

## 2 Satisfiability Modulo Theories (SMT)

So far:

decision procedures for satisfiability for various fragments of first-order theories;  
often only for ground conjunctions of literals.

Goals:

extend decision procedures efficiently to ground CNF formulas;  
later: extend to non-ground formulas (we will often lose completeness, however).

### 2.1 The DPLL(T) Procedure

Goal:

Given a propositional formula in CNF (or alternatively, a finite set  $N$  of clauses), where the atoms represent ground formulas over some theory  $\mathcal{T}$ , check whether it is satisfiable in  $\mathcal{T}$  (and optionally: output *one* solution, if it is satisfiable).

Assumption:

As in the propositional case, clauses contain neither duplicated literals nor complementary literals.

For propositional DPLL, we have considered partial valuations, i. e., partial mappings from propositional variables to truth values.

A partial valuation  $\mathcal{A}$  corresponds to a set  $M$  of literals that does not contain complementary literals, and vice versa:

$\mathcal{A}(L)$  is true, if  $L \in M$ .

$\mathcal{A}(L)$  is false, if  $\bar{L} \in M$ .

$\mathcal{A}(L)$  is undefined, if neither  $L \in M$  nor  $\bar{L} \in M$ .

We will now consider partial mappings from ground  $\mathcal{T}$ -atoms to truth values (which correspond to sets of  $\mathcal{T}$ -literals).

In order to check whether a (partial) valuation is permissible, we identify the valuation  $\mathcal{A}$  or the set  $M$  with the conjunction of all literals in  $M$ :

The valuation  $\mathcal{A}$  or the set  $M$  is called  $\mathcal{T}$ -satisfiable, if the literals in  $M$  have a  $\mathcal{T}$ -model.

Since the elements of  $M$  can be interpreted both as propositional variables and as ground  $\mathcal{T}$ -formulas, we have to distinguish between two notions of entailment:

We write  $M \models F$  if  $F$  is entailed by  $M$  propositionally. We write  $M \models_{\mathcal{T}} F$  if the ground  $\mathcal{T}$ -formulas represented by  $M$  entail  $F$ .

$M$  is called a  $\mathcal{T}$ -model of  $F$ , if it is  $\mathcal{T}$ -satisfiable and  $M \models F$ .

We write  $F \models_{\mathcal{T}} G$ , if the formula  $F$  entails  $G$  w.r.t.  $\mathcal{T}$ , that is, if every  $\mathcal{T}$ -model of  $F$  is also a model of  $G$ .

## Idea

Naive Approach:

Use DPLL to find a propositionally satisfying valuation.

If the valuation found is  $\mathcal{T}$ -satisfiable, stop; otherwise continue DPLL search.

Note: The DPLL procedure may *not* use “pure literal” checks.

Improvements:

Check already partial valuations for  $\mathcal{T}$ -satisfiability.

If  $\mathcal{T}$ -decision procedure yields explanations, use them for non-chronological backjumping.

If  $\mathcal{T}$ -decision procedure can provide  $\mathcal{T}$ -entailed literals, use them for propagation.

Since  $\mathcal{T}$ -satisfiability checks may be costly, learn clauses that incorporate useful  $\mathcal{T}$ -knowledge, in particular explanations for backjumping. (The procedure is also called CDCL, i. e., “conflict-driven clause learning”.)

## DPLL( $\mathcal{T}$ )

The “DPLL Modulo Theories” procedure is modelled by a transition relation  $\Rightarrow_{\text{DPLL}(\mathcal{T})}$  on a set of states.

States:

- *fail*
- $M \parallel N$ ,

where  $M$  is a *list of annotated literals* and  $N$  is a set of clauses.

Annotated literal:

- $L$ : deduced literal, due to unit propagation.
- $L^d$ : decision literal (guessed literal).

## DPLL(T) Rules from DPLL

Unit Propagate:

$$M \parallel N \cup \{C \vee L\} \Rightarrow_{\text{DPLL}(\mathcal{T})} M L \parallel N \cup \{C \vee L\}$$

if  $C$  is false under  $M$  and  $L$  is undefined under  $M$ .

Decide:

$$M \parallel N \Rightarrow_{\text{DPLL}(\mathcal{T})} M L^d \parallel N$$

if  $L$  is undefined under  $M$ .

Fail:

$$M \parallel N \cup \{C\} \Rightarrow_{\text{DPLL}(\mathcal{T})} \text{fail}$$

if  $C$  is false under  $M$  and  $M$  contains no decision literals.

## Specific DPLL(T) Rules

$\mathcal{T}$ -Backjump:

$$M L^d M' \parallel N \Rightarrow_{\text{DPLL}(\mathcal{T})} M L' \parallel N$$

if  $M L^d M' \models \neg C$  for some  $C \in N$

and if there is some “backjump clause”  $C' \vee L'$  such that

$N \models_{\mathcal{T}} C' \vee L'$  and  $M \models \neg C'$ ,

$L'$  is undefined under  $M$ , and

$L'$  or  $\overline{L'}$  occurs in  $N$  or in  $M L^d M'$ .

$\mathcal{T}$ -Learn:

$$M \parallel N \Rightarrow_{\text{DPLL}(\mathcal{T})} M \parallel N \cup \{C\}$$

if  $N \models_{\mathcal{T}} C$  and each atom of  $C$  occurs in  $N$  or  $M$ .

$\mathcal{T}$ -Forget:

$$M \parallel N \cup \{C\} \Rightarrow_{\text{DPLL}(\mathcal{T})} M \parallel N$$

if  $N \models_{\mathcal{T}} C$ .

$\mathcal{T}$ -Propagate:

$$M \parallel N \Rightarrow_{\text{DPLL}(\mathcal{T})} M L \parallel N$$

if  $M \models_{\mathcal{T}} L$  where  $L$  is undefined in  $M$ , and  $L$  or  $\overline{L}$  occurs in  $N$ .

## DPLL( $\mathcal{T}$ ) Properties

The system DPLL( $\mathcal{T}$ ) consists of the rules Decide, Fail, Unit Propagate,  $\mathcal{T}$ -Propagate,  $\mathcal{T}$ -Backjump,  $\mathcal{T}$ -Learn and  $\mathcal{T}$ -Forget.

**Lemma 2.1** *If we reach a state  $M \parallel N$  starting from  $\emptyset \parallel N$ , then:*

- (1)  *$M$  does not contain complementary literals.*
- (2) *Every deduced literal  $L$  in  $M$  follows from  $\mathcal{T}$ ,  $N$  and decision literals occurring before  $L$  in  $M$ .*

**Proof.** By induction on the length of the derivation. □

**Lemma 2.2** *If no clause is learned infinitely often, then every derivation starting from  $\emptyset \parallel N$  terminates.*

**Proof.** Similar to the propositional case.

**Lemma 2.3** *If  $\emptyset \parallel N \Rightarrow_{\text{DPLL}(\mathcal{T})}^* M \parallel N'$  and there is some conflicting clause in  $M \parallel N'$ , that is,  $M \models \neg C$  for some clause  $C$  in  $N$ , then either Fail or  $\mathcal{T}$ -Backjump applies to  $M \parallel N'$ .*

**Proof.** Similar to the propositional case. □

**Lemma 2.4** *If  $\emptyset \parallel N \Rightarrow_{\text{DPLL}(\mathcal{T})}^* M \parallel N'$  and  $M$  is  $\mathcal{T}$ -unsatisfiable, then either there is a conflicting clause in  $M \parallel N'$ , or else  $\mathcal{T}$ -Learn applies to  $M \parallel N'$ , generating a conflicting clause.*

**Proof.** If  $M$  is  $\mathcal{T}$ -unsatisfiable, then there exists a subsequence  $(L_1, \dots, L_n)$  of  $M$  such that  $\emptyset \models_{\mathcal{T}} \overline{L_1} \vee \dots \vee \overline{L_n}$ . Hence the conflicting clause  $\overline{L_1} \vee \dots \vee \overline{L_n}$  is either in  $M \parallel N'$ , or else it can be learned by one  $\mathcal{T}$ -Learn step. □

**Theorem 2.5** *Consider a derivation  $\emptyset \parallel N \Rightarrow_{\text{DPLL}(\mathcal{T})}^* S$ , where no more rules of the DPLL( $\mathcal{T}$ ) procedure are applicable to  $S$  except  $\mathcal{T}$ -Learn or  $\mathcal{T}$ -Forget, and if  $S$  has the form  $M \parallel N'$  then  $M$  is  $\mathcal{T}$ -satisfiable. Then*

- (1)  *$S$  is fail iff  $N$  is  $\mathcal{T}$ -unsatisfiable.*
- (2) *If  $S$  has the form  $M \parallel N'$ , then  $M$  is a  $\mathcal{T}$ -model of  $N$ .*

## The Solver Interface

The general  $\text{DPLL}(\mathcal{T})$  procedure has to be connected to a “Solver” for  $\mathcal{T}$ , a theory module that performs *at least*  $\mathcal{T}$ -satisfiability checks.

The solver is initialized with a list of all literals occurring in the input of the  $\text{DPLL}(\mathcal{T})$  procedure.

Internally, it keeps a stack  $I$  of theory literals that is initially empty. The solver performs the following operations on  $I$ :

$\text{SetTrue}(L: \mathcal{T}\text{-Literal})$ :

Check whether  $I \cup \{L\}$  is consistent.

If no: return an explanation for  $\bar{L}$ , that is, a subset  $J$  of  $I$  such that  $J \models_{\mathcal{T}} \bar{L}$ .

If yes: push  $L$  on  $I$ .

Optionally: Return a list of literals that are  $\mathcal{T}$ -consequences of  $I \cup \{L\}$  (and have not yet been detected before).

Note: Depending on  $\mathcal{T}$ , detecting (all)  $\mathcal{T}$ -consequences may be very cheap or very expensive.

$\text{Backtrack}(n: \mathbb{N})$ :

Pop  $n$  literals from  $I$ .

$\text{Explanation}(L: \mathcal{T}\text{-Literal})$ :

Return an explanation for  $L$ , that is, a subset  $J$  of  $I$  such that  $J \models_{\mathcal{T}} L$ .

We assume that  $L$  has been returned previously as a result of some  $\text{SetTrue}(L')$  operation. No literal of  $J$  may occur in  $I$  after  $L'$ .

## 2.2 Heuristic Instantiation

DPLL(T) is limited to ground (or existentially quantified) formulas. Even if we have decidability for more than the ground fragment of a theory  $\mathcal{T}$ , we cannot use this in DPLL(T).

Most current SMT implementations offer a limited support for universally quantified formulas by heuristic instantiation.

Goal:

Create potentially useful ground instances of universally quantified clauses and add them to the given ground clauses.

Idea (Detlefs, Nelson, Saxe: Simplify):

Select subset of the terms (or atoms) in  $\forall \vec{x} C$  as “trigger” (automatically, but can be overridden manually).

If there is a ground instance  $C\theta$  of  $\forall \vec{x} C$  such that  $t\theta$  occurs (modulo congruence) in the current set of ground clauses for every  $t \in \text{trigger}(C)$ , add  $C\theta$  to the set of ground clauses (incrementally).

Conditions for trigger terms (or atoms):

- (1) Every quantified variable of the clause occurs in some trigger term (therefore more than one trigger term may be necessary).
- (2) A trigger term is not a variable itself.
- (3) A trigger is not explicitly forbidden by the user.
- (4) There is no larger instance of the term in the formula:  
(If  $f(x)$  were selected as a trigger in  $\forall x P(f(x), f(g(x)))$ , a ground term  $f(a)$  would produce an instance  $P(f(a), f(g(a)))$ , which would produce an instance  $P(f(g(a)), f(g(g(a))))$ , and so on.)
- (5) No proper subterm satisfies (1)–(4).

Also possible (but expensive, therefore only in restricted form): Theory matching

The ground atom  $P(a)$  is not an instance of the trigger atom  $P(x + 1)$ ; it is however equivalent (in linear algebra) to  $P((a - 1) + 1)$ , which is an instance and may therefore produce a new ground clause.

Heuristic instantiation is obviously incomplete

e. g., it does not find the contradiction for  $f(x, a) \approx x$ ,  $f(b, y) \approx b$ ,  $a \not\approx b$

but it is quite useful in practice:

modern implementations: CVC, Yices, Z3.

## 2.3 Local Theory Extensions

Under certain circumstances, instantiating universally quantified variables with “known” ground terms is sufficient for completeness.

Scenario:

$\Sigma_0 = (\Omega_0, \Pi_0)$ : base signature;  
 $\mathcal{T}_0$ :  $\Sigma_0$ -theory.

$\Sigma_1 = (\Omega_0 \cup \Omega_1, \Pi_0)$ : signature extension;  
 $K$ : universally quantified  $\Sigma_1$ -clauses;  
 $G$ : ground clauses.

Assumption: clauses in  $G$  are  $\Sigma_1$ -flat and  $\Sigma_1$ -linear:

- only constants as arguments of  $\Omega_1$ -symbols,
- if a constant occurs in two terms below a  $\Omega_1$ -symbol, then the two terms are identical,
- no term contains the same constant twice below a  $\Omega_1$ -symbol.

Example: Monotonic functions over  $\mathbb{Z}$ .

$\mathcal{T}_0$ : Linear integer arithmetic.  
 $\Omega_1 = \{f/1\}$ .  
 $K = \{ \forall x, y (\neg x \leq y \vee f(x) \leq f(y)) \}$ .  
 $G = \{ f(3) \geq 6, f(5) \leq 9 \}$ .

Observation: If we choose interpretations for  $f(3)$  and  $f(5)$  that satisfy the  $G$  and monotonicity axiom, then it is always possible to define  $f$  for all remaining integers such that the monotonicity axiom is satisfied.

Example: Strictly monotonic functions over  $\mathbb{Z}$ .

$\mathcal{T}_0$ : Linear integer arithmetic.  
 $\Omega_1 = \{f/1\}$ .  
 $K = \{ \forall x, y (\neg x < y \vee f(x) < f(y)) \}$ .  
 $G = \{ f(3) > 6, f(5) < 9 \}$ .

Observation: Even though we can choose interpretations for  $f(3)$  and  $f(5)$  that satisfy  $G$  and the strict monotonicity axiom (map  $f(3)$  to 7 and  $f(5)$  to 8), we cannot define  $f(4)$  such that the strict monotonicity axiom is satisfied.

To formalize the idea, we need partial algebras:

like (usual) total algebras, but  $f_{\mathcal{A}}$  may be a partial function.

There are several ways to define equality in partial algebras (strong equality, Evans equality, weak equality, etc.). Here we use weak equality:

an equation  $s \approx t$  holds w. r. t.  $\mathcal{A}$  and  $\beta$  if both  $\mathcal{A}(\beta)(s)$  and  $\mathcal{A}(\beta)(t)$  are defined and equal or if at least one of them is undefined;

a negated equation  $s \not\approx t$  holds w. r. t.  $\mathcal{A}$  and  $\beta$  if both  $\mathcal{A}(\beta)(s)$  and  $\mathcal{A}(\beta)(t)$  are defined and different or if at least one of them is undefined.

If a partial algebra  $\mathcal{A}$  satisfies a set of formulas  $N$  w. r. t. weak equality, it is called a weak partial model of  $N$ .

A partial algebra  $\mathcal{A}$  embeds weakly into a partial algebra  $\mathcal{B}$  if there is an injective total mapping  $h : U_{\mathcal{A}} \rightarrow U_{\mathcal{B}}$  such that if  $f_{\mathcal{A}}(a_1, \dots, a_n)$  is defined in  $\mathcal{A}$  then  $f_{\mathcal{B}}(h(a_1), \dots, h(a_n))$  is defined in  $\mathcal{B}$  and equal to  $h(f_{\mathcal{A}}(a_1, \dots, a_n))$ .

A theory extension  $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup K$  is called *local*, if for every set  $G$ ,  $\mathcal{T}_0 \cup K \cup G$  is satisfiable if and only if  $\mathcal{T}_0 \cup K[G] \cup G$  has no (partial) model, where  $K[G]$  is the set of instances of clauses in  $K$  in which all terms starting with an  $\Omega_1$ -symbol are ground terms occurring in  $K$  or  $G$ .

If every weak partial model of  $\mathcal{T}_0 \cup K$  can be embedded into a total model, then the theory extension  $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup K$  is local (Sofronie-Stokkermans 2005).

Note: There are many variants of partial models and embeddings corresponding to different kinds of locality.

Examples of local theory extensions:

free functions

constructors/selectors

monotonic functions

Lipschitz functions.