
parking maneuver (53)								
#conflicts (x10^3)	5.0	5.2	4.9	4.5	4.7	4.7	7.1	76.2
atoms/conflict clause	**21.5**	**24.3**	31.5	39.0	45.3	67.2	**17.9**	127.2
#decisions (x10^3)	23.3	32.8	30.4	30.0	31.4	31.4	55.5	307.4
#implications (x10^6)	18.3	12.8	12.7	11.4	10.9	10.9	15.7	136.4
runtime (sec)	54.3	**39.2**	**41.2**	**41.8**	**42.6**	90.2	144.7	1694.1

Table 1. Empirical results: different learning heuristics *without* restarts

Learning heuristics without restarts. First of all, we noticed that the learning scheme "decision", which is similar to the method implemented in [JL98], performs worst compared to all other schemes, and was not able to solve all instances of the more complex ETCS and aircraft protocol benchmarks within the time limit. Note that "decision" is the most expensive method for the clause generation since it traces back the entire implication graph until all decision bounds contributing to the conflict are found. Moreover, as indicated by the number of encountered conflicts, which is significantly higher than with other schemes even if not all instances are solved, the quality of conflict clauses generated by "decision" seems to be bad.

Before we consider the runtimes, we have a look at the "quality" of the different schemes, i.e. the average size of learned clauses, encountered conflicts, and their combination. Concerning the size of learned clauses, the conflict analysis procedure "minimal" behaves best while, surprisingly, conflict clauses generated by 1UIP often are just negligibly larger. Except for the parking maneuver, allUIP produces short clauses also. All other schemes are not that stable. However, our results do not give a clear answer which heuristics encounter the smallest number of conflicts in general. Considering the product of the number of conflicts and the average clause size, which is actually the total number of learned atoms, it turns out that "minimal" and –except for the aircraft protocol– 1UIP are most efficient. This result is very relevant because in addition to the runtime of a solver, memory consumption is equally important.

Although "minimal" and allUIP show a nice learning behavior, they are not competitive wrt. the runtimes, obviously due to the additional overhead within the

benchmark	1UIP	2UIP	3UIP	4UIP	5UIP	allUIP	minimal	decision
bouncing ball (57)								
#conflicts (x10^3)	**15.1**	18.3	17.5	19.9	**14.7**	23.4	**15.8**	91.9
atoms/conflict clause	**32.8**	38.2	46.6	46.1	43.0	**30.3**	**23.1**	45.5
#restarts	10	16	15	17	10	23	13	103
runtime (sec)	**42.7**	54.2	59.4	65.3	52.6	80.2	49.8	344.0
ETCS3 v1 (33)								**28**
#conflicts (x10^3)	38.6	37.5	35.7	36.1	34.4	36.2	32.7	*139.0*
atoms/conflict clause	**141.4**	172.3	259.6	268.3	273.2	169.0	**109.9**	*183.8*
#restarts	43	40	38	36	37	37	35	*123*
runtime (sec)	**496.0**	**490.8**	**491.2**	537.9	**468.3**	614.4	586.3	*1747.5*
ETCS3 v2 (17)								
#conflicts (x10^3)	8.2	7.7	**5.6**	8.5	9.1	9.2	8.8	60.5
atoms/conflict clause	**80.5**	**89.8**	97.2	171.8	206.9	110.2	**84.1**	139.2
#restarts	10	9	6	9	9	9	9	48
runtime (sec)	**41.5**	**37.8**	**30.9**	62.4	63.6	74.4	70.6	575.1
aircraft protocol (3)								
#conflicts (x10^3)	**2.7**	**3.0**	6.9	6.8	10.1	8.7	20.7	68.5
atoms/conflict clause	**15.3**	**18.0**	31.3	44.7	42.4	28.5	29.8	50.5
#restarts	4	4	9	8	12	11	18	37
runtime (sec)	**3.1**	**3.6**	12.6	11.8	21.8	23.1	122.1	584.8
parking maneuver (53)								
#conflicts (x10^3)	**5.0**	16.7	**7.7**	11.8	41.0	**7.5**	33.7	75.9
atoms/conflict clause	**20.7**	**22.8**	43.8	73.4	114.2	78.8	**16.7**	119.7
#restarts	3	15	7	11	25	7	22	60
runtime (sec)	**36.3**	92.9	71.4	150.0	564.8	219.3	675.1	1773.6

Table 2. Empirical results: different learning heuristics *with* restarts

conflict analysis procedures. Again, we cannot detect a fastest scheme. However, due to its competitive runtimes and excellent memory consumption, we regard the 1UIP conflict analysis scheme as the best choice among the others, also in the mixed Boolean and arithmetic framework.

Finally, we investigate the bad performance of 1UIP on the aircraft protocol benchmark: On the *unsatisfiable* first and second BMC instances of this benchmark, 1UIP is faster than all other schemes. I.e., 1UIP is outperformed on a *satisfiable* formula. Since all versions of the solver use the same static variable selection heuristics, finding a satisfying branch earlier than other schemes is due to different backjump levels in interplay with the accidental position of the variable selection pointer. Therefore, we do not see the bad performance on just one (satisfiable) formula as a major drawback of the 1UIP heuristics. We will see that the runtimes change dramatically when restarting is applied. However, an important question for future work is which decision heuristics performs best in the iSAT setting both on satisfiable and unsatisfiable formulae. For some ideas the reader is referred to [THF+07, Section 4].

Restarts. First note that the described restart policy is applied for each BMC instance, i.e. the inner and outer limits were reset to its initial values for each instance. Consequently, the total number of conflicts and the total number of restarts per benchmark do not correspond to the policy in general.

Looking at the results presented in Table 2, we conclude that the restart technique is in general also worthwhile in the iSAT framework. In particular, the learning schemes allUIP and "decision" were now able to solve almost all BMC instances. ("decision" still fails on ETCS3 v1, albeit solving 28 instances now.) Most impressively, on the aircraft protocol benchmark the speed-ups obtained are up to a factor of 70. This performance gain is a consequence of the fact that the restart heuristics pays off enormously on the satisfiable third BMC instance. Very surprisingly, restarting is also beneficial for unsatisfiable formulae. On most of the unsatisfiable

instances, all schemes perform better when restarting is employed. The other way round, the slow-downs of the solver for the schemes 2UIP, ..., allUIP, and "minimal" on the parking maneuver benchmark originate from the 53^{rd} satisfiable instance. Unexpectedly, the aforementioned schemes show a bad behavior on this concrete satisfiable instance. Again, we point to the open question of suitable decision heuristics for the iSAT case. We conjecture that some *dynamic* decision heuristics which make potentially better decisions after restarting, e.g. a generalized VSIDS [MMZ+01], will help here.

However, the most important observation and the main result of this paper is that 1UIP is the most efficient and most stable conflict analysis scheme among all others when restarting is exploited, on both satisfiable and unsatisfiable formulae.

5 Conclusion and future work

In this paper, we gave a detailed insight into the conflict analysis procedure of the iSAT algorithm. We discussed different conflict analysis heuristics in order to answer the question whether the efficiency of the solver in our more general case of mixed Boolean and non-linear arithmetic constraint formulae can be improved by more complex learning schemes. On our set of benchmarks from the hybrid systems domain we found out that, analogously to the purely propositional case, the (adapted) 1UIP technique is the most efficient and most stable one concerning both runtime and memory consumption.

Another issue we investigated is the integration of restarts into the iSAT algorithm. Our empirical results show that in general the restart heuristics is beneficial on satisfiable as well as unsatisfiable mixed Boolean and arithmetic formulae. It turns out that again the 1UIP technique further improves the efficiency of the solver when restarting is applied, while other learning schemes are not that stable across the benchmarks.

For future work, we target two main directions. First, we will further increase the *efficiency* of the HySAT tool. As already mentioned, one crucial issue is to investigate suitable decision heuristics which are of utmost importance for the stability of modern propositional SAT solvers. Additionally, we will enhance our tool by numerical methods like the interval Newton method and linear programming, and go for parallel versions of HySAT, e.g. the classical divide-and-conquer approach and parallel solving of different BMC instances.

A second direction is the *extension* of HySAT in several ways. Some of them are (a) the integration of safe numerical solving of *ordinary differential equations* (ODEs) in order to directly handle ODEs in the solver without an a priori approximation (cf. [Egg06]), (b) a generalization of the iSAT algorithm wrt. supporting *stochastic quantification* of discrete variables for the reachability analysis of probabilistic hybrid systems (cf. [Tei07,FHT08]), and (c) the *generation of Craig interpolants* for *full* model checking of hybrid systems.

References

[BG06] F. Benhamou and L. Granvilliers. Continuous and interval constraints. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, Foundations of Artificial Intelligence, chapter 16, pages 571–603. Elsevier, Amsterdam, 2006.

[BHZ06] L. Bordeaux, Y. Hamadi, and L. Zhang. Propositional Satisfiability and Constraint Programming: A comparative survey. *ACM Comput. Surv.*, 38(4):12, 2006.

[BNOT06] C. Barrett, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Splitting on Demand in SAT Modulo Theories. In M. Hermann and A. Voronkov, editors, *Proceedings of the 13th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR'06)*, volume 4246 of *LNCS*, pages 512–526. Springer, 2006.

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

[DLL62] M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem Proving. *Communications of the ACM*, 5:394–397, 1962.

[DP60] M. Davis and H. Putnam. A Computing Procedure for Quantification Theory. *Journal of the ACM*, 7(3):201–215, 1960.

[Egg06] A. Eggers. Einbettung sicherer numerischer Integration von Differentialgleichungen in DPLL-basiertes arithmetisches Constraint-Solving für hybride Systeme. Master's thesis, Carl von Ossietzky Universität, Dpt. Informatik, Oldenburg, Germany, 2006. in German.

[ES03] N. Eén and N. Sörensson. An Extensible SAT-Solver. In *6th International Conference on Theory and Applications of Satisfiability Testing*, 2003.

[FH07] M. Fränzle and C. Herde. HySAT: An efficient proof engine for bounded model checking of hybrid systems. *Formal Methods in System Design*, 30(3):179–198, 2007.

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

[FHT08] M. Fränzle, H. Hermanns, and T. Teige. Stochastic Satisfiability Modulo Theory: A Novel Technique for the Analysis of Probabilistic Hybrid Systems. In *Proceedings of the 11th International Conference on Hybrid Systems: Computation and Control (HSCC'08)*, 2008.

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

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

[HEFT08] C. Herde, A. Eggers, M. Fränzle, and T. Teige. Analysis of Hybrid Systems using HySAT. In *Proceedings of the Third International Conference on Systems (ICONS 2008)*. IEEE Computer Society, 2008.

[Hua07] J. Huang. The Effect of Restarts on the Efficiency of Clause Learning. In M. M. Veloso, editor, *IJCAI*, pages 2318–2323, 2007.

[JL98] N. Jussien and O. Lhomme. Dynamic domain splitting for numeric CSPs. In *European Conference on Artificial Intelligence*, pages 224–228, Brighton, United Kingdom, August 1998.

[KB05] G. Katsirelos and F. Bacchus. Generalized NoGoods in CSPs. In M. M. Veloso and S. Kambhampati, editors, *AAAI*, pages 390–396. AAAI Press / The MIT Press, 2005.

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

[LSTV07] C. Lecoutre, L. Saïs, S. Tabary, and V. Vidal. Recording and Minimizing Nogoods from Restarts. *JSAT Special Issue on SAT/CP Integration*, 1:147–167, 2007.

[LSZ93] M. Luby, A. Sinclair, and D. Zuckerman. Optimal speedup of Las Vegas algorithms. *Inf. Process. Lett.*, 47(4):173–180, 1993.

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

[MSS96] J. P. Marques-Silva and K. A. Sakallah. GRASP – a new search algorithm for satisfiability. In *ICCAD '96: Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design*, pages 220–227, Washington, DC, USA, 1996. IEEE Computer Society.

[NOT06] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, November 2006.

[Pro93] P. Prosser. Hybrid Algorithms for the Constraint Satisfaction Problem. *Computational Intelligence*, 9:268–299, 1993.

[RCOJ06] G. Richaud, H. Cambazard, B. O'Sullivan, and N. Jussien. Automata for Nogood Recording in Constraint Satisfaction Problems. In *Proceedings of the CP 2006 First International Workshop on the Integration of SAT and CP Techniques*, pages 113–127, 2006.

[Tei07] T. Teige. SAT-Modulo-Theory based Analysis of Probabilistic Hybrid Systems. In *Proceedings of the Dagstuhl Graduate School Meeting 2007 "Dagstuhl Zehn plus Eins"*, pages 86–87, Aachen, 2007. Verlag Mainz. ISBN: 3-86130-882-7.

[THF+07] T. Teige, C. Herde, M. Fränzle, N. Kalinnik, and A. Eggers. A Generalized Two-watched-literal Scheme in a mixed Boolean and Non-linear Arithmetic Constraint Solver. In *Proceedings of the* 13^{th} *Portuguese Conference on Artificial Intelligence (EPIA 2007)*, December 2007. To appear.

[TPS98] C. Tomlin, G. Pappas, and S. Sastry. Conflict resolution for air traffic management : A study in muti-agent hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):509–521, April 1998.

[WvB07] H. Wu and P. van Beek. On Universal Restart Strategies for Backtracking Search. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, pages 681–695, 2007.

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