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

Alessandro Armando\textsuperscript{1}, Claudio Castellini\textsuperscript{1}, Enrico Giunchiglia\textsuperscript{1}, Fausto Giunchiglia\textsuperscript{2,3}, and Armando Tacchella\textsuperscript{1}

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

\textbf{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.

\section{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., 2006, 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 \( L = (L, T) \) where

\[
L \text{ is the language, i.e., a set of formulae in some formal language which includes the standard propositional connectives, i.e., the unary connective } \neg, \text{ the binary connective } \lor, \text{ and the } k\text{-ary } (k \geq 0) \text{ connectives } \land, \Leftrightarrow; \text{ and } \\
T \text{ is a theory, i.e. a subset of the language closed under propositional reasoning and the additional rules specific of the logic at hand.}
\]

We also assume that the logic is consistent, i.e., that for any formula \( \varphi \) in the language \( L \), it is not the case that both \( \varphi \) 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 \( L = (L, T) \) and a formula \( \varphi \in L \), is the formula \( L \)-consistent? That is, does \( \neg \varphi \) belong to \( L \setminus T \)?

The decidability of the logic ensures that the task of deciding the \( L \)-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 \( \{ \neg, \lor, \land, \Leftrightarrow \} \). A literal is an atom or the negation of an atom. An assignment \( \mu \) is a finite conjunction of literals such that it is not the case that both \( \psi \) and \( \neg \psi \) are conjuncts of \( \mu \). An assignment \( \mu \) satisfies a formula \( \varphi \) if the formula \( \mu \cup \varphi \) can be proved by propositional reasoning. We write \( \mu(A) = \top \) as an abbreviation for "\( A \) is a conjunct of \( \mu \)" and \( \mu(A) = \bot \) as an abbreviation for "\( \neg A \) is a conjunct of \( \mu \)."

The basic idea behind the SAT-based approach to determine the \( L \)-consistency of a formula \( \varphi \) 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. \( L \).
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 $\varphi$ 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 $\varphi$, if $\varphi$ is propositionally logically equivalent to the disjunction of the assignments in $S$; and
- **irredundant** for a formula $\varphi$, 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-Lemmon-Loveland (DLL) method [Davis et al., 1962] to develop our decision procedures. The reasons for this choice are manifold:

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], satos v3.2 [Zhang, 1997], sim [Gunnigie et al., 2001], and more recently chaff [Moskewicz et al., 2001].

A basic DLL implementation for SAT-based reasoning is 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. $l$, $s$, $t$) 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 $\text{NIL}$. Stacks are considered a primitive data type and are accessed with the usual constant-time primitives \texttt{push} and \texttt{pop}, while the function \texttt{empty} builds and returns an empty stack. The primitive \texttt{assign($l$)} returns the set of clauses $l$ minus all the clauses in which literal $l$ occurs, and with all the occurrences of its negation, $\overline{l}$, removed. Finally, we assume that $t$, $f$, $la$, $lb$, and $hr$ are five pairwise distinct constants, each one being distinct from $\text{NIL}$. In particular, $t$ and $f$ represent logical truth and falsehood, respectively.

Function \texttt{DLL-Solve} takes a formula $\varphi$ as input and returns $t$ ($f$) exactly when $\varphi$ is satisfiable (unsatisfiable, resp.). Function \texttt{DLL-Solve} converts $\varphi$ into
Fig. 1. Implementation of the DLL method for SAT-based reasoning.

an equi-satisfiable clausal normal form $\Gamma$ (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 $\varphi$. Function DLL-SOLVE also initializes the search stack $S$ (line 2) and then calls DLL-SOLVE-CNF to determine the satisfiability of $\Gamma$. The elements of the search stack are triples of the form $(\Gamma, l, flag)$, where $\Gamma$ is a set of clauses, $l$ a literal, and $flag \in \{T, F, LA, LB, HR\}$. Function ASSIGN-INST takes as input the search stack $S$ and returns a conjunction of the literals stored in it, i.e. $\wedge_{l \in S} l$. Function DLL-SOLVE-CNF solves $\Gamma$ by iteratively applying one of the following steps:

Look-Ahead to deduce new truth assignments from $\Gamma$. Look-Ahead keeps simplifying $\Gamma$ (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 $\Gamma$, then the return value is $\text{HR}$; if $\Gamma$ is empty, then all the clauses have been satisfied and the function $\text{Is-Consistent}$ is invoked.

**Heuristic** to decide the next truth assignment and to enforce it: the decision is taken by considering $\Gamma$ and/or possibly some $\Gamma'$ obtained from $\Gamma$ 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 $\Gamma$ 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 $\Gamma$ and $S$ and returns the next action to be taken which is stored in next (lines 4-6 in the program). $\text{LA}$, $\text{LB}$, $\text{HR}$, $\text{T}$, and $\text{F}$ 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 $\text{T}$ then $\Gamma$ is satisfied, otherwise (i.e. if next is assigned $\text{F}$) $\Gamma$ is unsatisfiable. When $\Gamma$ is satisfiable, the corresponding satisfying truth assignment $\mu$ can be extracted from $S$. This task is accomplished in Figure 1 by $\text{Is-Consistent}$ which extracts $\mu$ from $S$ (lines 1-3) and then calls the consistency test $\text{L-Consist}$ specific for the logic at hand. Notice that in the case of mere propositional satisfiability $\text{Is-Consistent}$ simply returns $\text{T}$.

# 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.\footnote{It is worth pointing out that this check can be carried out by any correct, even though not necessarily complete, procedure.} If $S$ is one of such sets, the clause $\bigvee_{l \in S} \neg l$ 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 $\varphi$ be a formula of the form

$$ (x = y \land \neg y = x) \land \ldots $$

(1)
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

$$-x = y \lor y = x$$

(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 \textit{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

$$\Box(\psi_1 \lor \psi_2) \land \neg\Box(\psi_2 \lor \psi_1) \land \ldots$$

(3)

is translated into

$$\Box(\psi_1 \lor \psi_2) \land \neg\Box(\psi_1 \lor \psi_2) \land \ldots$$

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

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

and this would add the constraint

$$-\Box(\psi_1 \lor \psi_2) \lor \Box(\psi_2 \lor \psi_1)$$

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

$$\Box(\psi_1 \lor \psi_2) \land \neg\Box(\psi_1 \lor \psi_2 \lor \psi_3).$$

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

$$-\Box(\psi_1 \lor \psi_2) \lor \Box(\psi_1 \lor \psi_2 \lor \psi_3)$$
### 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 $\mu$ is an assignment which falsifies the input formula $\varphi$, then a reason $\nu$ for $\mu$ is a subset of the literals in $\mu$ such that any assignment extending $\nu$ falsifies $\varphi$. 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 $\langle \Gamma, I, r \rangle$, where $\Gamma$ is a set of clauses, $I$ 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 $I$ occurs in a unit

<table>
<thead>
<tr>
<th><strong>Look-Ahead</strong>$(\Gamma$, $S)$</th>
<th><strong>Heuristic</strong>$(\Gamma$, $S)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 for each $I$ s.t. $I$ $\land$ <strong>Assignment-In</strong>$(S)$ falsifies $\Gamma$ with reason $r$ do</td>
<td></td>
</tr>
<tr>
<td>2 $S \leftarrow$ <strong>Push</strong>$(S, \langle \Gamma, I, r \rangle)$</td>
<td></td>
</tr>
<tr>
<td>3 $\Gamma \leftarrow$ <strong>Assign</strong>$(\Gamma, I)$</td>
<td></td>
</tr>
<tr>
<td>4 if a clause $r' \in \Gamma$ has become empty then</td>
<td></td>
</tr>
<tr>
<td>5 $S \leftarrow$ <strong>Push</strong>$(S, \langle \text{NIL}, \text{NIL}, r' \rangle)$</td>
<td></td>
</tr>
<tr>
<td>6 return $\text{LB}$</td>
<td></td>
</tr>
<tr>
<td>7 if $\Gamma$ is not empty then</td>
<td></td>
</tr>
<tr>
<td>8 return $\text{HR}$</td>
<td></td>
</tr>
<tr>
<td>9 else</td>
<td></td>
</tr>
<tr>
<td>10 $r'' \leftarrow$ <strong>Is-Consistent</strong>$(S)$</td>
<td></td>
</tr>
<tr>
<td>11 if $r'' \neq \text{NIL}$ then</td>
<td></td>
</tr>
<tr>
<td>12 $S \leftarrow$ <strong>Push</strong>$(S, \langle \text{NIL}, \text{NIL}, r'' \rangle)$</td>
<td></td>
</tr>
<tr>
<td>13 return $\text{LB}$</td>
<td></td>
</tr>
<tr>
<td>14 else</td>
<td></td>
</tr>
<tr>
<td>15 return $T$</td>
<td></td>
</tr>
</tbody>
</table>

**Is-Consistent**$(S)$

1 $\mu \leftarrow$ **Assignment-In**$(S)$

2 if $\text{L-Consist}$(\mu) = $T$ then

3 return $\text{NIL}$

4 else

5 return $\text{L-Extract-Reason}$(\mu)

**Look-Back**$(\Gamma$, $S)$

1 $(\neg, \neg, r) \leftarrow$ **Pop**$(S)$

2 $wr \leftarrow$ **Init-Reason**$(r)$

3 repeat

4 $(\Gamma, I, r) \leftarrow$ **Pop**$(S)$

5 $wr \leftarrow$ **Update-Reason**$(wr, I, r)$

6 until $r = \text{NIL}$ and

7 if $\text{length}$(\[\begin{array}{c} S \end{array}\]) > 0 then

8 $S \leftarrow$ **Push**$(S, \langle \Gamma, I, wr \rangle)$

9 $\Gamma \leftarrow$ **Assign**$(\Gamma, I)$

10 return $\text{LA}$

11 else

12 return $F$
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 $I$, 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 $\mu$ 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 $\mu$. 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 or 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 \( \varphi \) be formula in this language and let \( \mu \) be an assignment returned by DLL-Solve(\( \varphi \)). If \( \varphi \) does not contain equalities, the existence of such an assignment \( \mu \) is sufficient for the \( L \)-consistency of \( \varphi \). Thus, in this case, it is sufficient for \( L \)-CONSIST to return \( \top \). But if \( \varphi \) contains equalities, then \( L \)-CONSIST must determine the satisfiability of \( \mu \) w.r.t. the properties of equality. More in detail, let \( C \) be the set of terms occurring in \( \varphi \) and let \( \simeq \) be the smallest equivalence relation over \( C \) such that if \( \mu(c_1 = c_2) = \top \) then \( c_1 \simeq c_2 \). Similarly, if \( A \) is the set of atomic subformulae occurring in \( \varphi \) then let \( \equiv \) be the smallest equivalence relation over \( A \) such that if \( P(r_1, \ldots, r_n) \in A \) and \( P(s_1, \ldots, s_n) \in A \) then \( r_i \simeq s_i \) for \( i = 1, \ldots, n \), then \( P(r_1, \ldots, r_n) \equiv P(s_1, \ldots, s_n) \). An assignment \( \mu \) is satisfiable if and only if it is not the case that there are two terms \( c_1, c_2 \in C \) such that \( \mu(c_1 = c_2) = \bot \) and \( c_1 \simeq c_2 \), or there exist atomic formulae \( A_1, A_2 \in A \) such that \( \mu(A_1) = \top, \mu(A_2) = \bot \), and \( A_1 \equiv A_2 \). 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 \( \equiv \), 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 \textit{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 \textit{disjunctive temporal constraint} is a disjunction of the form \( c_1 \lor \cdots \lor c_n \) where \( c_1, \ldots, c_n \) are temporal constraints and \( n \geq 1 \). A \textit{disjunctive temporal problem} (DTP) is a finite set of disjunctive temporal constraints to be intended conjunctively. A \textit{temporal assignment} is a function which maps each variable into a real number. A temporal assignment \( \sigma \) \textit{satisfies} an assignment \( \mu \) 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 [Baro 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 \( \mu = \land_i (\land_j \Box^i \alpha_{ij}) \land \land_i \land_j \neg \Box^i \beta_{ij} \land \gamma \) where \( \gamma \) is a propositional formula.

In the modal logic \( E \), \( \mu \) is satisfiable if for each pair \( \Box^i \alpha_{ij}, \neg \Box^i \beta_{ik} \) of conjuncts in \( \mu \), 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].
<table>
<thead>
<tr>
<th>L-CONSIST($\land_i \Box \alpha_i \land \land_j \neg \Box j \beta_j \land \gamma$)</th>
<th>L-CONSIST($\land_i \Box \alpha_i \land \land_j \neg \Box j \beta_j \land \gamma$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 for each conjunct $\Box j \beta_j$ do</td>
<td>1 for each conjunct $\Box j \beta_j$ do</td>
</tr>
<tr>
<td>2 for each conjunct $\Diamond \alpha_i$ do</td>
<td>2 if not DLL-SOLVE($\land_i \alpha_i \land \neg \beta_j$)</td>
</tr>
<tr>
<td>3 if not DLL-SOLVE($\land_i \alpha_i \equiv \neg \beta_j$)</td>
<td>3 then return $F$</td>
</tr>
<tr>
<td>4 return $T$</td>
<td>4 return $T$</td>
</tr>
</tbody>
</table>

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

In the modal logic $K$, $\mu$ is satisfiable if for each conjunct $\neg \Box j \beta_j$ in $\mu$, the formula $\land_i \alpha_i \land \neg \beta_j$ 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 [Gurchiglia 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 [Gurchiglia and Tacchella, 2001].

5 Conclusions

In this paper we have provided a unifying perspective of a family of 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


[Giunchiglia and Tacchella, 2003] Enrico Giunchiglia and Armando Tacchella. A
subset-matching size-bounded cache for testing satisfiability in modal logics. As-

[Giunchiglia et al., 1998] E. Giunchiglia, F. Giunchiglia, R. Sebastiani, and A. Tac-
chella. More evaluation of decision procedures for modal logics. In Sixth Interna-
tional Conference on Principles of Knowledge Representation and Reasoning

[Giunchiglia et al., 2000a] E. Giunchiglia, F. Giunchiglia, R. Sebastiani, and A. Tac-
chella. SAT vs. Translation Based decision procedures for modal logics: a com-

[Giunchiglia et al., 2000b] E. Giunchiglia, F. Giunchiglia, and A. Tacchella. SAT-
Based Decision Procedures for Classical Modal Logics. Journal of Automated Rea-

[Giunchiglia et al., 2001] Enrico Giunchiglia, Marco Maratea, Armando Tacchella, and
Davide Zambonin. Evaluating search heuristics and optimization techniques in pro-
positional satisfiability. In Proc. of the International Joint Conference on Auto-

[Giunchiglia, 2000] Enrico Giunchiglia. Planning as satisfiability with expressive action
languages: Concurrency, constraints and nondeterminism. In Seventh Interna-
tional Conference on Principles of Knowledge Representation and Reasoning

miths for the satisfiability (sat) problem: A survey. Satisfiability Problem: Theory


[Kautz and Selman, 1996] Henry Kautz and Bart Selman. Pushing the envelope: plan-
nning, 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 propaga-
ation for satisfiability problems. In Proceedings of the 15th International Joint
Conference on Artificial Intelligence (IJCAI-97), pages 366–371, San Francisco, Au-

[Moskewicz et al., 2001] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lin-
tao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In Pro-
cedings of the 38th Design Automation Conference (DAC’01), June 2001.


[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

[Prosser, 1993] Patrick Prosser. Hybrid algorithms for the constraint satisfaction prob-


[Wolfman and Weld, 1999] Steven Wolfman and Daniel Weld. The LPSAT-engine & its

editor, Proceedings of the 14th International Conference on Automated deduction,