ON EFFICIENTLY INTEGRATING BOOLEAN AND THEORY-SPECIFIC SOLVING PROCEDURES

Roberto Sebastiani

June 2004

Technical Report # DIT-04-036

On Efficiently Integrating Boolean and Theory-Specific Solving Procedures

Roberto Sebastiani[*]
DIT, University of Trento, I-38050 Povo, Trento, Italy.
`www.dit.unitn.it/~rseba`
`rseba@dit.unitn.it`.

June 6, 2004

Abstract

Efficient decision procedures have been conceived in different communities for a very heterogeneous collection of expressive decidable theories. However, despite in the last years we have witnessed an impressive advance in the efficiency of SAT procedures, techniques for efficiently integrating boolean reasoning and theory-specific decision procedures have been deeply investigated only recently, producing impressive performance improvements when applied. The goal of this paper is to analyze, classify and represent within a uniform framework the most effective techniques which have been proposed in various communities in order to maximize the efficiency of boolean reasoning within decision procedures.

1 Motivation and goals

In the last decade we have witnessed an impressive advance in the efficiency of SAT techniques, which has brought large previously intractable problems at the reach of state-of-the-art solvers [38, 12, 46]. As a consequence, some hard real-world problems have been successfully solved by encoding them into SAT. Propositional planning [28] and boolean model-checking [13] are among the best achievements. Unfortunately, simple boolean expressions are not expressive enough for representing many real-world problems.

Efficient decision procedures for expressive decidable theories have been conceived in different communities, like, e.g., automated theorem proving, knowledge representation and reasoning, AI planning, formal verification. These procedures involve a very heterogeneous collection of theories, like modal logics [23, 42, 25, 21], description logics [24, 26, 32, 25], difference logic [1, 4, 29, 2], linear arithmetic [44, 4], logic of equality and uninterpreted functions (EUF) [18, 8, 20], logics of bit arrays and inductive datatypes [18, 8, 41], and, more generally, decidable subclasses of first order logic [8, 10, 43, 11].

In these communities a big effort has been concentrated in producing ad hoc procedures of increasing expressiveness and efficiency, and in combining them in the most efficient way [31,

[*]Sponsored by the CALCULEMUS! IHP-RTN EC project, code `HPRN-CT-2000-00102`, by a MIUR COFIN02 project, code `2002097822_003`, and by a grant from the Intel Corporation.

37, 18, 9, 36]. In many such communities, however, the importance of efficiently handling the embedded boolean component of reasoning in the theories has been under-estimated for a long time. Only recently new techniques for integrating efficient boolean reasoning within theory-specific decision procedures have been deeply investigated, producing impressive performance improvements when applied [23, 26, 32, 1, 18, 8, 41]. Nearly all such results have been produced by using SAT techniques based on variants of the DPLL algorithm [17, 16, 38, 12, 46].

The goal of this paper is to analyze, classify and represent within a uniform framework the most effective integration techniques which have been proposed in various communities, for the collection of theories listed above, in order to maximize the efficiency of boolean reasoning within decision procedures. We have made a big effort for making the description of these techniques independent from the theories which they have been conceived for.

The paper is structured as follows. In Section 2 we describe a common framework by which representing the various reasoning techniques, independently from their background theories. In Section 3 we present a basic and general schema for integrating DPLL-style boolean solving algorithms with specific theory solvers. In Section 4 we describe many integration techniques for maximizing the efficiency of the combined procedures. In Section 5 we conclude by surveying the chronological evolution of DPLL-based (and OBDD-based) procedures in the various domains and communities.

2 A uniform framework

In this paper we abstract away the information specifically related to the theories, and we consider a generic decidable theory $\mathcal{T}$, such that its semantics interprets the boolean connectives in the usual way. Notationally, we will often use the prefix "$\mathcal{T}$-" to denote "in the theory $\mathcal{T}$". E.g., we call a "$\mathcal{T}$-model" a model in $\mathcal{T}$, and so on.

For better readability, for our examples we will use the theory of linear arithmetic, because of its intuitive semantics. Nevertheless, analogous examples can be built with many other theories of interest.

2.1 Theoretical background.

Definition 2.1 *We call* $\mathcal{T}$-**atom** *any $\mathcal{T}$-formula that cannot be decomposed propositionally, that is, any $\mathcal{T}$-formula whose main connective is not a boolean operator. A* $\mathcal{T}$-**literal** *is either an atom (a* **positive** *literal) or its negation (a* **negative** *literal).*

Examples of $\mathcal{T}$-literals in different theories are, A_1 and $\neg A_2$ (boolean), $(x - y \leq 6)$ and $\neg(z - y \leq 2)$ (difference logic), $(v_1 + 5.0 \leq 2.0v_3)$ and $\neg(2v_1 + v_2 + 4v_3 = 5)$ (linear arithmetic), $\Box_1(A_1 \wedge \Box_2 A_2)$ and $\neg\Box_1(A_2 \rightarrow \Box_2 A_2)$ (modal logics), $\exists$ CHILDREN (MALE $\wedge$ TEEN) and $\neg\forall$ PARENT (OLD) (description logic), $((c = f(g(a,b)))$ and $\neg(c = g(f(b),h(a)))$ (EUF), and others.

If l is a negative $\mathcal{T}$-literal $\neg\psi$, then by "$\neg l$" we conventionally mean ψ rather than $\neg\neg\psi$. We denote by $Atoms(\varphi)$ the set of $\mathcal{T}$-atoms in φ. For simplicity, we sometimes omit the "$\mathcal{T}$-" prefix from "atom" and "literal" when not necessary

Definition 2.2 *We call a* **total truth assignment** *μ for a $\mathcal{T}$-formula φ a set*

$$\mu = \{\alpha_1, \ldots, \alpha_N, \neg\beta_1, \ldots, \neg\beta_M, A_1, \ldots, A_R, \neg A_{R+1}, \ldots, \neg A_S\}, \quad (1)$$

such that every atom in $Atoms(\varphi)$ occurs as a either positive or negative literal in μ. A **partial truth assignment** *μ for φ is a subset of a total truth assignment for φ. If $\mu_2 \subseteq \mu_1$, then we say that μ_1* **extends** *μ_2 and that μ_2* **subsumes** *μ_1.*

A total truth assignment μ is interpreted as a truth value assignment to all the atoms of φ: $\alpha_i \in \mu$ means that α_i is assigned to $\top$, $\neg\beta_i \in \mu$ means that β_i is assigned to $\bot$. Syntactically identical instances of the same atom are always assigned identical truth values; syntactically different atoms, e.g., $(t_1 \geq t_2)$ and $(t_2 \leq t_1)$, are treated differently and may thus be assigned different truth values.

Notationally, we use the Greek letters μ, η to represent truth assignments. Furthermore, we often write a truth assignment like (1) as the conjunction of its elements:

$$\mu = \bigwedge_i \alpha_i \wedge \bigwedge_j \neg\beta_j \wedge \gamma, \tag{2}$$

where $\gamma = \bigwedge_{k=1}^{R} A'_k \wedge \bigwedge_{h=R+1}^{S} \neg A'_h$ is a conjunction of propositional literals (if any). Thus, we say that an assignment is $\mathcal{T}$-satisfiable meaning that its corresponding $\mathcal{T}$-formula (2) is $\mathcal{T}$-satisfiable. We indifferently represent a truth assignment as a set (1) or as a formula (2).

Definition 2.3 *We say that a total truth assignment μ for φ* **propositionally $\mathcal{T}$-satisfies** *φ, written $\mu \models_p \varphi$, if and only if it makes φ evaluate to $\top$, that is, for all sub-formulas φ_1, φ_2 of φ:*

$$\begin{array}{lcl}
\mu \models_p \varphi_1,\ \varphi_1 \in Atoms(\varphi) & \iff & \varphi_1 \in \mu; \\
\mu \models_p \neg\varphi_1 & \iff & \mu \not\models_p \varphi_1; \\
\mu \models_p \varphi_1 \wedge \varphi_2 & \iff & \mu \models_p \varphi_1 \text{ and } \mu \models_p \varphi_2.
\end{array}$$

We say that a partial truth assignment μ **propositionally $\mathcal{T}$-satisfies** *φ if and only if all the total truth assignments for φ which extend μ propositionally $\mathcal{T}$-satisfy φ.*

From now on, if not specified, when dealing with propositional $\mathcal{T}$-satisfiability we do not distinguish between total and partial assignments.

We say that φ is *propositionally $\mathcal{T}$-satisfiable* if and only if there exist an assignment μ s.t. $\mu \models_p \varphi$. Intuitively, if we consider a $\mathcal{T}$-formula φ as a propositional formulas in its atoms, then $\models_p$ is the standard satisfiability in propositional logic. Thus, for every φ_1 and φ_2, we say that $\varphi_1 \models_p \varphi_2$ if and only if $\mu \models_p \varphi_2$ for every μ s.t. $\mu \models_p \varphi_1$. We also say that $\models_p \varphi$ (φ is *propositionally valid*) if and only if $\mu \models_p \varphi$ for every assignment μ for φ. It is easy to verify that $\varphi_1 \models_p \varphi_2$ if and only if $\models_p \varphi_1 \to \varphi_2$, and that $\models_p \varphi$ iff $\neg\varphi$ is propositionally $\mathcal{T}$-unsatisfiable.

Example 2.1 Consider the following $\mathcal{T}$-formula φ:

$$\begin{array}{rl}
\varphi = & \{\underline{\neg(2v_2 - v_3 > 2)} \vee A_1\} \wedge \\
& \{\underline{\neg A_2} \vee (2v_1 - 4v_5 > 3)\} \wedge \\
& \{\underline{(3v_1 - 2v_2 \leq 3)} \vee A_2\} \wedge \\
& \{\neg(2v_3 + v_4 \geq 5) \vee \underline{\neg(3v_1 - v_3 \leq 6)} \vee \neg A_1\} \wedge \\
& \{A_1 \vee \underline{(3v_1 - 2v_2 \leq 3)}\} \wedge \\
& \{\underline{(v_2 - v_4 \leq 6)} \vee (v_5 = 5 - 3v_4) \vee \neg A_1\} \wedge \\
& \{A_1 \vee \underline{(v_3 = 3v_5 + 4)} \vee A_2\}
\end{array}$$

The partial truth assignment given by the underlined literals above is:

$$\begin{array}{rclllr}
\mu & = & (v_2 - v_4 \leq 6) & \wedge\, (v_3 = 3v_5 + 4) & \wedge\, (3v_1 - 2v_2 \leq 3) \wedge & [\bigwedge_i \alpha_i] \\
 & & \neg(2v_2 - v_3 > 2) & \wedge\, \neg(3v_1 - v_3 \leq 6) & \wedge & [\bigwedge_j \neg\beta_j] \\
 & & \neg A_2. & & & [\gamma]
\end{array}$$

μ is a partial assignment which propositionally $\mathcal{T}$-satisfies φ, as it sets to true one literal of every disjunction in φ, so that every total assignment which extends μ propositionally $\mathcal{T}$-satisfies φ. Notice that μ is not $\mathcal{T}$-satisfiable, as its sub-assignment

$$(3v_1 - 2v_2 \leq 3) \wedge \neg(2v_2 - v_3 > 2) \wedge \neg(3v_1 - v_3 \leq 6) \tag{3}$$

(rows 3, 1 and 4 in φ) does not have any model. $\diamond$

Definition 2.4 *We say that a collection $\mathcal{M} = \{\mu_1, \ldots, \mu_n\}$ of partial assignments propositionally $\mathcal{T}$-satisfying φ is* **complete** *if and only if*

$$\models_p \varphi \leftrightarrow \bigvee_j \mu_j. \tag{4}$$

$\mathcal{M}$ is complete in the sense that, for every total assignment η s.t. $\eta \models_p \varphi$, there exists $\mu_j \in \mathcal{M}$ s.t. $\mu_j \subseteq \eta$. Therefore $\mathcal{M}$ is a compact representation of the whole set of total assignments propositionally $\mathcal{T}$-satisfying φ.

Proposition 2.1 *Let φ be a $\mathcal{T}$-formula and let $\mathcal{M} = \{\mu_1, \ldots, \mu_n\}$ be a complete collection of truth assignments propositionally satisfying φ. Then, φ is $\mathcal{T}$-satisfiable if and only if μ_j is $\mathcal{T}$-satisfiable for some $\mu_j \in \mathcal{M}$.*

2.2 Merging Boolean and Theory Solving

Proposition 2.1 allows for splitting $\mathcal{T}$-satisfiability of φ into two orthogonal components:

- a *purely boolean* one, consisting in the existence of a propositional model μ for φ;
- a *purely theory-dependent* one, consisting in the existence of a $\mathcal{T}$-model M for the set of $\mathcal{T}$-literals in μ.

This suggests that a decision procedure for a theory $\mathcal{T}$ can be decomposed by combining two basic ingredients: a *Truth Assignment Enumerator* and a *Theory Solver* for $\mathcal{T}$.

Definition 2.5 *We call a* **Truth Assignment Enumerator** *(*ENUMERATOR *henceforth) a total function which takes as input a $\mathcal{T}$-formula φ and returns a complete collection $\mathcal{M} := \{\mu_1, \ldots, \mu_n\}$ of assignments propositionally satisfying φ.*

Notice the difference between a truth assignment enumerator and a standard boolean solver: a boolean solver has to find *only one* satisfying assignment —or to decide there is none— while an enumerator has to find a *complete collection* of satisfying assignments.

Definition 2.6 *We call a* **Theory Solver** *($\mathcal{T}$-*SOLVER *henceforth) a total function which takes as input a collection of $\mathcal{T}$-literals μ and returns a $\mathcal{T}$-model satisfying μ, or Null if there is none.*

With some theories, like modal logics and description logics, the $\mathcal{T}$-models can be exponentially big wrt. the size of φ. In these cases, $\mathcal{T}$-SOLVER can simply return a boolean value stating the $\mathcal{T}$-satisfiability of μ.

2.3 Basic efficiency issues

According to the schema previously described, one critical point for the efficiency of a decision procedure is that the number of assignments in a complete set for φ is exponential in worst-case, and typically very big on average. This induces two considerations.

First, it is highly desirable to reduce as much as possible the number of assignments generated by the enumerator and checked by the theory solver. To do this, a key issue is choosing a *non redundant* ENUMERATOR, that is, one which always avoids generating assignments which subsume or are subsumed by already-generated ones [25, 5]. To this extent, DPLL [17, 16] and Ordered Binary Decision Diagrams (OBDD's) [14] can be used as non-redundant enumerators, whilst Semantic Tableaux [39, 19] are instead intrinsically redundant [23, 25, 5].

Second, if the $\mathcal{T}$-satisfiability problem is in PSPACE, we do not want our procedure to require exponential memory. To achieve that, it is necessary to adopt a generate-check-and-drop interaction paradigm between the ENUMERATOR and $\mathcal{T}$-SOLVER: at each step, ENUMERATOR generate the next assignment $\mu_i \in \mathcal{M}$, $\mathcal{T}$-SOLVER check its $\mathcal{T}$-satisfiability, and then μ is dropped —or the part of it which is not common to the next assignment is dropped— before passing to the $i+1$-step. This means that ENUMERATOR must be able to generate the assignments $\mu_1, ..., \mu_n$ one at a time. To this extent, DPLL and Semantic tableaux are suitable, as they perform a depth-first-search on assignments, whilst OBDD's are not, as they generate in one shot the whole OBDD, and thus the whole set of assignments.

The two consideration above motivate the fact that most state-of-the-art decision procedures in most theories are built on top of a DPLL-style boolean solver (see, e.g., [1, 2, 4, 44, 20, 21, 26, 32, 42, 18, 41]), although some efficient procedures for formal verification are built on top of OBDD's (e.g., [45, 30]). In the next session we will concentrate thus on analyzing DPLL-style decision procedures.

3 Integrating DPLL and $\mathcal{T}$-solver: a basic schema

A basic and general schema for integrating a DPLL-style assignment enumerator with a $\mathcal{T}$-SOLVER is reported in Figure 1. [1] $\mathcal{T}$-SAT takes in input a $\mathcal{T}$-formula φ and returns a truth value asserting whether φ is $\mathcal{T}$-satisfiable or not. $\mathcal{T}$-SAT invokes $\mathcal{T}$-DPLL passing as arguments φ and (by reference) an empty assignment μ and an empty $\mathcal{T}$-model M. $\mathcal{T}$-DPLL tries to build a $\mathcal{T}$-satisfiable truth assignment μ satisfying φ. This is done recursively, according to the following steps:

- (base) If $\varphi = T$, then μ propositionally $\mathcal{T}$-satisfies φ. Thus, if μ is $\mathcal{T}$-satisfiable, then φ is $\mathcal{T}$-satisfiable. Therefore $\mathcal{T}$-DPLL invokes $\mathcal{T}$-SOLVER(μ), which returns a $\mathcal{T}$-model for μ if it is $\mathcal{T}$-satisfiable, *Null* otherwise. $\mathcal{T}$-DPLL returns *True* in the first case, *False* otherwise.

- (backtrack) If $\varphi = F$, then μ has lead to a propositional contradiction. Therefore $\mathcal{T}$-DPLL returns *False*.

- (unit) If a literal l occurs in φ as a unit clause, then l must be assigned T.To obtain this, $\mathcal{T}$-DPLL is invoked recursively with arguments the formula returned by *assign(*l, φ*)* and

[1] For simplicity, we assume here that the input formulas are given in CNF. To see how to use a DPLL with non-CNF formulas, see [3, 22].

```
boolean $\mathcal{T}$-SAT(formula $\varphi$)
    $\mu := \{\}$;
    $M := Null$;
    return $\mathcal{T}$-DPLL($\varphi, \mu, M$);

boolean $\mathcal{T}$-DPLL(formula $\varphi$, assignment & $\mu$, $\mathcal{T}$-model & $M$)
    if ($\varphi = T$)                                          /* base      */
    then  $M := \mathcal{T}$-SOLVER($\mu$) ;
          return ($M \neq Null$) ;
    if ($\varphi = F$)                                          /* backtrack */
    then return False;
    if {$l$ occurs in $\varphi$ as a unit clause}               /* unit      */
    then return $\mathcal{T}$-DPLL(assign($l, \varphi$), $\mu \wedge l, M$);
    $l$ := choose-literal($\varphi$);                           /* split     */
    return    $\mathcal{T}$-DPLL(assign($l, \varphi$), $\mu \wedge l, M$)  or
              $\mathcal{T}$-DPLL(assign($\neg l, \varphi$), $\mu \wedge \neg l, M$);
```

Figure 1: The basic schema of a DPLL-based procedure.

the assignment obtained by adding l to μ. *assign(l, φ)* substitutes every occurrence of l in φ with T and propositionally simplifies the result.

- (split) If none of the above situations occurs, then *choose-literal(φ)* returns an unassigned literal l according to some heuristic criterion. Then $\mathcal{T}$-DPLL is first invoked recursively with arguments *assign(l, φ)* and $\mu \wedge \{l\}$. If the result is *False*, then $\mathcal{T}$-DPLL is invoked with arguments *assign($\neg l, \varphi$)* and $\mu \wedge \{\neg l\}$.

$\mathcal{T}$-DPLL is a variant of DPLL, modified to work as an assignment enumerator, whose $\mathcal{T}$-satisfiability is recursively checked by $\mathcal{T}$-SOLVER, in accordance with the generate-check-and-drop paradigm described in the previous section. The key difference wrt. standard DPLL is in the "base" step: standard DPLL simply returns *True*, as it needs finding only one satisfying assignment. The following result holds.

Proposition 3.1 *Given a $\mathcal{T}$-formula φ, the set of assignments generated and checked by the call $\mathcal{T}$-SAT(φ) is complete and non-redundant.*

Notice that Proposition 3.1 does not hold for all SAT solvers. For instance, the sets of assignments generated by DPLL with *pure literal rule* [16] may be incomplete; the sets generated by standard analytic tableaux are complete but may be redundant; as for Ordered Binary Decision Diagrams (OBDDs) [14], all the sets of the paths leading to "1" are complete and non-redundant, but they cannot be generated with a generate-check-and-drop policy.

4 Integrating DPLL and $\mathcal{T}$-solver: pushing the envelope

The basic and general schema of Figure 1 is rather naive from different viewpoints. First, it refers to a very basic schema of DPLL, without taking into consideration the most efficient techniques exploited by the recent DPLL implementations [12, 38, 46]. Second, and even more important, in the above schema $\mathcal{T}$-DPLL interacts with $\mathcal{T}$-SOLVER in a blind way, without exchanging any information with $\mathcal{T}$-SOLVER about the semantics of the $\mathcal{T}$-atoms to which it is assigning boolean values. This may cause a up to huge amount of calls to $\mathcal{T}$-SOLVER on assignments which are obviously $\mathcal{T}$-inconsistent, or whose inconsistency could have been easily derived from that of previously checked assignments.

In this section we collect, classify and describe the most effective integration techniques which have been proposed in various communities, for the theories listed in Section 1. Such techniques have proved extremely effective in their respective domains.

4.0.1 Preprocessing atoms [23, 26, 32, 4].

One potential source of inefficiency for $\mathcal{T}$-DPLL is the fact that semantically equivalent but syntactically different atoms are not recognized to be identical [resp. one the negation of the other] and thus they may be assigned different [resp. identical] truth values. For instance, syntactically different atoms may be equivalent modulo ordering $((v_1+v_2 \le v_3), (v_2+v_1 \le v_3))$, associativity $((v_1+(v_2+v_3))$ and $((v_1+v_2)+v_3)))$, factorization $((2v_1-6v_2=14)$ and $(3v_1-9v_2=21))$, simple term rewriting $((v_1+v_2 \le v_3)$ and $(v_1+v_2-v_3 \le 0))$ and so on, or may be one the negation of the other $((v_1 < v_2)$ and $(v_1 \ge v_2)$, or $(2v_1-6v_2 \le 4)$ and $(3v_2+2 < v_1))$. This causes the undesired generation of a potentially very big amount of intrinsically $\mathcal{T}$-unsatisfiable assignments (for instance, up to $2^{Atoms(\varphi)-2}$ assignments of the kind $\{(v_1 < v_2), (v_1 \ge v_2), ...\}$).

To avoid these problems, it is wise to preprocess atoms so that to map as many as possible semantically equivalent atoms into syntactically identical ones. Of course, this mapping depends on the the theory addressed. Some common steps can be:

- *exploit associativity* (e.g., $(v_1+(v_2+v_3))$ and $((v_1+v_2)+v_3)) \Longrightarrow (v_1+v_2+v_3)$);
- *sorting* (e.g., $(v_1+v_2-1 \le v_3)$, $(v_2+v_1 \le v_3+1) \Longrightarrow (v_1+v_2-v_3 \le 1)$);
- *removing dual operators* (e.g., $(v_1 < v_2)$, $(v_1 \ge v_2) \Longrightarrow (v_1 < v_2)$, $\neg(v_1 < v_2)$);
- *exploiting specific properties of* $\mathcal{T}$ (e.g., $(v_1 \le 3)$, $(v_1 < 4) \Longrightarrow (v_1 \le 3)$ if $v_1 \in \mathbb{Z}$).

4.0.2 Early pruning [23, 4, 41]

Another improvement starts from the observation that typically most assignments found by $\mathcal{T}$-DPLL are "trivially" $\mathcal{T}$-unsatisfiable, in the sense that their $\mathcal{T}$-unsatisfiability is caused by much smaller subsets, which we call *conflict sets*. If an assignment μ' is $\mathcal{T}$-unsatisfiable, then all its extensions are $\mathcal{T}$-unsatisfiable. If the unsatisfiability of an assignment μ' is detected during its recursive construction, then this prevents checking the $\mathcal{T}$-satisfiability of all the up to $2^{|Atoms(\varphi)|-|\mu'|}$ truth assignments which extend μ'.

This suggests to introduce an intermediate $\mathcal{T}$-satisfiability test on intermediate assignments just before the "split" step:

```
if Likely-Unsatisfiable($\mu$)                /* early pruning */
    if ($\mathcal{T}$-SOLVER($\mu$) = Null)
        then return False;
```

If the heuristic *Likely-Unsatisfiable* returns *True*, then $\mathcal{T}$-SOLVER is invoked on the current assignment μ. If $\mathcal{T}$-SOLVER(μ) returns *Null*, then all possible extensions of μ are unsatisfiable, and therefore $\mathcal{T}$-DPLL returns *False*, avoiding a possibly big amount of useless search.

Example 4.1 Consider the formula φ of Example 2.1. Suppose that, after four recursive calls, $\mathcal{T}$-DPLL builds the intermediate assignment:

$$\mu' \quad = \quad \neg(2v_2 - v_3 > 2) \wedge \neg A_2 \wedge (3v_1 - 2v_2 \leq 3) \wedge \neg(3v_1 - v_3 \leq 6). \tag{5}$$

(rows 1, 2, 3 and 4 of φ). If $\mathcal{T}$-SOLVER is invoked on μ', it returns *Null*, and $\mathcal{T}$-DPLL backtracks without exploring any extension of μ'. $\diamond$

Likely-Unsatisfiable avoids invoking $\mathcal{T}$-SOLVER only when it is very unlikely that, since last call, the new literals added to μ' can cause inconsistency. For instance, this is the case when they are added only literals which either are purely-propositional or contain new variables. Notice that, in most solvers implementing this technique, *Likely-Unsatisfiable* returns true most often or nearly always [23, 4, 41, 2].

Some DPLL-based procedures (e.g., [41, 20]) perform a more eager form of early pruning, in which the theory solver is invoked every time a new $\mathcal{T}$-atom is added to the assignment (including those added by unit propagation). The eager approach benefits of a more aggressive pruning due to $\mathcal{T}$-inconsistencies, but pays for extra overhead.

Early pruning techniques, similar to these described here, have also been used for OBDD-based procedures to prune the size of the OBDD's [15, 45, 30].

4.0.3 Theory-driven Backjumping [26, 32, 44]

An alternative optimization arises from the same observations as those for early pruning. Any branch containing a conflict set is $\mathcal{T}$-unsatisfiable. Thus, suppose $\mathcal{T}$-SOLVER is modified to return also a conflict set η causing the $\mathcal{T}$-unsatisfiability of the input assignment μ. If so, $\mathcal{T}$-DPLL can jump back in its search to the deepest branching point in which a literal $l \in \eta$ is assigned a truth value, pruning the search space below.

This technique is very similar to that of backjumping in standard DPLL procedures (see, e.g., [38, 12]). The only difference is in the notion of conflict set used, which is an assignment which is intrinsically inconsistent in the theory $\mathcal{T}$, rather than an assignment which, if added to φ, forces propositional inconsistency.

Example 4.2 Consider the formula φ and the assignment μ of Example 2.1. Suppose that $\mathcal{T}$-DPLL generates μ following the order of occurrence within φ, and that $\mathcal{T}$-SOLVER(μ) returns the conflict set (3). Thus $\mathcal{T}$-DPLL can backjump directly to the branching point $\neg(3v_1 - v_3 \leq 6)$, without branching first on $(v_3 = 3v_5 + 4)$ and $\neg(2v_2 - v_3 > 2)$. $\diamond$

Remark 4.1 *Early Pruning vs. Theory-driven Backjumping.* Theory-driven backjumping is alternative to Early Pruning. In fact, if $\mathcal{T}$-SOLVER is invoked at every branching point, then Theory-driven Backjumping is useless, because the procedure will always backtrack to the most recent branching point, as is standard DPLL.

There is an efficiency tradeoff between the two techniques. On the one hand, Early Pruning always prunes search at the highest branching point possible; this is not the case of Theory-driven Backjumping, because the conflict set returned is not necessarily the one which allows for the highest backjumping. On the other hand, unlike Theory-driven Backjumping, Early Pruning may perform lots of useless calls to $\mathcal{T}$-SOLVER.

Thus, the choice between the two techniques depends on the relative cost of $\mathcal{T}$-SOLVER, and thus on how much one can trade boolean search reduction for extra $\mathcal{T}$-solving. For instance, is $\mathcal{T}$ is a modal/description logics with high nesting depths, each call to $\mathcal{T}$-SOLVER can be very expensive [2] and thus backjumping can be a better option; instead, for $\mathcal{T}$ being linear arithmetic, or a modal/description logic with low depths, $\mathcal{T}$-solving is relatively cheap wrt. the benefits of better boolean search pruning, so that early pruning becomes a better option [27, 33, 4]. □

4.0.4 Memo-izing [32, 42, 21]

When $\mathcal{T}$-SOLVER is invoked on an assignment μ, typically only a strict subpart of μ is actually analyzed by the $\mathcal{T}$-SOLVER (e.g., in most theories all purely boolean literals in μ are irrelevant). As a consequence, the two following situations may happen frequently:

1. (The relevant part of) μ is a superset of an assignment η which has already been found $\mathcal{T}$-inconsistent by $\mathcal{T}$-SOLVER.

2. (The relevant part of) μ is a subset of an assignment η which has already been found $\mathcal{T}$-consistent by $\mathcal{T}$-SOLVER.

In both cases, the call to $\mathcal{T}$-SOLVER(μ) is useless. Thus, if efficient data structure are used to memorize the in/consistency values of the assignments analyzed by $\mathcal{T}$-SOLVER, this may save a lot of redundant calls to $\mathcal{T}$-SOLVER.

The drawback of this technique is that it may require exponential space to memorize the up to exponentially many assignments checked. Nevertheless, this technique proved to be extremely successful in modal and description logics, where the calls to $\mathcal{T}$-SOLVER may be extremely expensive [32, 42, 21].

4.0.5 Deduction of unassigned atoms [1, 4, 10, 43, 11, 20]

For some theories, it is possible to implement $\mathcal{T}$-SOLVER in such a way that any call to it can also produce information which can be used to reduce search afterwords. In fact, $\mathcal{T}$-SOLVER may be sophisticated enough to assign truth values to some yet-unassigned atoms as a logical consequence (in $\mathcal{T}$) of the literals in μ. If this is the case, these new assignments can be returned by $\mathcal{T}$-SOLVER as a new assignment η, which is unit-propagated away by $\mathcal{T}$-DPLL.

For instance, assume that all the following $\mathcal{T}$-atoms occur in the input formula φ. If $(v_1 - v_2 \leq 4) \in \mu$ and $(v_1 - v_2 \leq 6) \notin \mu$, then $\mathcal{T}$-SOLVER can derive deterministically that the latter is true, and thus return an assignment η containing $(v_1 - v_2 \leq 6)$. Similarly, if $(v_1 = v_3), (v_2 - v_3 > 4) \in \mu$ and $(v_2 - v_1 < 2) \notin \mu$, then $\mathcal{T}$-SOLVER(μ) returns η containing $\neg(v_2 - v_1 < 2)$.

As for the early-pruning technique, the deduction of unassigned atoms can be applied either in a lazy way, before any new branching [4], or, more eagerly, every time a new $\mathcal{T}$-atom is added to the assignment (including those added by unit propagation) [1, 10, 43, 11, 20]. The benefits of this technique depend strongly on how cheap is its implementation. For some simple theories, like those involving simple equality propagation, this can be implemented very efficiently [4, 20]. [3]

[2] This is due to the fact that, for these logics, $\mathcal{T}$-SOLVER requires recursive calls to $\mathcal{T}$-DPLL [23, 26, 32].

[3] An obvious and general way to implement this technique is by calling $\mathcal{T}$-SOLVER($\mu \wedge \neg l$) for every unassigned literal l. Of course, unless some particular applications [1], this is not an efficient way to implement it in general.

4.0.6 Static Learning [1, 7]

On some specific kind of satisfiability problems, like that of disjunctive temporal constraints problems (DTP) [1], it is possible to easily detect a priori pairs of $\mathcal{T}$-literals which are intrinsically mutually inconsistent (like, e.g., $\{(x-y \leq 3), (y-x \leq -2)\}$ and $\{\neg(x-y \leq 3), (x-y \leq 2)\}$). If so, binary mutual exclusion clauses (e.g., $\neg(x-y \leq 3) \vee \neg(y-x \leq -2)$ and $(x-y \leq 3) \vee \neg(x-y \leq 2)$) can be added a priori to the formula, which allow the solver to avoid generating any assignment including the inconsistent pairs. This technique, called $IS(2)$, allows for dramatically reducing the computational effort of solving DTP tests [1].

$IS(2)$ is a particular form of a more general technique which we call *static learning*: in a preprocessing phase, constraints on the $\mathcal{T}$-atoms of the formula can be "learned" and added in form of clauses to the input formula, which guide the SAT solver to avoid generating theory-inconsistent assignments. (Similar such techniques were embedded in the encoder for Bounded Model Checking for Timed and Hybrid automata [7, 6].)

Notice that, unlike the extra clauses added in [40, 35], the clauses added by static learning refer only to atoms which already occur in the original formula, so that no new atom is added. This means that the boolean search space is not enlarged. Furthermore, we notice that, unlike with [40, 35], these added clauses are not needed for completeness, but rather they are only used for reducing the search space.

4.0.7 Theory-driven Learning [44, 4]

When $\mathcal{T}$-SOLVER returns a conflict set η, the clause $\neg\eta$ can be added in conjunction to φ. Since then, $\mathcal{T}$-DPLL will never again generate any branch containing η.

Example 4.3 As in Example 4.2, suppose $\mathcal{T}$-SOLVER(μ) returns the conflict set (3). Then the clause

$$\neg(3v_1 - 2v_2 \leq 3) \vee (2v_2 - v_3 > 2) \vee (3v_1 - v_3 \leq 6)$$

is added in conjunction to φ. Thus, whenever a branch contains two elements of (3), then $\mathcal{T}$-DPLL will assign the third to *False* by unit propagation. ◇

Notice that Theory-driven Learning can be used with both Intermediate Assignment Checking and Backjumping. Notice also that Theory-driven Learning subsumes Technique 1 of the Memorizing optimization described above.

Theory-driven Learning is a technique which must be used with some care, as it may cause an explosion in size of φ. To avoid this, one has to introduce techniques for discarding learned clauses when necessary [12].

4.0.8 Exploiting pure $\mathcal{T}$-atoms [44, 4]

This technique was implicitly proposed in [44] and then generalized in [4], and comes from the following general property.

Property 4.1 Let C be a non-boolean atom occurring only positively [resp. negatively] in φ. Let $\mathcal{M}$ be a complete set of assignments satisfying φ, and let

$$\mathcal{M}' := \{\mu_j/\neg C\} | \mu_j \in \mathcal{M}\} \quad [resp. \; \{\mu_j/C\} | \mu_j \in \mathcal{M}\}].$$

Then (i) $\eta' \models_p \varphi$ for every $\eta' \in \mathcal{M}'$, and (ii) φ is satisfiable if and only if there exist a satisfiable $\eta' \in \mathcal{M}'$. $\square$

Thus, if we have non-boolean $\mathcal{T}$-atoms occurring only positively [negatively] in the input formulas, we can drop any negative [positive] occurrence of them from the assignment to be checked by $\mathcal{T}$-SOLVER.

This is particularly useful in some situations. For instance, in mathematical reasoning many solvers cannot handle efficiently disequalities (e.g., $(v_1 - v_2 \neq 3.2)$), so that they are forced to split them into the disjunction of strict inequalities $(v_1 - v_2 > 3.2) \vee (v_1 - v_2 < 3.2)$. In many problems its very frequent that an equality like $(v_1 - v_2 = 3.2)$ occurs with positive polarity only. If so, and if it is assigned a false value by $\mathcal{T}$-DPLL, the technique described avoids adding to μ the corresponding disequality $(v_1 - v_2 \neq 3.2)$.

5 Credits

In this section we try to survey the chronological evolution of DPLL-based (and OBDD-based) procedures in the various domains and communities.

The first DPLL-based procedure we are aware of was a simple procedure for the logic of equality [3], which was embedded in the GETFOL interactive theorem prover.

[23, 24] presented KSAT, the first DPLL-based procedures for modal and description logics, which outperformed previous procedures based on semantic tableaux; they introduced the $\mathcal{T}$-DPLL schema of Section 3, together with the Atom Preprocessing and the Early Pruning techniques; they analyzed and discussed the advantages of DPLL-based procedures wrt. those based on semantic tableaux.

[34, 25] provided a general schema for SAT-based procedures for every normal modal logic, and the general formal framework to build SAT based procedures of Section 2.

[26, 32] introduced FACT and DLP, very effective DPLL-besed decision procedures for expressive description logics; Theory-Driven Backjumping and Memoizing, plus some further Preprocessing, were introduced there.

[42, 21] presented ESAT and *SAT, DPLL-based procedures for both normal and non-normal modal logics, introducing new memoizing schemas.

[1] presented TSAT, a DPLL-based procedure for difference logic, which outperformed previous tableau-based procedures for the same problem. IS(2) and a form of Deduction of Unassigned Atoms were introduced there. TSAT++ is a recent optimized reimplementation of TSAT [2].

[44] presented LPSAT, a DPLL-based procedure for sets of disjunctive linear equalities and non-strict inequalities (only positive occurrences of $\mathcal{T}$-atoms were admitted), and used it to solve problems in the domain of resource planning. Theory-driven Learning, eager Early Pruning and Pure $\mathcal{T}$-atoms exploiting were introduced there.

[10, 43, 11] presented general frameworks for DPLL-based procedures for first order logics. Effective forms of eager Deduction of Unassigned Atoms and Early Pruning were introduced there.

[4, 5, 7, 6] presented MATHSAT, a DPLL-based procedure for difference logic and linear arithmetic constraints, which allowed to solve new kinds of verification problems on timed and hybrid systems. MATHSAT features a layered structure of $\mathcal{T}$-SOLVER.

Contemporarily, in [8, 41, 18] the expressive decision procedures SVC/CVC and ICS for Shostak-style combined theories —including difference logic, arithmetic, EUF, bit arrays and inductive datatypes— were integrated with a DPLL solver, gaining impressive performance gaps. These integrated procedures perform eager forms of Early Pruning and of Deduction of Unassigned Atoms.

In the Model Checking community there has been some work on integrating OBDDs [14] with numerical information in order to handle complex verification problems. [15] integrated OBDDs with a quadratic constraint solver to verify transition systems with integer data values; [30] developed Difference Decision Diagrams (DDDs), OBDD-like data structures handling boolean combinations of temporal constraints, and used them to verify timed systems; [45] developed a library of procedures combining OBDDs and a solver for Pressburger arithmetic, and used them to verify infinite-state systems. Unfortunately, all these approaches inherit from OBDDs the drawback of requiring exponential space in worst case.

Acknowledgements

Both my own work on SAT-based decision procedures and the survey and classification of the main works on the topic, which have been described here, have been carried on with the direct collaboration of many people, including G. Audemard, M. Bozzano, P. Bertoli, A. Cimatti, E. Giunchiglia, F. Giunchiglia, I. Horrocks, A. Kornilowicz, P.F. Patel-Schneider, M. Roveri, and A. Tacchella. I have also much benefited from fruitful discussions with many other people, including A. Armando, E. Franconi, U. Hulstadt, D. McAllester, R. Schmidt, B. Selman, L. Serafini, C. Tinelli, S. Tonetta, and M. Vardi.

Some material discussed here was presented at the IJCAI’03 and CADE’03 Tutorials “SAT Beyond Propositional Satisfiability”, whose slides can be downloaded from the link
`www.dit.unitn.it/~rseba/DIDATTICA/Tutorials/Slides_tutorial_cade04.pdf`.

The list of references is by no means intended to be all-inclusive. I apologize with both the authors and the readers for all the relevant papers which they do not find there.

References

[1] A. Armando, C. Castellini, and E. Giunchiglia. SAT-based procedures for temporal reasoning. In *Proc. European Conference on Planning, CP-99*, 1999.

[2] A. Armando, C. Castellini, E. Giunchiglia, and M. Maratea. A SAT-based Decision Procedure for the Boolean Combination of Difference Constraints. Vancouver, BC, Canada, 2004. Presented at SAT 2004.

[3] 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.

[4] G. Audemard, P. Bertoli, A. Cimatti, A. Korniłowicz, and R. Sebastiani. A SAT Based Approach for Solving Formulas over Boolean and Linear Mathematical Propositions. In *Proc. CADE'2002.*, volume 2392 of *LNAI*. Springer, July 2002.

[5] G. Audemard, P. Bertoli, A. Cimatti, A. Korniłowicz, and R. Sebastiani. Integrating Boolean and Mathematical Solving: Foundations, Basic Algorithms and Requirements. In *Proc. AIARSC'2002*, volume 2385 of *LNAI*. Springer, 2002.

[6] G. Audemard, M. Bozzano, A. Cimatti, and R. Sebastiani. Verifying Industrial Hybrid Systems with MathSAT. In *Proc. PDPAR'03*, 2003.

[7] G. Audemard, A. Cimatti, A. Korniłowicz, and R. Sebastiani. SAT-Based Bounded Model Checking for Timed Systems. In *Proc. FORTE'02.*, volume 2529 of *LNCS*. Springer, November 2002.

[8] C. Barrett, D. Dill, and A. Stump. Checking Satisfiability of First-Order Formulas by Incremental Translation to SAT. In *14th International Conference on Computer-Aided Verification*, 2002.

[9] C. W. Barrett, D. L. Dill, and A. Stump. A generalization of Shostak's method for combining decision procedures. In *Frontiers of Combining Systems (FROCOS)*, LNAI. Springer-Verlag, April 2002. Santa Margherita Ligure, Italy.

[10] P. Baumgartner. FDPLL - A First Order Davis-Putnam-Longeman-Loveland Procedure. In *Proceedings of CADE-17*, pages 200–219. Springer-Verlag, 2000.

[11] P. Baumgartner and C. Tinelli. The Model Evolution Calculus. In F. Baader, editor, *Proc. CADE-19*, number 2741 in LNAI, pages 350–364. Springer, 2003.

[12] R. J. Bayardo, Jr. and R. C. Schrag. Using CSP Look-Back Techniques to Solve Real-World SAT instances. In *Proc. AAAI'97*, pages 203–208. AAAI Press, 1997.

[13] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proc. CAV'99*, 1999.

[14] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[15] W. Chan, R. J. Anderson, P. Beame, and D. Notkin. Combining constraint solving and symbolic model checking for a class of systems with non-linear constraints. In *Proc. CAV'97*, volume 1254 of *LNCS*, pages 316–327, Haifa, Israel, June 1997. Springer.

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

[17] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[18] Jean-Christophe Filliâtre, Sam Owre, Harald Rueß, and N. Shankar. ICS: Integrated Canonizer and Solver. Proc. CAV'2001, 2001.

[19] M. Fitting. *Proof Methods for Modal and Intuitionistic Logics*. D. Reidel Publishg, 1983.

[20] H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): Fast Decision Procedures. In *Proc. CAV'04*, LNCS. Springer, 2004. (To appear).

[21] E. Giunchiglia, F. Giunchiglia, and A. Tacchella. SAT Based Decision Procedures for Classical Modal Logics. Journal of Automated Reasoning. Special Issue: Satisfiability at the start of the year 2000, 2001.

[22] E. Giunchiglia and R. Sebastiani. Applying the Davis-Putnam procedure to non-clausal formulas. In *Proc. AI*IA'99*, volume 1792 of *LNAI*. Springer, 1999.

[23] F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures - the case study of modal K. In *CADE-13*, LNAI, New Brunswick, NJ, USA, August 1996. Springer Verlag.

[24] 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.

[25] F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures - the case study of modal K(m). *Information and Computation*, 162(1/2), October/November 2000.

[26] I. Horrocks. The FaCT system. In H. de Swart, editor, *Proc. TABLEAUX-98*, volume 1397 of *LNAI*, pages 307–312. Springer, 1998.

[27] I. Horrocks, P. F. Patel-Schneider, and R. Sebastiani. An Analysis of Empirical Testing for Modal Decision Procedures. *Logic Journal of the IGPL*, 8(3):293–323, May 2000.

[28] H. Kautz, D. McAllester, and B. Selman. Encoding Plans in Propositional Logic. In *Proc. KR'96*, 1996.

[29] M. Mahfoudh, P. Niebert, E. Asarin, and O. Maler. A Satisfibaility Checker for Difference Logic. In *Proceedings of SAT-02*, pages 222–230, 2002.

[30] J. Moeller, J. Lichtenberg, H. R. Andersen, and H. Hulgaard. Fully symbolic model checking of timed systems using difference decision diagrams. In *Proc. Workshop on Symbolic Model Checking (SMC), FLoC'99*, Trento, Italy, July 1999.

[31] C. G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *TOPLAS*, 1(2):245–257, 1979.

[32] P. F. Patel-Schneider. DLP system description. In *Proc. DL-98*, pages 87–89, 1998.

[33] P. F. Patel-Schneider and R. Sebastiani. A New General Method to Generate Random Modal Formulae for Testing Decision Procedures. *Journal of Artificial Intelligence Research, (JAIR)*, 18:351–389, May 2003. Morgan Kaufmann.

[34] R. Sebastiani and A. Villafiorita. SAT-based decision procedures for normal modal logics: a theoretical framework. In *Proc. AIMSA'98*, volume 1480 of *LNAI*. Springer, 1998.

[35] S. A. Seshia, S. K. Lahiri, and R. E. Bryant. A Hybrid SAT-Based Decision Procedure for Separation Logic with Uninterpreted Functions. In *Proc. DAC'03*, 2003.

[36] N. Shankar and Harald Rueß. Combining shostak theories. Invited paper for Floc'02/RTA'02, 2002.

[37] R. Shostak. A Pratical Decision Procedure for Arithmetic with Function Symbols. *Journal of the ACM*, 26(2):351–360, 1979.

[38] J. P. M. Silva and K. A. Sakallah. GRASP - A new Search Algorithm for Satisfiability. In *Proc. ICCAD'96*, 1996.

[39] R. M. Smullyan. *First-Order Logic*. Springer-Verlag, NY, 1968.

[40] O. Strichman, S. Seshia, and R. Bryant. Deciding separation formulas with SAT. In *Proc. of Computer Aided Verification, (CAV'02)*, LNCS. Springer, 2002.

[41] A. Stump, C. W. Barrett, and D. L. Dill. CVC: A Cooperating Validity Checker. In *Proc. CAV'02*, number 2404 in LNCS. Springer Verlag, 2002.

[42] A. Tacchella. *SAT system description. In *Proc. DL'99*, pages 142–144, 1999.

[43] C. Tinelli. A DPLL-based Calculus for Ground Satisfiability Modulo Theories. In *Proc. JELIA-02*, volume 2424 of *LNAI*, pages 308–319. Springer, 2002.

[44] S. Wolfman and D. Weld. The LPSAT Engine & its Application to Resource Planning. In *Proc. IJCAI*, 1999.

[45] T. Yavuz-Kahveci, M. Tuncer, and T. Bultan. A Library for Composite Symbolic Representation. In *Proc. TACAS2001*, volume 2031 of *LNCS*. Springer Verlag, 2000.

[46] L. Zhang and S. Malik. The quest for efficient boolean satisfiability solvers. In *Proc. CAV'02*, number 2404 in LNCS, pages 17–36. Springer, 2002.