SAT-Based Decision Procedures for Automated Reasoning: a Unifying Perspective

Alessandro Armando[1], Claudio Castellini[1], Enrico Giunchiglia[1], Fausto Giunchiglia[2,3], and Armando Tacchella[1]

[1] DIST, Università di Genova
viale Causa 13, 16145 Genova – Italy
{armando,drwho,enrico,tac}@dist.unige.it
[2] DICT, Università di Trento
Povo, 38100 Trento – Italy
fausto@cs.unitn.it
[3] ITC-IRST
via Sommarive 18, 38050 Trento – Italy

Abstract. Propositional reasoning (SAT) is an essential part of many reasoning tasks. Many problems in computer science can be compiled to SAT and then effectively decided using state-of-the-art solvers. Alternatively, if reduction to SAT is not feasible, the ideas and technology of state-of-the-art SAT solvers can be useful in deciding the propositional component of the reasoning task being considered. This last approach has been used in different contexts by different authors, many times by authors of this paper. Because of the essential role played by the SAT solver, these decision procedures have been called "SAT-based". SAT-based decision procedures have been proposed for various logics, but also in other areas such as planning. In this paper we present a unifying perspective on the various SAT-based approaches to these different reasoning tasks.

1 Introduction

Propositional reasoning (SAT) is an essential part of many reasoning tasks. Many problems in computer science can be compiled to SAT and then effectively solved using state-of-the-art solvers, see, e.g., [Kautz and Selman, 1992, Kautz and Selman, 1996, Biere *et al.*, 1999]. Alternatively, if reduction to SAT is not feasible, the ideas and technology of state-of-the-art SAT solvers can be useful in deciding the propositional component of the reasoning task being considered. This last approach has been used in different contexts by different authors, many times by authors of this paper. Because of the essential role played by the SAT solver, it has been called "SAT-based" in [Giunchiglia and Sebastiani, 1996b]. That paper is about decision procedures for modal logics. The same topic is dealt with in [Giunchiglia and Sebastiani, 1996a, Giunchiglia *et al.*, 1998, Giunchiglia *et al.*, 2000b]. SAT-based decision procedures for decidable fragments of first order logic are presented in [Armando and Giunchiglia, 1989, Armando and

Giunchiglia, 1993]. Finally, SAT-based decision procedures have been proposed in temporal reasoning [Armando *et al.*, 1999] and planning [Giunchiglia *et al.*, 2000b, Giunchiglia *et al.*, 2001, Wolfman and Weld, 1999].

In this paper, we present a unifying perspective on the various SAT-based approaches to the different reasoning tasks previously considered. In particular, in Section 2 we present the common ideas of all these various works. Then in Section 3 we show some optimizations to the basic procedures described in Section 2. Finally, we review the cited works on SAT-based decision procedures, presenting them in the context of the unifying framework previously introduced. We conclude the paper in Section 5, with some final remarks.

2 SAT-Based Decision Procedures: a Unifying Perspective

In the following, we consider an arbitrary logic characterized as a pair $\mathcal{L} = \langle L, T \rangle$ where

- L is the *language*, i.e., a set of formulae in some formal language which includes the standard propositional connectives, i.e., the unary connective $\neg$, the binary connective $\supset$, and the k-ary ($k \geq 0$) connectives $\vee, \wedge, \equiv$; and
- T is a theory, i.e. a subset of the language closed under propositional reasoning and the additional rules specific of the logic at hand.

We also assume that the logic is consistent, i.e., that for any formula φ in the language L, it is not the case that both φ and $\neg\varphi$ belong to T; and decidable in the standard sense, see, e.g., [Dreben and Goldfarb, 1979].

In this paper we focus on the following problem:

> Given a logic $\mathcal{L} = \langle L, T \rangle$ and a formula $\varphi \in L$, is the formula $\mathcal{L}$-consistent? That is, does $\neg\varphi$ belong to $L \setminus T$?

The decidability of the logic ensures that the task of deciding the $\mathcal{L}$-consistency of any given formula in L can be accomplished.

In the following, an *atom* of L is a formula whose main symbol is not a propositional connective, i.e., is not in $\{\neg, \supset, \vee, \wedge, \equiv\}$. A *literal* is an atom or the negation of an atom. An *assignment* μ is a finite conjunction of literals such that it is not the case that both ψ and $\neg\psi$ are conjuncts of μ. An assignment μ *satisfies* a formula φ if the formula $\mu \supset \varphi$ can be proved by propositional reasoning. We write $\mu(A) = \top$ as an abbreviation for "A is a conjunct of μ", and $\mu(A) = \bot$ as an abbreviation for "$\neg A$ is a conjunct of μ".

The basic idea behind the SAT-based approach to determine the $\mathcal{L}$-consistency of a formula φ is very simple and consists of the following two steps:

1. *generate* a (possibly partial) assignment which propositionally satisfies the formula, and then
2. *test* that the generated assignment is indeed consistent w.r.t. $\mathcal{L}$.

Given that the generation step involves propositional reasoning only, it is possible to use state-of-the-art SAT solvers for generating assignments. Because of this, we inherit the many optimizations and heuristic strategies (improving the average case behavior) which are implemented in current SAT solvers. Notice that in principle any set S of assignments satisfying φ can be generated and tested. However, in all the SAT-based procedures, the main focus has been on the generation of complete and irredundant sets of assignments. A set S of assignments is

- *complete* for a formula φ, if φ is propositionally logically equivalent to the disjunction of the assignments in S; and
- *irredundant* for a formula φ, if for any assignment $\mu \in S$ we have that $S \setminus \{\mu\}$ is not complete.

Completeness is required to have correct and complete SAT-based procedures. Irredundancy is very important too, as it improves efficiency in many cases.

Several algorithms and techniques have been proposed to solve the satisfiability problem in propositional logic (see, e.g., [Gu *et al.*, 1997]). Among this variety of approaches we have chosen the Davis-Logemann-Loveland (DLL) method [Davis *et al.*, 1962] to develop our decision procedures. The reasons for this choice are manyfold:

- DLL is a simple and elegant algorithm whose implemented variants proved to be very effective in attacking hard SAT instances;
- since most state-of-the-art solvers are based on DLL, there is a lot of knowledge on data structures and algorithms that we can inherit in our setting for free;
- even if DLL is usually tuned to find a single satisfying assignment, it is rather easy to modify it to generate a complete and irredundant set of assignments.

Examples of state-of-the-art SAT solvers based on DLL are BÖHM [Buro and Buning, 1992, Böhm and Speckenmeyer, 1996], SATZ [Li and Anbulagan, 1997], RELSAT v2.0 [Bayardo, Jr. and Schrag, 1997], SATO v3.2 [Zhang, 1997], SIM [Giunchiglia *et al.*, 2001], and—more recently—CHAFF [Moskewicz *et al.*, 2001].

A basic DLL implementation for SAT-based reasoning in outlined in Figure 1. The conventions that we use to present the algorithms are those of [Cormen *et al.*, 1998], described at pages 4 and 5. In particular, variables (e.g. Γ, S, l) are treated as pointers to the data structures representing the corresponding entities. If a pointer does not refer to any object, we give it the special value NIL. Stacks are considered a primitive data type and are accessed with the usual constant-time primitives PUSH and POP, while the function EMPTY builds and returns an empty stack. The primitive ASSIGN(Γ, l) returns the set of clauses Γ minus all the clauses in which literal l occurs, and with all the occurrences of its negation, $\bar{l}$, removed. Finally, we assume that T, F, LA, LB, and HR are five pairwise distinct constants, each one being distinct from NIL. In particular, T and F represent logical truth and falsehood, respectively.

Function DLL-SOLVE takes a formula φ as input and returns T (F) exactly when φ is satisfiable (unsatisfiable, resp.). Function DLL-SOLVE converts φ into

```
DLL-Solve(φ)
  1 Γ ← CNF-Convert(φ)
  2 S ← Empty( )
  3 return DLL-Solve-CNF(Γ, S)

DLL-Solve-CNF(Γ, S)
  1 next ← LA
  2 repeat
  3   case next of
  4     LA : next ← Look-Ahead(Γ, S)
  5     HR : next ← Heuristic(Γ, S)
  6     LB : next ← Look-Back(Γ, S)
  7 until next ∈ {T, F}
  8 return next

Look-Ahead(Γ, S)
  1 for each l deduced from Γ do
  2   S ← Push(S, ⟨Γ, l, LA⟩)
  3   Γ ← Assign(Γ, l)
  4   if an empty clause is in Γ then
  5     return LB
  6 if Γ is not empty then
  7   return HR
  8 else
  9   return Is-Consistent(S)

Is-Consistent(S)
  1 μ ← Assignment-In(S)
  2 if L-Consist(μ) = T then
  3   return T
  4 else
  5   return LB

Heuristic(Γ, S)
  1 Choose a literal l in Γ
  2 S ← Push(S, ⟨Γ, l, HR⟩)
  3 Γ ← Assign(Γ, l)
  4 return LA

Look-Back(Γ, S)
  1 repeat
  2   ⟨Γ, l, r⟩ ← Pop(S)
  3 until r = HR
  4 if length[S] = 0 then
  5   return F
  6 else
  7   S ← Push(S, ⟨Γ, l̄, LB⟩)
  8   Γ ← Assign(Γ, l̄)
  9   return LA
```

Fig. 1. Implementation of the DLL method for SAT-based reasoning.

an equi-satisfiable clausal normal form Γ (line 1) using the function CNF-Convert. We do not discuss here CNF-Convert and the issues related to conversion in clausal normal form. More details can be found in [Giunchiglia *et al.*, 2000b]. Here it is sufficient to say that the conversion can be done in such a way that $|\Gamma|$ is in $O(|\varphi|)$ where $|\varphi|$ is the size, i.e. the number of symbols of φ. Function DLL-Solve also initializes the search stack S (line 2) and then calls DLL-Solve-CNF to determine the satisfiability of Γ. The elements of the search stack are triples of the form $\langle \Gamma, l, flag \rangle$, where Γ is a set of clauses, l a literal, and $flag \in \{\textsc{t}, \textsc{f}, \textsc{la}, \textsc{lb}, \textsc{hr}\}$. Function Assignment-In takes as input the search stack S and returns a conjunction of the literals stored in it, i.e. $\bigwedge_{\langle -,l,- \rangle \in S} l$. Function DLL-Solve-CNF solves Γ by iteratively applying one of the following steps:

Look-Ahead to deduce new truth assignments from Γ. Look-Ahead keeps simplifying Γ (for instance, by exploiting the unit clauses in it) until an inconsistency arises or a fix point is reached. In case of inconsistency (line 4), the return value of Look-Ahead is LB, meaning that the main loop has to call Look-Back. In case of a fix point, we have two possibilities: if there are still clauses in Γ, then the return value is HR; if Γ is empty, then all the clauses have been satisfied and the function IS-CONSISTENT is invoked.

HEURISTIC to decide the next truth assignment and to enforce it; the decision is taken by considering Γ and/or possibly some Γ' obtained from Γ by tentatively assigning truth values to literals.

LOOK-BACK to undo truth assignments, until a point from which the search can continue without loosing solutions. If there is no such a point (i.e., the search tree is complete), then LOOK-BACK concludes that the initial formula cannot be satisfied.

Since Γ and S are pointers, the actions LOOK-AHEAD, LOOK-BACK, and HEURISTIC all update the input formula and stack of DLL-SOLVE-CNF in various ways during the **repeat** ... **until** loop (lines 2-7). Each action modifies Γ and S and returns the next action to be taken which is stored in *next* (lines 4-6 in the program). LA, LB, HR, T, and F are the possible values taken by *next*, meaning that the next action must be LOOK-AHEAD, LOOK-BACK, HEURISTIC, or that of stopping the loop respectively. In the latter case, if *next* is assigned T then Γ is satisfied, otherwise (i.e. if *next* is assigned F) Γ is unsatisfiable. When Γ is satisfiable, the corresponding satisfying truth assignment μ can be extracted from S. This task is accomplished in Figure 1 by IS-CONSISTENT which extracts μ from S (lines 1-3) and then calls the consistency test L-CONSIST specific for the logic at hand. Notice that in the case of mere propositional satisfiability IS-CONSISTENT simply returns T.

3 Optimizations

The simple generate and test strategy implemented by the SAT-based procedure outlined in Section 2 can be improved in several ways. Here we present two optimizations which often lead to dramatic improvements in the performance of the procedure.

3.1 Adding Constraints to the Input Formula

A key feature of the SAT-based procedure presented in Section 2 is that all the consistency checks are carried on-line. An alternative is to preprocess the formula and look for sets of literals in the input formula that are $\mathcal{L}$-inconsistent.[1] If S is one of such sets, the clause $\bigvee_{l \in S} \overline{l}$ can be added to the formula at hand without affecting its $\mathcal{L}$-consistency. This simple optimization can be very effective as shown by the following example.

Let $\mathcal{L}$ be the quantifier-free fragment of first-order logic with equality and let φ be a formula of the form

$$(x = y \wedge \neg y = x) \wedge \ldots \qquad (1)$$

[1] It is worth pointing out that this check can be carried out by any correct, even though not necessarily complete, procedure.

All the propositional assignments generated by DLL-SOLVE are then rejected by L-CONSIST. The useless generation of many propositional assignments is due to the failure of DLL-SOLVE to recognize that the truth values of $x = y$ and $y = x$ are not independent. The role of the constraints added by the proposed optimization is to rule out such assignments. For instance, by adding the constraint

$$\neg x = y \vee y = x \tag{2}$$

to (1) we obtain a formula which is readily found unsatisfiable by DLL-SOLVE.

In the context of SAT-based procedures, the idea of adding constraints has been introduced in [Armando *et al.*, 1999]. In that paper, all pairs of mutually inconsistent inequalities (i.e., $x - y \leq 0$ and $x - y \geq 5$) are detected *a priori* at a reasonable cost, and for each such pair a constraint is added. As a result, the search is greatly reduced. The idea can be generalized to n-uples of inconsistent inequalities, but only until the cost of the preprocessing remains sustainable.

The idea of constraints generalizes the pre-processing technique introduced in [Giunchiglia and Sebastiani, 1996a], and since then used in all the subsequent papers on SAT-based procedures for modal logics. In that paper, the input formula is initially pre-processed by taking into account standard properties of the propositional connectives, e.g. associativity and commutativity. Thus, for example, the formula

$$\Box(\psi_1 \vee \psi_2) \wedge \neg\Box(\psi_2 \vee \psi_1) \wedge \ldots \tag{3}$$

is translated into

$$\Box(\psi_1 \vee \psi_2) \wedge \neg\Box(\psi_1 \vee \psi_2) \wedge \ldots$$

and thus easily recognized as unsatisfiable. In the current approach, we can detect the $\mathcal{L}$-inconsistency of the set of formulae:

$$\{\Box(\psi_1 \vee \psi_2), \neg\Box(\psi_2 \vee \psi_1)\}$$

and this would add the constraint

$$\neg\Box(\psi_1 \vee \psi_2) \vee \Box(\psi_2 \vee \psi_1)$$

to (3) thereby leaving us with a trivially inconsistent formula. As a final remark, it is worth emphasizing that the strategy here presented is more general than pre-processing because it allows us to rule out assignments in cases where pre-processing is of no help. For example, consider the formula

$$\Box(\psi_1 \vee \psi_2) \wedge \neg\Box(\psi_1 \vee \psi_2 \vee \psi_3).$$

Using our strategy, by simple syntactic manipulations, we can build and add the following constraint:

$$\neg\Box(\psi_1 \vee \psi_2) \vee \Box(\psi_1 \vee \psi_2 \vee \psi_3)$$

```
LOOK-AHEAD(Γ, S)
 1 for each l s.t. l̄ ∧ ASSIGNMENT-IN(S)
       falsifies Γ with reason r do
 2   S ← PUSH(S, ⟨Γ, l, r⟩)
 3   Γ ← ASSIGN(Γ, l)
 4   if a clause r′ ∈ Γ
        has become empty then
 5      S ← PUSH(S, ⟨NIL, NIL, r′⟩)
 6      return LB
 7 if Γ is not empty then
 8   return HR
 9 else
10   r″ ← IS-CONSISTENT(S)
11   if r″ ≠ NIL then
12      S ← PUSH(S, ⟨NIL, NIL, r″⟩)
13      return LB
14   else
15      return T

IS-CONSISTENT(S)
 1 μ ← ASSIGNMENT-IN(S)
 2 if L-CONSIST(μ) = T then
 3   return NIL
 4 else
 5   return L-EXTRACT-REASON(μ)

HEURISTIC(Γ, S)
 1 Choose a literal l in Γ
 2 S ← PUSH(S, ⟨Γ, l, NIL⟩)
 3 Γ ← ASSIGN(Γ, l)
 4 return LA

LOOK-BACK(Γ, S)
 1 ⟨−, −, r⟩ ← POP(S)
 2 wr ← INIT-REASON(r)
 3 repeat
 4   ⟨Γ, l, r⟩ ← POP(S)
 5   wr ← UPDATE-REASON(wr, l, r)
 6 until r = NIL and
          IS-IN-REASON(l, wr)
 7 if length[S] > 0 then
 8   S ← PUSH(S, ⟨Γ, l̄, wr⟩)
 9   Γ ← ASSIGN(Γ, l̄)
10   return LA
11 else
12   return F
```

Fig. 2. Modifying DLL-SOLVE to introduce CBJ.

3.2 Introducing CBJ and Learning

Since the basic DLL algorithm of Section 2 relies on simple chronological backtracking, it is not infrequent for DLL to keep exploring a possibly large subtree whose leaves are all dead-ends. This phenomenon occurs also when the formula is satisfiable, but some choice performed way up in the search tree is responsible for the constraints to be violated. A solution borrowed from the constraint satisfaction literature (see, e.g., [Prosser, 1993]) is to jump back over the choices that do not belong to the reason for the failure. Intuitively, if μ is an assignment which falsifies the input formula φ, then a *reason* ν for μ is a subset of the literals in μ such that any assignment extending ν falsifies φ. Reasons are initialized as soon as an inconsistency is detected, and updated while backtracking. The corresponding technique is widely known as (Conflict-Directed) Backjumping (CBJ). In Figure 2 we show how to modify the functions LOOK-AHEAD, IS-CONSISTENT, HEURISTIC, and LOOK-BACK presented in Figure 1 to introduce CBJ. The elements of the search stack are now triples of the form $\langle \Gamma, l, r\rangle$, where Γ is a set of clauses, l a literal, and r a reason.

Looking at Figure 2 we see that each deduction carried out by LOOK-AHEAD is now justified by a *reason* (line 1). For example, if a literal l occurs in a unit clause, then it is assigned the truth value T and the reason for such an assignment is the set of literals that caused the clause to become unit, see, e.g., [Prosser, 1993]. Notice that LOOK-AHEAD records the reasons of each deduction using the search stack S (line 2), but also the reason for a propositional dead-end (line 5) and the possible failure of the IS-CONSISTENT test (line 12). All such reasons are used by the function LOOK-BACK in order to identify the choices performed by HEURISTIC that led to the dead-end in the search. Notice that the algorithms shown in Figure 2 smoothly combine CBJ for propositional failures as well as for failures originated by IS-CONSISTENT. In this case, the assignment in S propositionally satisfies Γ, so triggering propositional backjumping would lead to incorrect results. This is why we need an additional function L-EXTRACT-REASON in IS-CONSISTENT to extract the reason for the failure of μ in the specific logic at hand. The function L-EXTRACT-REASON has to return a subset of the literals in the current assignment which is not $\mathcal{L}$-consistent. In principle any such a set can be returned, e.g. the set of literals in μ. However, returning a smaller subset has the advantage of potentially enabling backjumping.

CBJ can be very effective in "shaking" the solver from regions where no solutions can be found. However, since the reasons of the conflict are discarded as soon as it gets mended, the solver may get repeatedly stuck in such regions. To escape this pattern, some sort of global knowledge is needed: the reasons of the conflicts may be turned into additional constraints (i.e., clauses) that have to be satisfied. As long as we have a function that turns reasons into clauses, it is quite easy to implement learning on top of DLL with CBJ. With reference to Figure 2, it is sufficient to add an instruction that converts the working reasons *wr* created inside LOOK-BACK into additional constraints. In this way, we end up adding all the clauses corresponding to the reasons of the discovered conflicts and the same mistake is never repeated. On the other hand, this may cause an exponential blow up of the size of the formula. In practice, it is necessary to introduce some limit to the number of stored clauses, either by dropping some of the clauses that should be learned, or by periodically removing some of the learnt clauses. For more details on learning see, e.g., [Giunchiglia *et al.*, 2001].

CBJ and learning have been proposed and successfully used in SAT-based procedures for planning [Wolfman and Weld, 1999, Castellini *et al.*, 2001]. In particular, [Wolfman and Weld, 1999] proposes a SAT-based procedure for classical planning with resources, while [Giunchiglia, 2000, Castellini *et al.*, 2001] present a SAT-based procedure for conformant planning in nondeterministic domains, see the respective papers for more details. It is worth mentioning that both in [Wolfman and Weld, 1999] and in [Castellini *et al.*, 2001] the function L-EXTRACT-REASON returns a minimal subset of the current assignment whose extensions are bound to fail. By returning such a subset, the hope is to maximise the effects of CBJ and learning.

4 SAT-Based Decision Procedures: Examples

In this Section we briefly review some specific examples of SAT-based decision procedures for quantifier-free decidable fragments of First-Order Logic (Subsection 4.1), temporal reasoning (Subsection 4.2) and various modal logics (Subsection 4.3).

4.1 Quantifier and Function-Free FOL

In [Armando and Giunchiglia, 1989, Armando and Giunchiglia, 1993], a SAT-based decision procedure for the quantifier- and function-free fragment of First-Order Logic (FOL) is presented. The language may thus have individual constants and variables as well as predicate symbols of any arity.

Let φ be formula in this language and let μ be an assignment returned by DLL-Solve(φ). If φ does not contain equalities, the existence of such an assignment μ is sufficient for the $\mathcal{L}$-consistency of φ. Thus, in this case, it is sufficient for L-Consist to return T. But if φ contains equalities, then L-Consist must determine the satisfiability of μ w.r.t. the properties of equality. More in detail, let $\mathcal{C}$ be the set of terms occurring in φ and let $\simeq$ be the smallest equivalence relation over $\mathcal{C}$ such that if $\mu(c_1 = c_2) = \top$ then $c_1 \simeq c_2$. Similarly, if $\mathcal{A}$ is the set of atomic subformulae occurring in φ then let $\cong$ be the smallest equivalence relation over $\mathcal{A}$ such that if $P(r_1, \ldots, r_n) \in \mathcal{A}$, $P(s_1, \ldots, s_n) \in \mathcal{A}$, and $r_i \simeq s_i$ for $i = 1, \ldots, n$, then $P(r_1, \ldots, r_n) \cong P(s_1, \ldots, s_n)$. An assignment μ is *satisfiable* if and only if it is not the case that there are two terms $c_1, c_2 \in \mathcal{C}$ such that $\mu(c_1 = c_2) = \bot$ and $c_1 \simeq c_2$, or there exist atomic formulae $A_1, A_2 \in \mathcal{A}$ such that $\mu(A_1) = \top$, $\mu(A_2) = \bot$, and $A_1 \cong A_2$. With reference to Figure 1, L-Consist

1. looks for the equalities $c_1 = c_2$ such that $\mu(c_1 = c_2) = \top$,
2. builds the data structures representing $\simeq$ and $\cong$, and
3. detects inconsistencies by exploiting the strategy suggested above.

The above procedure can be readily generalized to a SAT-based procedure for the quantifier-free fragment of FOL with uninterpreted function symbols by using a standard congruence closure algorithm (see, e.g., [Nelson and Oppen, 1980]) to perform the consistency checks.

4.2 Linear Constraints over the Reals

In [Armando *et al.*, 1999], the logic admits the function constant "$-$" and the domain of interpretation is fixed to the set of the real numbers. Formally, a *temporal constraint* is a linear inequality of the form $x - y \leq r$, where x and y are variables ranging over the real numbers and r is a real constant. A *disjunctive temporal constraint* is a disjunction of the form $c_1 \vee \cdots \vee c_n$ where $c_1, \ldots, c_n$ are temporal constraints and $n \geq 1$. A *disjunctive temporal problem* (DTP) is a finite set of disjunctive temporal constraints to be intended conjunctively. A *temporal assignment* is a function which maps each variable into a real number. A temporal assignment σ *satisfies* an assignment μ if, for each temporal constraint $x - y \leq r$,

- if $\mu(x - y \leq r) = \top$ then it is indeed the case that $\sigma(x) - \sigma(y) \leq r$,
- if $\mu(x - y \leq r) = \bot$ then it is indeed the case that $\sigma(x) - \sigma(y) > r$.

An *assignment is satisfiable* iff there exists a temporal assignment satisfying it. In the literature, the problem of determining whether an assignment is satisfiable or not is called a *Simple Temporal Problem* (STP). There are a number of procedures for checking the satisfiability of an STP, see, e.g., [Chleq, 1995]. The SAT-based decision procedure for checking the satisfiability of DTPs, TSAT, was implemented on top of Böhm's SAT solver [Buro and Buning, 1992, Böhm and Speckenmeyer, 1996]. TSAT proved to be more effective than the other procedures presented in the literature.[2] One of the reasons is that the semantic branching characteristic of DLL-SOLVE is superior (see also [Oddi and Cesta, 2000]) to the syntactic branching performed by tableau-based procedures proposed in [Stergiou and Koubarakis, 1998]. Moreover, since in TSAT consistency checks are performed by an optimized implementation of the simplex method, the system can efficiently handle temporal constraints involving the "$-$" and the "$+$" function symbols, multiplication by constants, and any finite number of variables, whereas the procedures proposed in [Stergiou and Koubarakis, 1998] and in [Oddi and Cesta, 2000] can only deal with the temporal constraints as defined above. Finally, we point out that a SAT-based procedure similar to TSAT is at the basis of the planning system described in [Wolfman and Weld, 1999], which is implemented on top of RELSAT [Bayardo, Jr. and Schrag, 1997].

4.3 Modal Logics

SAT-based decision procedures for modal logics have been proposed in [Giunchiglia and Sebastiani, 1996a, Giunchiglia and Sebastiani, 1996b, Giunchiglia *et al.*, 2000b], and have been comparatively evaluated in [Giunchiglia *et al.*, 2000a, Giunchiglia *et al.*, 2000b]. Moreover, in [Giunchiglia and Sebastiani, 1996a, Giunchiglia and Sebastiani, 1996b] the authors clearly pointed out the potentials of the SAT-based approach.

In the modal logics considered in the above cited papers, the language is extended by allowing denumerately many (modal) unary operators $\Box^1, \ldots, \Box^n$. Depending on the specific properties of each operator, different logics are obtained, from the weakest classical modal logic E, to the normal modal logic K [Chellas, 1980]. Decision procedures for 8 modal logics have been proposed in [Giunchiglia *et al.*, 2000b]. Here, for the sake of conciseness, we restrict our attention to the modal logics E and K.

Consider an assignment $\mu = \wedge_i(\wedge_j \Box^i \alpha_{ij}) \wedge \wedge_i \wedge_j \neg\Box^i \beta'_{ij} \wedge \gamma$ where γ is a propositional formula.

- In the modal logic E, μ *is satisfiable* if for each pair $\Box^i \alpha_{ij}$, $\neg\Box^i \beta'_{ik}$ of conjuncts in μ, the formula $\alpha_{ij} \equiv \neg\beta'_{ik}$ is satisfiable.

[2] The experimental results reported in [Armando *et al.*, 1999] show that TSAT performs up to 2 orders of magnitude less consistency checks than the best procedure presented in [Stergiou and Koubarakis, 1998].

```
L-Consist(∧ᵢ□αᵢ ∧ ∧ⱼ¬□βⱼ ∧ γ)
for each conjunct □βⱼ do
  for each conjunct □αᵢ do
    if not DLL-Solve(αᵢ ≡ ¬βⱼ)
      return then F
return T.
```

```
L-Consist(∧ᵢ□αᵢ ∧ ∧ⱼ¬□βⱼ ∧ γ)
for each conjunct □βⱼ do
  if not DLL-Solve(∧ᵢαᵢ ∧ ¬βⱼ)
    then return F
return T
```

Fig. 3. L-Consist for the modal logics E (left) and K (right).

- In the modal logic K, μ *is satisfiable* if for each conjunct $\neg\Box^i\beta'_{ij}$ in μ, the formula $\wedge_j\alpha_{ij} \wedge \neg\beta'_{ij}$ is satisfiable.

Thus, in E and in K the problem of determining whether an assignment is satisfiable or not boils down to the problem of determining the satisfiability of "simpler" formulae. Simpler, because the number of modal operators gets reduced. Still, modal operators can be nested, as any standard connective. Thus, in order to determine the satisfiability of these simpler formulae, L-Consist calls the DLL-Solve procedure. As a result, we have two mutually recursive procedures. The fact that at each call from L-Consist to DLL-Solve the number of modal operators diminishes guarantees termination of the whole process. Definitions of the L-Consist procedure for E and K are sketched in Figure 3, in case there is a single modality $\Box$. The extension to multiple modalities is straightforward.

Notice that the procedures for E and K of Figure 3 are naive and suffer from the fact that consistency checks of the same set of formulae can be repeated many times. A way out of the problem which has been proved very effective is the incorporation of caching mechanisms. For E, we check the consistency of pairs of formulae, and thus caching can be accomplished using a matrix, see [Giunchiglia *et al.*, 2000b]. For K, we check the consistency of sets of formulae, and thus more complex data structures, such as bit matrices, are needed [Giunchiglia and Tacchella, 2001].

5 Conclusions

In this paper we have provided a unifying perspective of a family procedures for automated reasoning based on the common idea of combining state-of-the-art SAT-solvers with reasoning specialists for the theory at hand. To substantiate our claim, we have shown that a variety of SAT-based procedures developed in the last decade (namely decision procedures for quantifier-free decidable fragments of First-Order Logic, for temporal reasoning, and for several modal logics) can be readily recast in our framework.

References

[Armando and Giunchiglia, 1989] A. Armando and F. Giunchiglia. On tautology decision techniques: Complexity and implementation considerations. Technical Report 8911-08, IRST, Trento, Italy, 1989.

[Armando and Giunchiglia, 1993] A. Armando and E. Giunchiglia. Embedding Complex Decision Procedures inside an Interactive Theorem Prover. *Annals of Mathematics and Artificial Intelligence*, 8(3–4):475–502, 1993.

[Armando *et al.*, 1999] A. Armando, C. Castellini, and E. Giunchiglia. SAT-based procedures for temporal reasoning. In *Lecture Notes in Computer Science*, volume 1809, pages 97–108, 1999.

[Bayardo, Jr. and Schrag, 1997] Roberto J. Bayardo, Jr. and Robert C. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI-97/IAAI-97)*, pages 203–208, Menlo Park, July 27–31 1997. AAAI Press.

[Biere *et al.*, 1999] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proceedings of the Fifth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '99)*, 1999.

[Böhm and Speckenmeyer, 1996] M. Böhm and E Speckenmeyer. A fast parallel SAT-solver – efficient workload balancing. *Annals of Mathematics and Artificial Intelligence*, 17:381–400, 1996.

[Buro and Buning, 1992] M. Buro and H. Buning. Report on a SAT competition. Technical Report 110, University of Paderborn, Germany, November 1992.

[Castellini *et al.*, 2001] Claudio Castellini, Enrico Giunchiglia, and Armando Tacchella. Improvements to sat-based conformant planning. In *ECP*, 2001.

[Chellas, 1980] B. F. Chellas. *Modal Logic – an Introduction.* Cambridge University Press, 1980.

[Chleq, 1995] N. Chleq. Efficient algorithms for networks of quantitative temporal constraints. In *Proceedings of CONSTRAINTS95*, pages 40–45, April 1995.

[Cormen *et al.*, 1998] Thomas H. Cormen, Charles E. Leiserson, and Ronald R. Rivest. *Introduction to Algorithms.* MIT Press, 1998.

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

[Dreben and Goldfarb, 1979] Burton Dreben and Warren D. Goldfarb. *The Decision Problem: Solvable Classes of Quantificational Formulas.* Addison-Wesley Publishing Company, Reading, MA, 1979.

[Gent *et al.*, 2000] Ian Gent, Hans Van Maaren, and Toby Walsh, editors. *SAT2000. Highlights of Satisfiability Research in the Year 2000.* IOS Press, 2000.

[Giunchiglia and Sebastiani, 1996a] F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures - the case study of modal K. In *Proc. CADE-96*, Lecture Notes in Artificial Intelligence, New Brunswick, NJ, USA, August 1996. Springer Verlag.

[Giunchiglia and Sebastiani, 1996b] F. Giunchiglia and R. Sebastiani. A SAT-based decision procedure for ALC. In *Proc. of the 5th International Conference on Principles of Knowledge Representation and Reasoning - KR'96*, Cambridge, MA, USA, November 1996. Also DIST-Technical Report 9607-08 and IRST-Technical Report 9601-02.

[Giunchiglia and Tacchella, 2001] Enrico Giunchiglia and Armando Tacchella. A subset-matching size-bounded cache for testing satisfiability in modal logics. *Annals of Mathematics and Artificial Intelligence*, 33:39–67, 2001.

[Giunchiglia *et al.*, 1998] E. Giunchiglia, F. Giunchiglia, R. Sebastiani, and A. Tacchella. More evaluation of decision procedures for modal logics. In *Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, 1998.

[Giunchiglia *et al.*, 2000a] E. Giunchiglia, F. Giunchiglia, R. Sebastiani, and A. Tacchella. SAT vs. Translation Based decision procedures for modal logics: a comparative evaluation. *Journal of Applied Non Classical Logics*, 10(2):145–172, 2000.

[Giunchiglia *et al.*, 2000b] E. Giunchiglia, F. Giunchiglia, and A. Tacchella. SAT-Based Decision Procedures for Classical Modal Logics. *Journal of Automated Reasoning*, 2000. To appear. Reprinted in [Gent *et al.*, 2000].

[Giunchiglia *et al.*, 2001] Enrico Giunchiglia, Marco Maratea, Armando Tacchella, and Davide Zambonin. Evaluating search heuristics and optimization techniques in propositional satisfiability. In *Proc. of the International Joint Conference on Automated Reasoning (IJCAR'2001), LNAI 2083*, 2001.

[Giunchiglia, 2000] Enrico Giunchiglia. Planning as satisfiability with expressive action languages: Concurrency, constraints and nondeterminism. In *Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR'00)*, 2000.

[Gu *et al.*, 1997] Jun Gu, Paul W. Purdom, John Franco, and Benjamin W. Wah. Algorithms for the satisfiability (sat) problem: A survey. *Satisfiability Problem: Theory and Applications*, pages 19–153, 1997.

[Kautz and Selman, 1992] Henry Kautz and Bart Selman. Planning as satisfiability. In *Proc. ECAI-92*, pages 359–363, 1992.

[Kautz and Selman, 1996] Henry Kautz and Bart Selman. Pushing the envelope: planning, propositional logic and stochastic search. In *Proc. AAAI-96*, pages 1194–1201, 1996.

[Li and Anbulagan, 1997] Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 366–371, San Francisco, August 23–29 1997. Morgan Kaufmann Publishers.

[Moskewicz *et al.*, 2001] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, June 2001.

[Nelson and Oppen, 1980] Greg Nelson and Derek C. Oppen. Fast decision procedures based on congruence closure. *Journal of the ACM*, 27(2):356–364, 1980.

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

[Prosser, 1993] Patrick Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9(3):268–299, 1993.

[Stergiou and Koubarakis, 1998] Kostas Stergiou and Manolis Koubarakis. Backtracking algorithms for disjunctions of temporal constraints. In *Proc. AAAI*, 1998.

[Wolfman and Weld, 1999] Steven Wolfman and Daniel Weld. The LPSAT-engine & its application to resource planning. In *Proc. IJCAI-99*, 1999.

[Zhang, 1997] H. Zhang. SATO: An efficient propositional prover. In William McCune, editor, *Proceedings of the 14th International Conference on Automated deduction*, volume 1249 of *LNAI*, pages 272–275, Berlin, July13–17 1997. Springer.