Journal of Automated Reasoning (2005) 35: 237–263
DOI: 10.1007/s10817-005-9002-1

The SAT-based Approach to Separation Logic

ALESSANDRO ARMANDO, CLAUDIO CASTELLINI,
ENRICO GIUNCHIGLIA and MARCO MARATEA
DIST, University of Geneva, Viale F. Causa, 13-16145 Geneva, Italy.
e-mail: {armando, drwho, enrico, marco}@dist.unige.it

Abstract. The SAT-based approach to the decision problem for expressive, decidable, quantifier-free first-order theories has been investigated with remarkable results at least since 1993. One such theory, successfully employed in the formal verification of complex, infinite state systems, is Separation Logic (SL), which combines Boolean logic with arithmetic constraints of the form $x - y \bowtie c$, where $\bowtie$ is $\leq$, $<$, $>$, $\geq$, $=$, or $\neq$. The SAT-based approach to SL was first proposed and implemented in 1999: the results in terms of performance were good, and since then a number of other systems for SL have appeared. In this paper we focus on the problem of building efficient SAT-based decision procedures for SL. We present the basic procedure and four optimizations that improve dramatically its effectiveness in most cases: (a) IS_2 preprocessing, (b) early pruning, (c) model reduction, and (d) best reason detection. For each technique we give an example of how it might improve the performance. Furthermore, for the first three techniques, we give a pseudo-code representation and formally state the soundness and completeness of the resulting optimized procedure. We also show how it is possible to check the satisfiability of valuations involving constraints of the form $x - y < c$ using the Bellman–Ford algorithm. Lastly, we present an extensive comparative experimental analysis, showing that our solver TSAT++, built along the lines described in this paper, is currently the state of the art on various classes of problems, including randomly generated, hand-made, and real-world instances.

Key words: SAT-based decision procedures, separation logic.

1. Introduction

The SAT-based approach to satisfiability problems beyond propositional logic dates back to at least the early 1990s (Armando and Giunchiglia, 1993), when it was noted that, under some suitable conditions, the problem of determining the satisfiability of any decidable, quantifier-free first-order theory can be reduced to Boolean search coupled with a satisfiability procedure (i.e., procedure capable of deciding whether any given set of literals in satisfiable or not w.r.t. the given theory). In more detail, the SAT-based approach to the satisfiability problem of a formula ϕ in a theory T amounts to using:

- a SAT solver to *generate* a valuation μ entailing ϕ in propositional logic, and
- a satisfiability procedure to *test* whether μ is satisfiable in the theory T,

till a satisfiable μ is found (in which case also ϕ is satisfiable), or a set of valuations whose disjunction is logically equivalent to ϕ has been generated and tested (in which case ϕ is unsatisfiable). Over the years, the SAT-based approach has been applied to more theories and even to different problems, such as propositional modal logics (Giunchiglia and Sebastiani, 1996; Giunchiglia et al., 2002), conformant planning (Castellini et al., 2003), and combination of expressive theories (Stump et al., 2002), with remarkable results. As the research proceeded, it became clear that the approach could harvest the technological improvements achieved in propositional satisfiability. See (Armando et al., 2005b) for a unifying perspective on the SAT-based approach.

Many verification and scheduling problems involve arithmetic constraints of the form $x - y \bowtie c$, where x and y are variables ranging over the reals or the integers and $\bowtie$ is $\leq$, $<$, $>$, $\geq$, $=$, or $\neq$. These constraints are called *separation terms* by Pratt (Pratt, 1977), and *Separation Logic* (from now on, SL) is the name now used to denote the logic allowing for arbitrary Boolean combination of separation terms.[★] SL is also called “difference logic” by some authors (see, e.g., Cotton et al., (2004)) and can be seen as a generalization of a well-known framework for temporal reasoning, the Temporal Constraint Network, introduced by Decther, Meiri and Pearl (Dechter et al., 1989). SL is the logic we focus on in this paper.

The first application of the SAT-based approach to a significant fragment of SL was given in Armando et al. (1999). In this case, as well as with modal logics and conformant planning, excellent results were obtained. Since then, a number of other systems for SL have appeared (see, e.g., Oddi and Cesta, 2000; Audemard et al., 2002; Strichman et al., 2002; Armando et al., 2005a; Cotton et al., 2004).

In this paper we focus on the problem of building efficient SAT-based decision procedures for SL. To this end, we present the basic procedure and four optimizations that improve dramatically its effectiveness in most cases: (a) IS_2 preprocessing, (b) early pruning, (c) model reduction, and (d) best reason detection. Optimizations (a) and (b) were first proposed in Armando et al. (1999), whereas (c) and (d) have been presented for the first time in Armando et al. (2005a). For each technique we give an example of how it might improve performance. Furthermore, for the first three techniques, we give a pseudo-code representation and formally state the soundness and completeness of the corresponding procedure. We also show how it is possible to check the satisfiability of valuations involving constraints of the form $x - y < c$ using the well-known Bellman–Ford algorithm (from now on, BF).

We then present an extensive comparative experimental analysis, showing that our solver TSAT++, built along the theoretical lines of the approach, is

[★] Unfortunately, the name *Separation Logic* is also used to denote an extension of Hoare logic. Strichman et al. (2002) is the first reference we are aware of where the name is resumed from Pratt’s paper.

currently the state of the art on various classes of problems, including randomly generated, hand-made, and real-world instances.

The paper is structured as follows. Section 2 is about SL and presents its syntax, semantics, and some other formal properties of SL; Section 3 introduces the basic SAT-based procedure for SL, while the optimizations are presented in Section 4; in Section 5 we present a satisfiability algorithm for valuations based on BF; in Section 6 we describe the actual implementation of our system and present a thorough experimental evaluation; in Section 7 we outline the related work; lastly, in Section 8 we have the conclusions.

2. Theoretical Background

In this section we give some theoretical background and fix the terminology that will be used throughout the paper.

2.1. SEPARATION LOGIC

2.1.1. *Syntax*

Let $\mathcal{V}$ and $\mathcal{P}$ be two disjoint sets of symbols, called *variables* and *propositional letters*, respectively. A *constraint* is an expression of the form $x - y \bowtie c$, where $x, y \in \mathcal{V}$, $\bowtie \in \{\leq, <, >, \geq, =, \neq\}$ and c is a numeric constant. The notations $\underline{x} \bowtie y + c$ and $x - c \bowtie y$ will also be freely used in place of $x - y \bowtie c$. An *atom* is either a constraint or a propositional letter. A *formula* is a combination of atoms via the unary connective "$\neg$" for negation and the n-ary connectives "$\wedge$" and "$\vee$" ($n \geq 0$) for conjunction and disjunction, respectively. We will write $\top$ and $\bot$ for the empty conjunction and the empty disjunction, respectively. A *literal* is either an atom or its negation. If a is an atom, then $\bar{a}$ abbreviates $\neg a$ and $\overline{\neg a}$ stands for a.

EXAMPLE 1. In Bryant et al. (2002), the case-study is introduced of a bounded model checking problem for the memory unit of the Motorola Elf microprocessor. The unit is initially modeled as 20 K lines of VERILOG, with 80 integer-valued variables and 70 propositional letters. After some translation stages, the problem is reduced to checking satisfiability of a formula in SL, a fragment of which, call it ϕ_{Elf}, looks like this:

$$\begin{array}{ll}
(p_1 \vee \neg(VPred = I_{RR})) & \wedge \\
(\neg p_1 \vee VPred = I_{RR}) & \wedge \\
(\neg p_2 \vee VPred < I_{RR} + 1) & \wedge \\
(p_2 \vee \neg(VPred < I_{RR} + 1)) & \wedge \\
(p_3 \vee p_4) & \wedge \\
(p_3 \vee \neg p_4 \vee \neg VenI' = VenI) & \wedge \\
(p_5 \vee \neg(VenI' + 2 = VenI)) & \wedge \\
(\neg p_5 \vee VenI' + 2 = VenI) &
\end{array}$$

In the above formula, $VPred$, I_{RR}, $VenI$, $VenI'$ are variables and p_1, p_2, p_3, p_4, p_5 are propositional letters. $VPred < I_{RR} + 1$ is a constraint, and $p5$ and $\neg(VenI' + 2 = VenI)$ are literals.

2.1.2. *Semantics*

Let the set $\mathbb{D}$ (*domain of interpretation*) be either the set of the real numbers $\mathbb{R}$ or the set of integers $\mathbb{Z}$. An *assignment* is a total function mapping variables to $\mathbb{D}$ and propositional letters to the truth values *false* and *true*, standing for falsehood and truth respectively.

Let σ be an assignment and ϕ be a formula. Then $\sigma \models \phi$ (σ satisfies *a formula* ϕ) is defined as follows.

$\sigma \models x - y \bowtie c$ if and only if $\sigma(x) - \sigma(y) \bowtie c$,
$\sigma \models p$ with $p \in \mathcal{P}$ if and only if $\sigma(p) = true$,
$\sigma \models \neg\phi$ if and only if it is not the case that $\sigma \models \phi$,
$\sigma \models (\wedge_{i=1}^{n} \phi_i)$ if and only if for each $i \in [1, n]$, $\sigma \models \phi_i$, and
$\sigma \models (\vee_{i=1}^{n} \phi_i)$ if and only if for some $i \in [1, n]$, $\sigma \models \phi_i$.

If $\sigma \models \phi$, then σ will also be called a *model* of ϕ. We also say that

- a formula ϕ is *satisfiable* if and only if there exists an assignment satisfying it;
- a formula ϕ is *valid* if and only if every assignment satisfies it;
- two formulas ϕ and ψ are *logically equivalent* if and only if the formula $(\neg\phi \vee \psi) \wedge (\phi \vee \neg\psi)$ is valid.

Here we consider the problem of deciding whether a formula is satisfiable or not in the given domain of interpretation $\mathbb{D}$. Notice that satisfiability of a formula depends on $\mathbb{D}$, e.g., $x - y > 0 \wedge x - y < 1$ is clearly satisfiable if $\mathbb{D}$ is $\mathbb{R}$ but unsatisfiable if $\mathbb{D}$ is $\mathbb{Z}$. However, the problems of checking satisfiability in $\mathbb{Z}$ and $\mathbb{R}$ are closely related and will be treated uniformly almost always. Therefore, from now on, we will drop the distinction, and we will reintroduce it only when needed.

EXAMPLE 2. Consider Example 1. ϕ_{Elf} is satisfiable, and a model is $\sigma = \{p_1 \mapsto true, VPred \mapsto 12, I_{RR} \mapsto 12, p_2 \mapsto true, p_3 \mapsto true, p_4 \mapsto true, p_5 \mapsto true, VenI \mapsto 10, VenI' \mapsto 8\}$.

2.2. VALUATIONS

A *valuation* is a finite set μ of literals such that for each atom a, if $a \in \mu$ then $\neg a \notin \mu$. In the following if μ is a valuation, then by μ we also denote the formula $\wedge_{l \in \mu} l$. Context will make clear what is intended. Moreover, we say that

1. a valuation μ *propositionally entails* a formula ϕ if $(\neg\mu \wedge \phi)$ can be proved in propositional logic;
2. two formulas are *propositionallly logically equivalent* if one formula propositionally entails the other, and *vice versa*.

The following result shows the importance of valuations.

THEOREM 3. *A formula ϕ is satisfiable if and only if there exists a valuation μ such that*

1. *μ is satisfiable,*
2. *all atoms in μ occur in ϕ, and*
3. *μ propositionally entails ϕ.*

Proof. The right-to-left direction is trivial. For the left-to-right direction, first notice that it is always possible to convert ϕ to a logically equivalent formula in the same atoms and in disjunctive normal form (DNF). Let S be the set of disjuncts in the DNF. Then by the semantics of $\wedge$ it follows that ϕ is satisfiable if and only if there is $\mu \in S$ such that μ is satisfiable. Furthermore, for such μ, also the second and third properties hold. □

Given the above result, in order to check the satisfiability of a formula ϕ, the issue becomes that of efficiently building a set S of valuations that is *propositionally complete for* ϕ, that is, such that the disjunction of the valuations in S is propositionally logically equivalent to ϕ. Given such a set, we can then separately check the satisfiability of its elements.

3. The SAT-based Approach to Separation Logic

Theorem 3 lays the foundation of a simple method for determining the satisfiability of a formula ϕ:

1. *generate* a set S of valuations that is propositionally complete for ϕ, and then
2. *test* whether at least one of the valuations in S is satisfiable: if this is the case, then ϕ is satisfiable; otherwise ϕ is unsatisfiable.

Further, if one valuation μ in S is satisfiable, then the models of μ are also models of ϕ. Thus, in the above schema, the problem of finding a model of an arbitrary formula has been reduced to the problem of finding a model of a valuation. Notice that the ability to return a model if the formula is satisfiable is highly desirable in many applications. For example, if the formula represents an instance of a bounded model-checking problem, then from any model of the formula it is usually possible to extract a trace witnessing the violation of the desired property.

The reason why this method has become quite popular is that state-of-the-art SAT solvers can be employed to efficiently generate valuations on-the-fly. In fact, valuations propositionally entailing the formula can be generated one by one, and each can then be checked for satisfiability before generating the next one, until a positive answer is returned, or there are no more valuations left. This way the need to generate all (potentially exponentially many) satisfying valuations beforehand is avoided. This is the foundation of the SAT-based approach, first envisioned in Armando and Giunchiglia (1993) and first applied to SL in Armando et al. (1999).

The reasons of its success are at least three:

1. more than 40 years of research on propositional satisfiability have made SAT solvers reliable, efficient and, in some cases, reusable;
2. the two phases, namely, enumeration and satisfiability checking, can be effectively decoupled, nevertheless allowing for a great deal of search guiding information to flow between the modules that take care of each phase;
3. the range of theories this approach can tackle is quite wide and interesting.

In the rest of this section we give a precise characterization of the SAT-based approach and prove its fundamental properties.

Without loss of generality, in the following we assume that all formulas are in conjunctive normal form (CNF) and do not contain any constraint of the form $x - y - c$ or $x - y \neq c$. Constraints of the form $x - y = c$ and $x - y \neq c$ can be always replaced by the logically equivalent formulas $(x - y \leq c) \wedge (x - y \geq c)$ and $(x - y > c) \vee (x - y < c)$ respectively. Further, by using the structure-preserving clause form transformation described in, for example, Tseitin, 1970; Plaisted and Greenbaum, 1986, translation in CNF can be done efficiently. Given the CNF assumption, a formula is represented as a conjunctively intended set of clauses, each *clause* being a disjunctively intended set of literals.

3.1. BASIC PROCEDURE

A pseudo-code description of a procedure that can be used to carry out the propositional analysis phase is given in Figure 1. It is essentially the Davis, Logemann and Loveland algorithm (from now on, DLL) (Davis et al., 1962) for propositional satisfiability extended in such a way to support the enumeration of all the valuations propositionally entailing the input formula.

```
function DLL_ENUM($\phi$,$\mu$)
1  if {} $\in \phi$ then return FALSE
2  if $\phi = \emptyset$ then Print($\mu$); return FALSE
3  if {$l$} $\in \phi$ then return DLL_ENUM(Simplify($l$,$\phi$),$\mu \wedge l$)
4  $l$ := ChooseLiteral($\phi$)
5  return DLL_ENUM(Simplify($l$,$\phi$),$\mu \wedge l$) or
          DLL_ENUM(Simplify($\bar{l}$,$\phi$),$\mu \wedge \bar{l}$)
```

Figure 1. DLL algorithm as enumerator.

In the procedure:

1. Simplify (l, ϕ) simplifies the formula ϕ under the assumption that the literal l is true. This is done by removing from ϕ all clauses in which l appears and by moving $\bar{l}$ from all clauses in which $\bar{l}$ appears;
2. ChooseLiteral(ϕ) picks a literal l in ϕ according to some heuristic function.

Notice that if $\phi = \emptyset$, then the current valuation, μ, is printed and FALSE is returned so as to force backtracking.

There is strong empirical evidence in the literature (see, e.g., Le Berre and Simon (2003)) that DLL is the current best among the complete algorithms for solving the SAT problem. A number of improvements to DLL have been proposed, especially on the heuristic function used in ChooseLiteral(ϕ), on the data structures employed, on the way unit propagation and backtracking are performed, but the basic algorithm still stands unchanged.

LEMMA 4 (DLL as an enumerator). *Let ϕ be a propositional formula.* DLL_ENUM(ϕ, $\top$) *prints a set of valuations that is propositionally complete for ϕ.*

Proof. The statement is proved in Giunchiglia et al. (2002). □

DLL_ENUM(ϕ,μ) can be readily turned into a decision procedure for SL as shown in Figure 2. The modifications are limited to the case in which $\phi = \emptyset$ Instead of printing μ and unconditionally returning FALSE, we now return the result of invoking SatCheck(μ), where SatCheck(μ) is a satisfiability procedure for valuations, that is, it returns TRUE if μ is satisfiable, and FALSE otherwise. This procedure clearly depends on the decidable theory under consideration. As we will see in Section 5, a satisfiability procedure for SL valuations can be readily built by using BF, which runs in polynomial time (see, e.g., Cormen et al. (2001)).

THEOREM 5 (Soundess and completeness of TSAT). *Let ϕ be a formula. Then TSAT (ϕ, $\top$) returns* TRUE *if ϕ is satisfiable, and* FALSE *otherwise.*

Proof. It readily follows from Theorem 3, from the soundness and completeness of the DLL algorithm, and from Lemma 4. □

```
function TSAT($\phi$,$\mu$)
1  **if** $\{\} \in \phi$ **then return** FALSE
2  **if** $\phi = \emptyset$ **then return** SatCheck($\mu$)
3  **if** $\{l\} \in \phi$ **then return** TSAT(Simplify($l$,$\phi$),$\mu \wedge l$)
4  $l$ := ChooseLiteral($\phi$)
5  **return** TSAT(Simplify($l$,$\phi$),$\mu \wedge l$) **or**
             TSAT(Simplify($\bar{l}$,$\phi$),$\mu \wedge \bar{l}$)
```

Figure 2. Basic SAT-based decision procedure based on DLL.

EXAMPLE 6. Once again, let us consider Example 1. Assume, moreover, that ChooseLiteral simply returns the first atom in lexicographical order. Then here is how TSAT (ϕ_{Elf}, $\top$) works:

1. since there are no unit clauses, p_1 is chosen and $\mu = \{p_1\}$;
2. after Simplify (p_1, ϕ_{Elf}) is executed, the second clause has become unit since $\neg p_1$ has been removed from it; therefore $VPred = I_{RR}$ is detected as appearing in a unit clause and added to μ;
3. same as Items 1 and 2, but with p_2 and $VPred < I_{RR} + 1$; now $\mu = \{p_1, VPred = I_{RR,p2}, VPRED < I_{RR} + 1\}$;
4. again, there are no unit clauses, and therefore p_3 is chosen and added to μ;
5. after Simplify (p_3, ϕ_{Elf}) is executed, no unit clauses are left, so p_5 is chosen and added to μ;
6. lastly, $VenI' + 2 = VenI$ is detected in a unit clauses and added to μ where now $\{p_1, VPred = I_{RR,p2}, VPred < I_{RR} +1, p_3, p_5, VenI' + 2 = VenI\}$;
7. ϕ_{Elf} has now become empty; SatCheck is called and a model of μ, which also is a model of ϕ_{Elf}, is found, for instance, the model in Example 2.

4. Optimizations

The clear separation between the enumeration of valuations propositionally entailing ϕ and the check of their satisfiability is the key feature of the SAT-based approach to building decision procedure. However, the naïve application of this idea may suffer from the generation of exponentially many unsatisfiable valuations. The reason for this inefficiency is that the SAT solver is not aware of the properties of the background theory, in our case SL. To illustrate this point, let us again consider the problem of Example 1. If $VPred = I_{RR}$ is assigned to true then it is pointless to assign false to $VPred < I_{RR} + 1$ as this valuation (or any extension thereof) will be later found to be unsatisfiable and hence rejected by SatCheck.

As a matter of fact most optimizations to the basic procedure that have been proposed in the literature aim at preventing the generation of unsatisfiable (and hence useless) valuation. In this section we described four optimization that – as shown in Section 6 – make TSAT++ the current fastest decision procedure for SL on a wide range of benchmark problems.

4.1. IS_n PREPROCESSING

To reduce the enumeration of unfruitful valuations at a reasonable price, Armando et al., 1999 introduced the so-called IS_n preprocessing. The name stands for *inconsistent subsets* and the subscript number represents the size of the subsets sought for. Naively put, if P is the set of constraint literals occurring positively in the input formula, IS_n checks the satisfiability of all the valuations P' subset of P such that $|P'| \leq n$: for each unsatisfiable subset P', the clause $\vee_{l \in P'} \bar{l}$ is added to the imput formula before calling TSAT.

Although IS_n can be exponential in general, for each fixed n polynomially many subsets of cardinality n exists, and if satisfiability checking is done in polynomial time, the resulting procedure runs in polynomial time.

For a given value of n, it also makes sense to generalize the idea in order to check the satisfiability of set P, with $|P| \leq n$, of literals whose atom occurs in the input formula. To ease the presentation, we restrict to the case in which $n = 2$. The generalization of IS_2 works as follows: for each unordered pair $\{c_i, c_j\}$ of distinct SL-constraints appearing in ϕ and involving the same variables, all possible pairs of literals built out of them are checked for satisfiability.

The resulting optimized version of TSAT is given in Figure 3.

THEOREM 7 (Soundness and completeness of TSAT_IS$_2$). *Let ϕ be a formula. Then* TSAT_IS_2 (ϕ) *returns* TRUE *if ϕ satisfiable, and* FALSE *otherwise.*

Proof. By Theorem 5, since ϕ_0 is logically valid and therefore ϕ and $\phi_0 \wedge \phi$ are logically equivalent. □

EXAMPLE 8. Consider Example 1 once more. After the preprocessing step of TSAT_$IS_2(\phi_{Elf})$, the clauses

$$\neg(VPred = I_{RR}) \vee VPred < I_{RR} + 1$$

and

$$\neg(VenI' = VenI) \vee \neg(VenI' + 2 = VenI)$$

are added to ϕ_{Elf}. These added clauses allow for more pruning while descending the search tree.

```
function TSAT_$IS_2(\phi)$
1  let $\phi_0 := \top$
2  foreach unordered pair of SL-constraints $\{c_i, c_j\}$ in $\phi$
3          involving the same variables,
4    if SatCheck($c_i \wedge c_j$)=FALSE then $\phi_0 := \phi_0 \wedge (\neg c_i \vee \neg c_j)$
5    else if SatCheck($\neg c_i \wedge c_j$)=FALSE then $\phi_0 := \phi_0 \wedge (c_i \vee \neg c_j)$
6    else if SatCheck($c_i \wedge \neg c_j$)=FALSE then $\phi_0 := \phi_0 \wedge (\neg c_i \vee c_j)$
7    else if SatCheck($\neg c_i \wedge \neg c_j$)=FALSE then $\phi_0 := \phi_0 \wedge (c_i \vee c_j)$
8  return TSAT(($\phi_0 \wedge \phi$),$\top$)
```

Figure 3. IS_2 preprocessing.

Consider Example 6. In TSAT_*IS_2*(ϕ_{Elf}), choosing p_1 forces $VPred = I_{RR}$ by unit propagation; but now, thanks to the clause added by IS_2, this also forces $VPred < I_{RR} + 1$, which in turn forces p_2. TSAT (ϕ_{Elf}, $\top$) on the other hand, had to branch on p_2.

IS_2 is a simple way of guiding the generation phase by taking into account the structure of the constraints in the input formula. IS_2 has been proved to speed the search, especially on randomly generated problems such as the binary disjunctive temporal problems (DPTs), which are made of binary clauses containing constraints only (see Section 6.1). In that case, the effectiveness of the technique is dramatic, since adding more binary clauses, which is what IS_2 does, paves the way to detect and propagate more unit clauses once a literal has been selected by ChooseLiteral.

4.2. EARLY PRUNING

An alternative approach that aims at limiting the generation of unsatisfiable valuations is based on the idea of checking the valuations while they are generated by TSAT. This technique is called *early pruning* (EP) and relies on the fact that no unsatisfiable valuation can be extended into a satisfiable one by adding more constraints. EP can be readily incorporated in TAST, as shown in Figure 4.

THEOREM 9 (Soundness and completeness of TSAT_EP). *Let ϕ be a formula.* TSAT_EP (ϕ, $\top$) *returns true if ϕ is satisfiable, and* FALSE *otherwise.*

Proof. By Theorem 5 we know that TSAT is sound and complete. Now, first notice that TSAT_EP differs from TSAT only in that one more recursion base case, possibly returning FALSE, has been introduced at line 4. This fact ensures soundness of the function: if TSAT finds no model of ϕ, neither will TSAT_EP.

As far as completeness is concerned, assume by contradiction that a satisfiable valuation μ is found by TSAT, which is not found by TSAT_EP. By the above consideration, this means that a subset of μ, call it μ', must have been reached by

```
function TSAT_EP($\phi$,$\mu$)
1  if {} $\in \phi$ then return FALSE
2  if $\phi = \emptyset$ then return SatCheck($\mu$)
3  if {$l$} $\in \phi$ then return TSAT_EP(Simplify($l$,$\phi$),$\mu \wedge l$)
4  if SatCheck($\mu$)=FALSE then return FALSE
5  $l$ := ChooseLiteral($\phi$)
6  return TSAT_EP(Simplify($l$,$\phi$),$\mu \wedge l$) or
           TSAT_EP(Simplify($\bar{l}$,$\phi$),$\mu \wedge \bar{l}$)
```

Figure 4. TSAT with early pruning.

TSAT_EP and rejected. This means that μ' is unsatisfiable and μ, a superset of it, is satisfiable, which is contradictory. □

EXAMPLE 10. Consider Example 1, TSAT_EP as in the figure, and assume ChooseLiteral returns the first literal that appears in the formula. Then, TSAT_EP (ϕ_{Elf}, $\top$) picks and add to μ, in turn, p_1, $VPred = I_{RR}$ and $\neg p_2$. The last choice force $\neg(VPred < I_{RR} + 1)$ into μ by unit propagation, but clearly the valuation is now unsatisfiable. Therefore backtracking happens, and both $\neg(VPred < I_{RR} + 1)$ and $\neg p_2$ are removed from μ. ChooseLiteral then switches to p_2, and the algorithm goes on as in Example 6.

Notice that in this case TSAT, with the same ChooseLiteral, would have explored a totally useless portion of the search space, namely, checking all models prefixed with the unsatisfiable μ detected above by EP.

4.3. MODEL REDUCTION

A further optimization, called *model reduction*, is based on the observation that a valuation μ generated by TSAT can be *redundant*; that is, there might exist a valuation $\mu' \subset \mu$ that propositional entails the input formula. When this is the case, we can check the satisfiability of μ' instead μ. This has the following advantages:

1. if μ and μ' are either both satisfiable or both unsatisfiable, then the value returned by SatCheck is the same. However, checking the satisfiability of μ' can be easier if we use, for example, BF.
2. if μ is unsatisfiable, it may nevertheless be the case that μ' is satisfiable: in this case SatCheck(μ') returns TRUE, thereby pruning any further search.

Model reduction can be easily incorporated in TSAT as shown in Figure 5. The main difference with respect to TSAT is that the reduced valuation μ', rather than μ, is checked for satisfiability. It is assumed that ReduceModel(μ) returns a valuation $\mu' \subseteq \mu$ propositionally entailing the initial input formula.

THEOREM 11 (Soudness and completeness of TSAT_MR). *Let ϕ be a formula.* TSAT_MR (ϕ, $\top$) *returns* TRUE *if ϕ is satisfiable, and* FALSE *otherwise.*

Proof. It suffices to note that, since $\mu' \subseteq \mu$, there are three possible cases: both μ' and μ are satisfiable; both are unsatisfiable; or μ' is satisfiable, but μ is not. In the first two cases, SatCheck(ReduceModel(μ)) coincides with SatCheck(μ); in the third case, a satisfiable valuation propositionally entailing the input formula has been found, and the algorithm terminates. □

Here again it is important to check that, on average, the time spent in reducing the valuation does not overwhelm the advantage gained by reduc-

```
function TSAT_MR($\phi$,$\mu$)
1  if $\{\} \in \phi$ then return FALSE
2  if $\phi = \emptyset$ then return SatCheck(ReduceModel($\mu$))
3  if $\{l\} \in \phi$ then return TSAT_MR(Simplify($l$,$\phi$),$\mu \wedge l$)
4  $l$ := ChooseLiteral($\phi$)
5  return TSAT_MR(Simplify($l$,$\phi$),$\mu \wedge l$) or
          TSAT_MR(Simplify($\bar{l}$,$\phi$),$\mu \wedge \bar{l}$)
```

Figure 5. TSAT with model reduction.

ing it. So far, we have been experimenting with two techniques for reducing valuations:

Triggering: if μ contains a literal l that does not belong to any clause in the input formula ϕ, then μ propositionally entails ϕ if and only if $\mu \backslash \{l\}$ does; therefore l can be safely removed from μ. This technique, introduced in Wolfman and Weld (1999), is called *triggering*. Triggering has a linear cost in $|\mu|$ if realized, for example, via a simple table of the occurrences of literals in ϕ.
Minimization: a better idea is to remove as many redundant constraint literal l as possible. This can be done by recursively eliminating from μ one constraint literal l at a time such that for each clause C containing l, there exists another literal l' in $\mu \cup C$. Minimization can be done in linear time in the size of the input formula ϕ provided that a data structure associating to each literal l the clauses of ϕ whom l belongs to is available.

EXAMPLE 12. Consider again ϕ_{Elf}; in this case, a possible valuation found by TSAT_MR is $\mu = \{p_1, VPred = I_{RR}, p_2, VPred < I_{RR} + 1, VenI' = VenI, p_4, p_3, p_5, VenI' + 2 = VenI\}$. A reduced version of it, according to minimization, is $\mu' = \{p_1, VPred = I_{RR}, p_2\ VPred < I_{RR} + 1, p_3, p_4, p_5, VenI' + 2 = VenI\}$, obtained from μ by removing the constraint literal $VenI' = VenI$. Further, while μ is unsatisfiable, μ' is not.

Given a valuation μ it is important to notice that model reduction that is, ReducedModel(μ) in Figure 5, does not consider the set IS of clauses possibly added by IS_n to the input formula ϕ: these clauses are valid and thus do not need to be taken into account. Considering them would slow ReduceModel(μ) and, even worse, may partly shadow its effects. In fact if μ' and μ'' are the valuations returned by ReduceModel(μ) when considering ϕ and $\phi \cup IS$ respectively, we have that $\mu' \subseteq \mu''$. Furthermore, ReducedModel(μ) is not performed when the valuation μ does not propositionally entail the input formula ϕ, that is, when we are checking the satisfiability of a valuation because of early pruning. Indeed, with early pruning we hope to detect the unsatisfiability of μ in order to cut the search. On the other hand, it may be the case that μ' = ReducedModel(μ) in satisfiable while μ is not: in this case, considering μ' instead of μ would make vain early pruning.

4.4. BEST REASON DETECTION

So far, we have discussed how to extend an SAT solver in order to obtain a decider for SL, focusing in particular on SAT solvers based on DLL. Our motivation for this has been that most of the state-of-the-art complete SAT solvers are based on DLL. However, such solvers extend the basic DLL procedure in different ways in order to be more effective on different classes of problems. Broadly speaking, we can divide such solvers in two categories, following the distinction that is usually made in the SAT competition (Le Berre and Simon, 2003):

– those designed for real-world problems, e.g., zchaff (Moskewicz et al., 2001), the winner of the last SAT competition in this category. The features of these solvers are that they have a fast-to-compute heuristics, a simple but efficient pruning mechanism based on unit propagation, and a sophisticated backtracking mechanism based on back-jumping and learning (see Moskewicz et al., 2001).
– those designed for solving difficult either randomly generated or hand-made problems, for example, kcnfs (Dequen and Dubois, 2004) and March_eq (Heule and Maaren, 2005) the winners of the last SAT competition in these categories. These solvers have a complex-to-compute heuristics, sophisticated pruning mechanisms significantly extending unit-propagation, and a simple but efficient back-tracking mechanism without learning.

The modification needed in order to obtain a SAT-based solver for SL can be done along the lines so far outlined if we start from a solver without back-jumping and/or learning. Still, in case we want to use a backtracking schema based on learning, whenever FALSE is returned, a “reason” for the failure has to be computed. Intuitively, whenever we are backtracking from a valuation μ, a reason is a subset μ' of μ such that any valuation extending μ' will fail. While backtracking, these reasons μ' are used in order to back-jump over the literals which are not in μ'. Further, if the solver uses learning, the clause $\vee_{l \in \mu'} \bar{l}$ is (temporarily) added to the input set of clauses in order to avoid future explorations of valuations extending μ'.

Thus, in order to use SAT solvers with learning, it is not enough for SatCheck(μ) to return FALSE when μ is not satisfiable. Indeed, SatCheck(μ) must also compute a reason for such a failure, that is, an unsatisfiable subset μ' of μ. One such set is obviously μ itself. However, in order to try to maximize the advantages of learning, it is important that μ' be as “small” as possible with respect to some ordering relation on valuations. Let μ be an unsatisfiable valuation. We found it useful to consider the following forms of minimality:

– **Minimal reasons with respect to set inclusion.** An unsatisfiable valuation $\mu' \subseteq \mu$ is a *minimal reason for* μ *with respect to set inclusion* if and only if for all unsatisfiable valuations μ'' such that $\mu'' \subseteq \mu'$ we have that $\mu'' = \mu'$.

- **Reasons of minimal cardinality.** An unsatisfiable valuation $\mu' \subseteq \mu$ is a *reason for* μ *of minimal cardinality* if and only if for all unsatisfiable valuation $\mu'' \subseteq \mu$, we have that $|\mu'| \leq |\mu''|$.
- **Shallowest reasons.** Let $l_1, l_2, \ldots l_n$ ($n \geq 0$) be the literals in μ, listed according to the total order with which they have been assigned. Such a sequence induces a total order on the subsets of μ defined as follows: if μ' and μ'' are subsets of μ, then $\mu' \preceq \mu''$ if and only if for all literals $l_i \in \mu' \backslash \mu''$ there exists a literal $l_j \in \mu'' \backslash \mu'$ such that $i \leq j$. An unsatisfiable valuation $\mu' \subseteq \mu$ is the *shallowest reason for* μ if and only unsatisfiable valuation $\mu'' \subseteq \mu$, we have that $\mu' \preceq \mu''$.

Intuitively, there is no point in returning a reason that is not minimal under set inclusion: if we unnecessarily include a literal l in the reason, this may lead to branch on $\bar{l}$, and such a branch is bound to fail. Among the reasons that are minimal under set inclusion, those with minimal cardinality have the further advantage that, once added to the input formula because of learning, they prune a larger portion of the search space. Finally, while backtracking from a valuation μ, and even returning a reason μ' with minimal cardinality, it may still be the case that the next branch being explored is deemed to fail. In fact, μ may still contain a shallowest reason.

EXAMPLE 13. Consider Example 1 once again and assume that the heuristics is such that it first sets p_1 (forcing also $VPred = I_{RR}$ by unit propagation), then $\neg p_2$ (forcing $\neg(VPred < I_{RR} + 1)$), and then $VenI' = VenI$, p_3 and p_5 (this last one forcing also $VenI' + 2 = VenI$). The corresponding valuation $\{p_1, VPred = I_{RR}, \neg p_2, \neg(VPred < I_{RR} + 1), VenI' = VenI, p_3, p_5, VenI' + 2 = VenI\}$ propositionally entails ϕ_{Elf} but is unsatisfiable. The standard procedure detects that μ is unsatisfiable, but it backtracks only up to the choice of p_5, which is not involved in the unsatisfiability of μ; then a whole search branch is explored, which is totally useless, since the assignment still contains both $VPred = I_{RR}$ and $\neg(VPred < I_{RR} + 1)$, which are responsible of the contradiction. The same, even worse, goes for the choice of p_3.

On the other hand, if reason detection is enabled, upon detection of the unsatisfiability of μ, a reason is found, backtracking starts up to a point where the contradiction corresponding to the reason is solved. In our example, there are two minimal reasons, namely, $\xi = \{VPred = I_{RR}, \neg(Vpred < I_{RR} + 1)\}$ and $\xi' = \{VenI' = VenI, VenI' + 2 = VenI\}$. Both ξ and ξ' are minimal under set inclusion and of minimal cardinality. However, ξ is the shallowest. Indeed, if the reason is set to ξ, backtracking will stop at the choice point where $\neg p_2$ was chosen. Also notice that, assuming the reason being returned is ξ', backtracking will stop at the choice $VenI' = VenI$: however, the following search is bound to fail given that the valuation will still contain ξ.

The above example and discussion seems to point out that a reason of minimal size is better than a reason minimal under set inclusion, and that the shallowest reason is better than a reason of minimal size. Indeed, the shallowest reason tries to remove as soon as possible the unsatisfiability from the valuation built so far. However, despite the "smartness" of the reason being returned, there is no guarantee whatsoever that the tree being explored with a "smart" reason mechanism will be smaller than the tree explored with another reason mechanism. As Prosser (1993) pointed out, it may be the case that the *a priori* known fruitless exploration of a branch will lead to a failure and the discovery of a reason causing a long jump to the top of the search stack. To this end, a simple implementation of SatCheck(μ) returning, μ as reason whenever, μ is not satisfiable, can turn out to be more effective than other implementations, at least in some cases. However, trivially, a solver with back-jumping and/or learning can never explore more nodes than a solver with backtracking, assuming, for example, a static branching heuristics.

The first SAT-based solver for SL using a backtracking schema with learning has been proposed in Audemard et al. (2002). However, in that paper, there is no indication about how the reason is computed when SatCheck(μ) fails.

5. Satisfiability Checking

It is a well-known fact that BF can be used to check the satisfiability of a finite set Q of constraints of the form $x - y \leq c$; see, for example, Cormen et al. (2001). This is done by first building a *constraint graph for* Q, that is, a weighted directed graph whose nodes are the variables occurring in Q and having an *edge* from y to x of weight c for each constraint $x - y \leq c$ in Q. An extra node, the *source*, is also included and is linked to all the other nodes with edges of weight 0. BF is then used to solve the "single source shortest-paths" problem. The set of constraints Q is satisfiable if and only if the constraint graph for Q contains no *negative cycles*, that is, cycles with cumulative negative weight.

Here we show that satisfiability checking of a generic valuation μ can be done efficiently with BF. As a preliminary step, we turn μ into an equisatisfiable set $\mu^{\leq,<}$ whose literals are of the form $x - y \leq c$ or $x - y < c$. This can be done by deleting all the literals of the form p and $\neg p$ where p is a propositional letter and by replacing constraint literals

- $y - x \geq -c$, $\neg(y - x < -c)$, $\neg(x - y > c)$ with the logically equivalent constraint $x - y \leq c$, and
- $y - x > -c$, $\neg(y - x \leq -c)$, $\neg(x - y \geq c)$ with the logically equivalent constraint $x - y < c$.

A further step is needed to transform the valuation $\mu^{\leq,<}$ into an equisatisfiable set of constraints of the form $x - y \leq c$ whose satisfiability can be checked with

BF. If the domain of interpretation is $\mathbb{Z}$, this can be done by replacing in $\mu^{\leq,<}$ every constraint of the form $x - y < c$ with $x - y \leq c'$, where c' is the maximum integer strictly smaller than c. It is easy to see that the resulting set of constraint is satisfiable if and only if $\mu^{\leq,<}$ is. If the domain of interpretation is R, then we rely on the following result.

LEMMA 14. *Let Q and Q' be two finite sets of constraints of the form $x - y \leq c$ and $x - y < c$, respectively. Let n be the number of variables in Q'. Let p be the maximum number of digits appearing to the right of the decimal point in any numeric constant in $Q \cup Q'$. If C is $x - y < c$, let $C_{\leq}$ be $x - y \leq c - \frac{1}{10^p(n+1)}$. Finally, let $Q'_{\leq} = \{C_{\leq} : C \in Q'\}$.*

$Q \cup Q'$ is satisfiable in R if and only ij $Q \cup Q'_{\leq}$ is satisfiable in $\mathbb{R}$.

Proof. The right-to-left direction is trivial, and therefore here we focus on the left to right direction. In the following, if $Q'' \subseteq Q \cup Q'$ is a set of constraints, by $Q''_{\leq}$ we mean the set obtained from Q'' by replacing each constraint C of the form $x - y < c$ with $C_{\leq}$. Further, ϵ is $\frac{1}{10^p(n+1)}$.

We proceed by contradiction and assume that $Q \cup Q'$ is satisfiable while $Q \cup Q'_{\leq}$ is not. In this case, there exists a subset Q'' of $Q \cup Q'$ such that

- Q'' is satisfiable and $Q''_{\leq}$ is not,
- $Q''_{\leq}$ has the form $\{x_1 - x_2 \leq c_1 - e_1, x_2 - x_3 \leq c_2 - e_2, \ldots, x_m - x_i \leq c_m - e_m\}$, where each e_i is either 0 or ϵ, and
- in $Q''_{\leq}$ there are at least one and at most n constraints for which $e_i = \epsilon$, that is, $1 \leq |Q'' \cap Q'| \leq n$.

Q'' is satisfiable and $Q''_{\leq}$ unsatisfiable imply $\sum_{i=1}^{m} c_i > 0$ and $\sum_{i=1}^{m} (c_i - e_i) < 0$ respectively (notice that it cannot be the case that $\sum_{i=1}^{m} c_i = 0$ because $Q'' \cap Q' \neq \emptyset$ and Q'' has to be satisfiable by hypothesis). Since $\sum_{i=1}^{m} c_i > 0$, then $\sum_{i=1}^{m} c_i \geq \frac{1}{10^p}$. But then we have a contradiction, because

$$\begin{array}{ll}
\sum_{i=1}^{m} (c_i - e_i) & = \\
\sum_{i=1}^{m} c_i - \sum_{i=1}^{m} e_i & \geq \\
\sum_{i=1}^{m} c_i - n\epsilon & = \\
\sum_{i=1}^{m} c_i - \frac{1}{10^p} \frac{n}{n+1} & \geq \\
\frac{1}{10^p} - \frac{1}{10^p} \frac{n}{n+1} & > 0
\end{array}$$

□

Notice that the application of the above result requires, if the domain of interpretation is $\mathbb{R}$, to determine the values of n and p, which in turn depend on μ. The next result shows that the values for n and p can be computed beforehand and once and for all, on the basis of the input formula ϕ.

THEOREM 15. *Let ϕ be a formula with n variables. Let p be the maximum number of digits appearing to the right of the decimal point in any numeric constant in ϕ. Let μ be a valuation whose atoms occur in ϕ. The valuation μ is* *satisfiable in $\mathbb{R}$ if and only ij the valuation obtained from $\mu^{\leq,<}$ by replacing each constraint $x - y \leq c - \frac{1}{10^p(n+1)}$ is satisfiable in $\mathbb{R}$.*

Proof. Clearly, μ is satisfiable in $\mathbb{R}$ if and only if $\mu^{\leq,<}$ is satisfiable in $\mathbb{R}$. The thesis trivially follows from Lemma 14 once we observe that, given that the atoms in μ occur in ϕ,

– the number of variables in $\mu^{\leq,<}$ is less than or equal to n and
– the maximum number of digits appearing to the right of the decimal point in any of the numeric constants in $\mu^{\leq,<}$ is less than or equal to the maximum number of digits appearing to the right of the decimal point in any of the numeric constants in ϕ. $\square$

The above results allow us to use BF in order to check the satisfiability of any valuation. Given a valuation μ with n variables, BF runs in time $O(n \times |\mu|)$, and is the current best known method for this task (see Cormen et al., 2001). Further BF has the following advantages, in the case the valuation μ is unsatisfiable:

– each negative cycle in the constraint graph G corresponds to a minimal (with respect to set inclusion) unsatisfiable subset of μ, and
– assuming there is more than one negative cycle in G and that R is the corresponding set of reasons, it is easy to modify BF so to make it return a reason that is of minimal cardinality or the shallowest *among those in R* without modifying its overall complexity $O(n \times |\mu|)$.

6. Implementation and Experimental Analysis

We have implemented the techniques described in Sections 3–5 in a system called TSAT++. The system is based on a C++ implementation of an iterative version of the algorithm of Figure 2 featuring all optimizations presented in Section 4.

TSAT++ uses two distinct modules for the enumeration of valuations, μ propositionally entailing the input formula ϕ and for checking the satisfiability of μ. A detailed analysis of the architecture of TSAT++ is beyond the scope of this paper; the interested reader may refer to Armando et al. (2004).

In the current version, enumeration is done by a modified version of SIMO (Giunchiglia et al., 2003). SIMO features a number of SAT optimization techniques inspired by Chaff, among which are l-UIP learning, VSIDS heuristics, and two-literal watching (Moskewicz et al., 2001).

In order to assess the effectiveness of the optimizations described in Section 4, we have carried out a thorough experimental analysis using TSAT++ and TSAT++ plain, on a wide variety of publicly available random, hand-made, and real-world SL-formulas.[★] TSAT++ plain is the same as TSAT++ except that IS_2, early pruning, and model reduction are disabled while best reason detection is set so to return a reason minimal with respect to set inclusion. Further, in order to evaluate the effectiveness of our system, we have compared TSAT++ with a number of rival, publicly available, and state-of-the-art systems specifically designed for (a significant fragment of) SL or with a specialized satisfiability procedure for SL valuations.[★★] We have thus considered the system presented in Stergiou and Koubarakis (1998), which we will call SK; Tsat (Armando et al., 1999), the predecessor of TSAT++; CSPi (Oddi and Cesta, 2000); and Epilitis (Tsamardinos and Pollack, 2003). All these systems are restricted to DTPs (see Section 6.1). Moreover, we have considered SEP (Strichman et al., 2002) and MathSAT (Audemard et al., 2002). TSAT++ is as expressive as SEP and not comparable to MathSAT: while MathSAT allows for arbitrary linear constraints as atoms, it does not allow to consider the integers as domain of interpretation. After a first run, we have discarded SK, because it is clearly noncompetitive with respect to the others.

Each solver has been run on all the benchmarks it can deal with, not only on the benchmarks the solver was analyzed on by the authors. In particular, Epilitis can handle only DTPs with integer-valued variables; CSPi and TSAT can handle only DTPs with real-valued variables; Math-SAT can handle arbitrary SL-formulas with real-valued variables; SEP and TSAT++ can handle arbitrary SL-formulas with real- or integer-valued variables. Each solver has been run by using the settings or the version of the solver suggested by the authors for the *specific* class of problems. All the experiments have been run on a Linux box equipped with a Pentium IV 2.4 GHz processor and 1 GB of RAM. CPU time is measured in seconds; timeout has been set to 1,000 s.

6.1. DISJUNCTIVE TEMPORAL PROBLEMS

We start our analysis considering randomly generated DTPs as introduced in Stergiou and Koubarakis (1998) and since then used as a benchmark in (Armando et al., 1999; Oddi and Cesta, 2000: Audemard et al., 2002; Tsamardinos and Pollack, 2003). DTPs are randomly generated by fixing the number k of disjuncts per clause, the number n of arithmetic variables, and a

[★] The classification of the benchmarks in “tandem,” “handmade,” and “real-world” problems is borrowed from the SAT competition (Le Berre and Simon, 2003).

[★★] Notice that there exist other systems capable of handling SL, e.g., ICS (de Moura et al., 2004), CVC (Stump et al., 2002), CVC-Lite (Barrett and Berezin, 2004), Verifun (Flanagan et al., 2003). We did not include these solvers in our analysis since they are not tailored for SL. MathSAT has been included since it has a specialized satisfiability checker for SL based on BF.

positive integer L such that all the constants are taken in $[-L, L]$. Then, (1) the number of clauses m is increased in order to range from satisfiable to unsatisfiable instances, (2) for each tuple of values of the parameters, 100 instances are generated and then fed to the solvers, and (3) the median of the CPU time is plotted against the m/n ratio. The results for $k = 2$, $L = 100$, and $n = 35$ are given in Figure 6: plots (a) and (b) show the performance when the variables are real- and integer-valued respectively.

When $m/n \geq 6$, TSAT++ clearly outperforms the other systems, including TSAT++plain: in the peak region, the solver that is closer to TSAT++ in this domain, namely Epilitis, is a factor of 6 slower on 35 variables (Plot (b)). This is a very positive result, taking into account that Epilitis works only on DTP with $k = 2$, and it has been thoroughly tested and optimized on this type of problems (see Tsamardinos and Pollack (2003)). All the other systems are about two orders of magnitude slower than TSAT++ in the peak region. Even more important is the fact that the gap in performance between TSAT++ and the other systems increases with the number of variables (we have experimented with problems up to 50 variables). For this class of problems TSAT++ has been run with early pruning and preprocessing enabled, with the best reason detection optimization set to return shortest reason, and with model reduction disabled. The role of the optimizations is fundamental for the performance on this test set: TSAT++ is more than one order of magnitude faster than TSAT++plain in the peak region.

6.2. REAL-WORLD PROBLEMS

We have also carried out experiments on

1. the 40 post-office benchmarks introduced in Audemard et al. (2002), coming in four series (consisting of 7, 9, 11, and 13 instances, respectively) of increasing difficult. In these problems the domain of the interpretation is the set of real numbers.
2. the 16 hardware verification problems from Strichman et al. (2002), nine (resp. 7) of which are with real- (resp. integer-) valued variables.

The post-office benchmarks are bounded model checking problems for timed automata; the hardware verification suite includes scheduling, cache coherence protocol, load-store unit, and out-of-order execution problems. Considering the results of MathSAT, SEP, and TSAT++ on the post-office problems, our first observation is that SEP is not competitive on these problems: on 13 of the hardest instances, SEP had a segmentation fault in 11 cases, and on the other two hardest instances SEP is outperformed by different orders of magnitude by TSAT++ and MathSAT. Our second observation is that TSAT++ (with IS_2 pre-better than MathSAT up to a factor of 6 on *each single instance*: this is particularly remarkable given that the authors have customized a version of

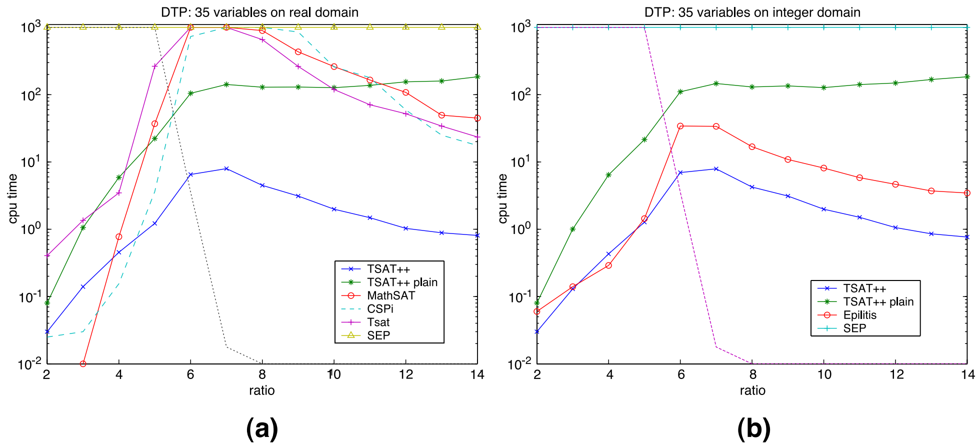

Figure 6. Performance on (a) randomly generated DTPs with 35 real valued variables and on (b) randomly generated DTPs with 35 integer-valued variables. The *dotted plot* indicates satisfiability percentage both in (a) and in (b).

MathSAT explicitly for this kind of problems.★ Considering the hardware verification problems, all of them are easy to solve (i.e., in less than 3 s each) for all the three solvers, except for SEP that timeouts on one instance. Of the nine (resp. 16) runs of MathSAT (resp. SEP and TSAT++), only three take more than 0.1 s. These observations are confirmed by Figure 7, which gives the overall picture of the results for MathSAT, SEP, and TSAT++ on the 49 instances with real valued variables: the x-axis is the number of instances solved by each solver within the CPU time specified on the y-axis. The plot also shows that TSAT++ plain can be faster than TSAT++ on the easy instances, that is, those requiring less than 1 s to be solved. For such problems, the overhead of the optimizations (and in particular of the preprocessing) outweighs the benefits.

6.3. HAND-MADE PROBLEMS

Finally, we have considered the “hand-made” diamond problems from Strichman et al. (2002). A diamond problem is a formula ϕ that depends on a parameter $K > 0$ and such that there exists a number of unsatisfiable valuations propositionally entailing ϕ that is exponential in K. Moreover, hard instances having a single satisfiable valuation propositionally entailing them can be generated. A second parameter T is also used and it affects the number of variables and the size of the problem. Variables range over the reals.

★ As indicated by the authors, we have used this customized version of MathSAT on this class of problems.

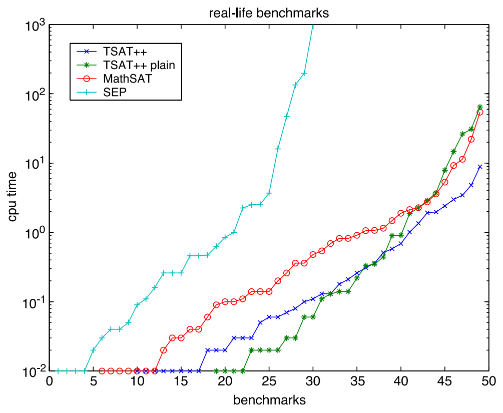

Figure 7. Performance on real-problems.

Table I shows comparative results on the diamond problems for various settings of K and T. In particular, we considered all the settings corresponding to nontrivially solvable instances reported in (Strichman et al., 2002). The third column denotes whether the problem has a unique valuation propositionally entailing it; the remaining columns show CPU times for TSAT++, TSAT++ plain, MathSAT, and SEP. For this class of problems TSAT++ has been run with best-reason detection set to shortest reason, and with model reduction. The experimental results clearly show that TSAT++ performs best, often by orders of magnitude. Instances with a unique solution are more difficult than nonunique ones, as expected, except for SEP.★

For this test set, it is of fundamental importance the model reduction optimization: without it, TSAT++ performance is significantly worse, up to the point that problems that are solved in 1 s by TSAT++ are not solved without model reduction within the time limit.

7. Related Work

Several systems tailored for SL, employing different approaches and techniques, have been built and tested over the years. We now give an overview of them, highlighting the pros and cons of each one and chronologically reviewing the techniques introduced by each one. SK (Stergiou and Koubarakis, 1998); Stergiou

★ Following a suggestion by Offer Strichman, we have also tried SEP with an option that disables the use of a specialized data structure called "conjunction matrix" (Strichman et al., 2002). This can have a dramatic impact on SEP: some problems that are solved with conjunction matrix within the time limit are not solved without, and vice versa.

Table I. Diamond problems: “TIME” indicates that the solver does not solve the instance within the time limit.

K	T	Unique	TSAT++	TSAT++ plain	MathSAT	SEP
50	4	NO	0.00	0.02	0.05	0.12
50	4	YES	0.01	0.14	TIME	0.07
100	5	NO	0.01	0.11	0.61	1.18
100	5	YES	0.04	7.57	TIME	0.17
250	5	NO	0.08	0.76	5.40	52.20
250	5	YES	0.21	194.99	TIME	0.77
500	5	NO	0.29	4.46	21.22	742.99
500	5	YES	1.05	TIME	TIME	4.85
1000	5	NO	1.07	22.3	–	TIME
1000	5	YES	6.45	TIME	–	22.53
2000	5	NO	3.76	94.23	–	–
2000	5	YES	29.90	TIME	–	–

and Koubarakis, 2000). The procedure SK has been the first dealing with a significant fragment of SL. Its main features are the combined usage of *forward-checking*, back-jumping, and the minimum remaining value heuristic (MRV). Forward-checking works by checking whether the valuation built so far entails either a literal or its negation, form each literal not yet in the valuation. This actually reduces the search space, at the price of performing a potentially large number of useless satisfiability. SK is also able to detect conflict sets and to improve on backtracking via a technique similar to back-jumping. MRV is used to choose literals that appear in disjunctions with the smallest number of unassigned disjuncts: if there is a unit clause, the literal in it will be selected by MRV and then propagated, thus mimicking unit propagation.

The main difference between SK and SAT-based procedures lies in the way valuations propositionally entailing the input formula are searched. In fact, SK is based on *syntatic branching*: given a disjunction $l \vee l'$, first l is added to the current valuation, and, upon failure, l' is considered. As explained below, this type of search may lead to the exploration of search space already explored.

Tsat (Armando et al., 1999). Tsat was the first application of the SAT-based approach to SL. The system employs a branching schema now known as *semantic branching*. Unlike syntactic branching, semantic branching selects a not yet assigned literal l, and considers in turn the case in which l is true and the case in which l is false. Notice that in the second case, the conjunction of $\bar{l}$ with $(l \vee l')$ forces the assignment of l' by unit propagation: as already observed in D’Agostino (1992), syntactic branching may lead to redundant exploration of parts of the search space, which semantic branching avoids. The following example, adapted from Armando et al. (1999), clearly illustrates this issue.

EXAMPLE 16. Let ϕ be a formula including the following clauses:

$$\begin{array}{l} x_1 - x_2 \leq 3 \vee x_7 - x_8 \leq 20 \\ x_1 - x_3 \leq 4 \vee x_4 - x_3 \leq -2 \\ x_2 - x_4 \leq 2 \vee x_3 - x_2 \leq 1 \\ \qquad\qquad \vdots \end{array}$$

Let $\phi(i, j)$ denote the jth disjunct of the ith disjunction displayed in ϕ; for example, $\phi(1, 2)$ is $x_7 - x_8 \leq 20$. Assume that the dots stand for further (possibly many) unspecified clauses such that no satisfiable extension of the valuation $\{\phi(1, 1), \phi(2, 1)\}$ exists.

Consider the behavior of syntactic *versus* semantic branching when $\{\phi(1, 1), \phi(2, 1)\}$ is the valuation built so far. Since no satisfiable extension of it exists, after some search, failure is necessarily detected; both procedures backtrack and remove $\phi(2, 1)$ from the current valuation.

Now syntactic branching goes on with the valuation $\{\phi(1, 1), \phi(2, 2)\}$, whereas semantic branching proceeds with $\{\phi(1, 1), \neg\phi(2, 1)\}$, which leads immediately, via unit propagation, to $\{\phi(1, 1), \neg\phi(2, 1), \phi(2, 2)\}$.

Working with the latter valuation rather than with the former may lead to considerable savings: assume that both procedures extend the valuation with $\phi(3, 1)$; since $\{\phi(1, 1), \neg\phi(2, 1), \phi(2, 2)\ \phi(3, 1)\}$ is unsatisfiable, semantic branching immediately backtracks and considers $\phi(3, 2)$, whereas syntactic branching may waste a big amount of resources in the vain attempt of finding a satisfiable extension of $\{\phi(1, 1), \phi(2, 2), \phi(3, 1)\}$.

Semantic branching was shown in Armando et al. (1999) to dramatically improve the performance with respect to SK, up to one order of magnitude on randomly generated binary DTPs.

In Tsat, also IS_2 was introduced, gaining to the system another order of magnitude in performance—this despite the fact that, to enumerate valuations, Tsat adapted a rather simple SAT solver, due to Böhm (Böhm and Speckenmeyer 1996), which did not employ any modern optimization such as back-jumping and learning. Satisfiability checking used *lp_solve* v2.2 (Berkelaar, 1997), which provided a free implementation of the Simplex method.

CSPi (Oddi and Cesta, 2000). CSPi features an essentially CSP-based solution schema, implementing an efficient incremental procedure for forward-checking. Semantic branching is used, showing results that are better than Tsat on small instances, and comparable on bigger ones. Notice that performance, up to (Oddi and Cesta, 2000), was measured in terms of how many calls to the satisfiability check function were done, rather than CPU time.

MathSAT (Audemard et al., 2002). MathSAT uses SIM (Giunchiglia et al. 2001) as enumerator and a hierarchical satisfiability checker employing – in this order – equality reasoning, BF for SL-constraints, the Simplex method for full linear arithmetic, and inequalities reasoning. The simplest solver is chosen on-the-fly, thereby obtaining both expressivity and efficiency at the same time. MathSAT also introduces a number of optimizations, among which are preprocessing based upon syntactic equivalence, enhanced early pruning, that is, early pruning conditioned upon a heuristic function, and back-jumping/learning based upon reason detection. Also, a form of model reduction is used, based upon triggering. On randomly generated binary DTPs, MathSAT improves the performance over Tsat in terms of CPU time. However, the gap between the two solvers decreases as the number of variables increases.

Epilitis (Tsamardinos and Pollack, 2003). Epilitis is, so far, the last CSP-based system. Epilitis is restricted to binary DTPs. It uses semantic branching, incremental forward checking, a MRV heuristics, and *size-bounded learning* of size n (Bayardo and Miranker, 1996). This means that conflict clauses are retrieved and stored only if they contain less than n literals (in practice, $n = 10$ is used). Once stored, a clause is never forgotten. On randomly generated binary DTPs, Epilitis shows significantly better performance than Tsat in terms of CPU time, of up to one order of magnitude.

SEP (Strichman et al., 2002). SEP is a back-end to the UCLID verification tool (Lahiri et al., 2002), employing the so-called eager variant of the SAT-based approach. Given a formula ϕ, rather than enumerating valuations and checking them for satisfiability, SEP builds a propositional formula ϕ' whose satisfying valuations are ensured to correspond to satisfiable valuations of ϕ. The current version of SEP uses Chaff to find valuations satisfying ϕ'. To the best of our knowledge, SEP is so far the only solver using the eager SAT-based approach to SL. SEP suffers from the fact that the size of ϕ' can be exponential in the size of ϕ. On the other hand, as reported in Strichman et al. (2002), if SEP can get past the encoding phase, the problem is easy to solve for Chaff.

8. Conclusions

In this paper we have focused on the problem of building efficient SAT-based decision procedures for SL. We have presented the basic procedure from Armando et al. (1999) along with some key optimizations. We have also shown how it is possible to check the satisfiability of valuations involving constraints of the form $x - y < c$ using BF. An extensive comparative experimental analysis shows that our solver TSAT++, built along the lines described in this paper, is currently the state of the art on various classes of problems, including randomly generated, hand-made, and real-world instances. We believe that the techniques described in this paper can be fruitfully extended to other (more expressive) logics than SL.

The benchmark problems used for the experiments presented in this paper and the executable of TSAT++ are publicly available at the URL http://www.ai.dist.unige.it/Tsat.

Acknowledgements

We acknowledge Massimo Idini's work on the satisfiability checking module. Mauro Di Manzo is thanked for the many fruitful discussions on the subject of this paper. Moreover, the authors of the solvers we have compared have helped us a lot: Gilles Audemard, Angelo Oddi, Ofer Strichman, Ioannis Tsamardinos. Sergey Berezin and Leonardo De Moura are thanked for discussions related to the topic of this paper. Sanjit Seshia and the UCLID group provided us with a lot of interesting problems. We are partially supported by MIUR.

References

Armando, A. and Giunchiglia, E. (1993) Embedding complex decision procedures inside an interactive theorem prover, *Ann. Math. Artif. Intell.* **8**(3–4), 475–502.

Armando, A., Castellini, C. and Giunchiglia, E. (1999) SAT-based procedures for temporal reasoning, in S. Biundo and M. Fox (eds.), *Proceedings of the 5th European Conferevace on Planning (Durham, UK)*, Vol. 1809 of *Lecture Notes in Computer Science*, Springer, pp. 97–108.

Armando, A., Castellini, C., Giunchiglia, E., Idini, M. and Maratea, M. (2004) TSAT++: an open platform for satisfiability modulo theories, in *Proceedings of PDPAR, Pragmatics of Decision Procedures in Automated Reasoning, Cork (Ireland)*, Vol. 125, Issue 3 of *ENTCS*, Elsevier, pp. 25–36.

Armando, A., Castellini, C., Giunchiglia, E. and Maratea, M. (2005a) A SAT-based decision procedure for the boolean combination of difference constraints, in *Proceedings of SAT, International Conference on Theory and Applications of Satisfiability Testing, Vancouver (Canada)*, Vol. 3542 of *LNCS*, Springer, pp. 16–29.

Armando, A., Castellini, C., Giunchiglia, E., Giunchiglia, F. and Tacchella, A. (2005b) SAT-based decision procedures for automated reasoning: a unifying perspective, in *Mechanizing Mathematical Reasoning: Essays in Honor of Jrg H. Siekmann on the Occasion of His 60th Birthday*, Vol. 2605 of *Lecture Notes in Computer Science*, Springer.

Audemard, G., Bertoli, P., Cimatti, A., Kornilowicz, A. and Sebastiani, R. (2002) A SAT based approach for solving formulas over Boolean and linear mathematical propositions, in A. Voronkov (ed.), *Automated Deduction – CADE-18*, Vol. 2392 of *Lecture Notes in Computer Science*, Springer, pp. 195–210.

Barrett, C. W. and Berezin, S. (2004) CVC Lite: a new implementation of the cooperating validity checker category B, in *16th International Conference on Computer Aided Verification (CAV'04)*, Vol. 3114, Springer, pp. 515–518.

Bayardo, Jr., R. J. and Miranker, D. P. (1996) A complexity analysis of space-bounded learning algorithms for the constraint satisfaction problem, in *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, Menlo Park, AAAI/MIT, pp. 298–304.

Berkelaar, M. (1997) The *lp_solve* Solver for Mixed Integer-Linear Programming. Version 2.2. Available at http://www.cs.sunysb.edu/~algorith/implement/lpsolve/implement.shtml.

Böhm, M. and Speckenmeyer, E. (1996) A fast parallel SAT-solver – efficient workload balancing, *Ann. Math. Artif. Intell.* **17**, 381–400.

Bryant, R. E., Lahiri, S. K. and Seshia, S. A. (2002) Deciding CLU logic formulas via Boolean and pseudo-Boolean encodings, in *Proceedings of International Workshop on Constraints in Formal Verification*. Associated with International Conference on Principles and Practice of Constraint Programming, Ithaca, New York (USA).

Castellini, C., Giunchiglia, E. and Tacchella, A. (2003) SAT-based planning in complex domains: concurrency, constraints and nondeterminism, *Artif. Intell.* **147**, 85–117.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C. (2001) *Introduction to Algorithms*, MIT.

Cotton, S., Asarin, E., Maler, O. and Niebert, P. (2004) Some progress in satisfiability checking for difference logic, in *Joint International Conferences on Formal Modelling and Analysis of Timed Systems (FORMATS) and Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT)*, Vol. 3253 of *Lecture Notes in Computer Science*, Springer, pp. 263–276.

D'Agostino, M. (1992) Are tableaux an improvement on truth-tables? *J. Logic, Lang. Inf.* **1**, 235–252.

Davis, M., Logemann, G. and Loveland, D. (1962) A machine program for theorem proving, *Journal of the ACM* **5**(7).

de Moura, L., Ruess, H., Shankar, N. and Rushby, J. (2004) The ICS decision procedures for embedded deduction, in *Proceedings of IJCAR, International Joint Conference on Automated Reasoning*, Cork, Ireland.

Dechter, R., Meiri, I. and Pearl, J. (1989) Temporal constraint networks, in H. J. L. R. J. Brachman and R. Reiter (eds.), *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning*, Toronto, Canada, Morgan Kaufmann, pp. 83–93.

Dequen, G. and Dubois, O. (2004) kcnfs: an efficient solver for random K-Sat formulae, in E. Giunchiglia and A. Taicchella (eds.), *6th International Conference on Theory an Applications of Satisfiability Testing. Selected Revised Papers*, Vol. 2919 of *Lecture Notes in Computer Science*, Springer, pp. 486–501.

Flanagan, C., Joshi, R., Ou, X. and Saxe, J. B. (2003) Theorem proving using lazy proof explication, in *15th International Conference on Computer Aided Verification (CAV'03)*, Vol. 2725, Springer, pp. 355–367.

Gent, I., Maaren, H. V. and Walsh, T. (eds.) (2000) *SAT2000. Highlights of Satisfiability Research in the Year 2000*, IOS.

Giunchiglia, F. and Sebastiani, R. (1996) Building decision procedures for modal logics from propositional decision procedures – the case study of modal K, in *Proc. CADE-96*, New Brunswick, New Jersey, USA, Springer.

Giunchiglia, E., Maratea, M., Tacchella, A. and Zambonin, D. (2001) Evaluating search heuristics and optimization techniques in propositional satisfiability, in *Automated Reasoning, First International Joint Conference (IJCAR)*, Vol. 2083 of *Lecture Notes an Computer Science*, Springer, pp. 347–363.

Giunchiglia, E., Giunchiglia, F. and Tacchella, A. (2002) SAT-based decision procedures for classical modal logics, *J. Autom. Reason.* **28**, 143–171. Reprinted in (Gent et al., 2000).

Giunchiglia, E., Maratea, M. and Tacchella, A. (2003) (In)Effectiveness of look-ahead techniques in a modern SAT solver, in *Principles and Practice of Constraint Programming (CP)*, Vol. 2833 of *Lecture Notes in Computer Science*, Springer, pp. 842–846.

Heule, M. and Maaren, H. V. (2005) March_eq: implementing additional reasoning into an efficient look-ahead SAT solver, in *8th International Conference on Theory an Applications of Satisfiability Testing*, Vol. 3542 of *LNCS*, Springer, pp. 345–353.

Lahiri, S. K., Seshia, S. A. and Bryant, B. (2002) Modeling and verification of out-of-order microprocessors in UCLID, *Lect. Notes Comput. Sci.* **2517**, 142–155.

Le Berre, D. and Simon, L. (2003) The essentials of the SAT'03 competition, in *Proceedings of the 6th International Conference on the Theory and Applications of Satisfiability Testing (SAT'03). Selected revised papers*, Vol. 2919 of *LNCS*.

Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L. and Malik, S. (2001) Chaff: engineering an efficient SAT solver, in *Proceedings of the 38th Design Automation Conference (DAC'01)*.

Oddi, A. and Cesta, A. (2000) Incremental forward checking for the disjunctive temporal problem, in *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI-2000)*, Berlin, pp. 108–112.

Plaisted, D. and Greenbaum, S. (1986) A structure-preserving clause form translation, *J. Symb. Comput.* **2**, 293–304.

Pratt, V. R. (1977) Two easy theories whose combination is hard, Technical report, Massachusetts Institute of Technology.

Prosser, P. (1993) Domain filtering can degrade intelligent backjumping search, in *Proc. IJCAI*, pp. 262–267.

Siekmann, J. and Wrightson, G. (eds.) (1983) *Automation of Reasoning: Classical Papers in Computational Logic 1967–1970*, Vol. 1–2, Springer.

Stergiou, K. and Koubarakis, M. (1998) Backtracking algorithms for disjunctions of temporal constraints, in *Proceedings of AAAI/IAAI, Madison, WI (USA)*, pp. 248–253.

Stergiou, K. and Koubarakis, M. (2000) Backtracking algorithms for disjunctions of temporal constraints, *Artif. Intell.* **120**(1), 81–117.

Strichman, O., Seshia, S. A. and Bryant, R. E. (2002) Deciding separation formulas with SAT, *Lect. Notes Comput. Sci.* **2404**, 209–222.

Stump, A., Barrett, C. W. and Dill, D. L. (2002) CVC: a cooperating validity checker, in J. C. Godskesen (ed.), *Proceedings of the International Conference on Computer-Aided Verification*.

Tsamardinos, I. and Pollack, M. (2003) Efficient solution techniques for disjunctive temporal reasoning problems, *Artif. Intell.* **151**, 43–89.

Tseitin, G. (1970) On the complexity of proofs in propositional logics, *Semin. Mat.* **8**. Reprinted in (Siekmann and Wrightson, 1983).

Wolfman, S. and Weld, D. (1999) The LPSAT-engine and its application to resource planning, in *Proceedings IJCAI-99*.