15-414: Bug Catching: Automated Program Verification

Lecture Notes on Satisfiability Modulo Theories

Matt Fredrikson

Carnegie Mellon University
Lecture 18
Tuesday, March 28, 2023

1 Introduction

In the previous lecture we studied decision procedures for two first-order theories: arrays and equality with uninterpreted functions (EUF). Both procedures assumed that the formula to be decided was in the conjunctive, quantifier-free fragment of either theory, which means that the procedures are unable to handle a formula with a disjunction, or a negation over other logical connectives. Additionally, the theory for arrays relied on case analysis to account for the ways in which the indices of read terms might relate to previously-written indices, and there was no obvious way to avoid the worst-case exponential cost of this analysis.

Today we will see how to decide formulas with arbitrary logical structure (i.e., they need not be in a conjunctive fragment), in any first-order theory for which we have a decision procedure capable of handling conjunctive, quantifier-free formulas. In particular, we will go back to DPLL, and see how to combine it with a theory solver, thus inheriting DPLL's heuristic optimizations that are often helpful in avoiding the worst-case cost of case-splitting. This approach is called DPLL(T), where T refers to the first-order theory that we wish to solve.

Learning Goals

1. DPLL(T) combines a conjunctive theory solver and DPLL to decide formulas in a given first-order theory.
2. Just as conflict clauses were important for DPLL, learning *theory lemmas* can dramatically improve the performance of DPLL(T).
3. The Nelson-Oppen procedure extends the approach to combinations of theories, but they must be *stably infinite*, and in some cases *convex*.

2 Review: First-Order Theories

A first-order theory T is defined by the following components.

- It's signature Σ is a set of constant, function, and predicate symbols.
- It's set of axioms $\mathcal{A}$ is a set of closed first-order logic formulae in which only constant, function, and predicate symbols of Σ appear.

Having defined a theory's signature and axioms, we can reason about the same type of properties related to the semantics of a formula as we have been so far, namely validity and satisfiability.

Definition 1 (T-valid)**.** A Σ-formula P is valid in the theory T (T-valid), if *every model* M that satisfies the axioms of T (i.e., $M \models A$ for every $A \in \mathcal{A}$) also satisfies P (i.e., $M \models P$).

Definition 2 (T-satisfiable)**.** Let T be a Σ-theory. A Σ-formula P is T-satisfiable if there *exists a model* M such that $M \models A$ and $M \models P$.

Definition 3 (T-decidable)**.** A theory T is decidable if $T \models P$ is decidable for every Σ-formula. That is, there exists an algorithm that always terminate with "yes" if P is T-valid or with "no" if P is T-invalid.

For example, the **theory of equality with uninterpreted functions** $\mathsf{T_E}$has a signature that consists of a single binary predicate $=$, and all possible constant $(a, b, c, x, y, z, \ldots)$ and function $(f, g, h, \ldots)$ symbols:

$$\Sigma_{\mathsf{E}} : \{=, a, b, c, \ldots, f, g, h, \ldots\}$$

The axioms of $\mathsf{T_E}$define the usual meaning of equality (reflexivity, symmetry, and transitivity), as well as *functional congruence*.

1. $\forall x . x = x$ (reflexivity)
2. $\forall x, y . x = y \to y = x$ (symmetry)
3. $\forall x, y, z . x = y \land y = z \to x = z$ (transitivity)
4. $\forall x, y . x = y \to f(\bar{x}) = f(\bar{y})$ (congruence)

3 DPLL(T) framework

To handle formulas with disjunction, one could always convert to Disjunctive Normal Form (DNF). However, this conversion is usually too expensive and is not the most efficient way of solving disjunctive first-order theories. One of the strengths of the DPLL algorithm is its ability to handle disjunctions efficiently via Boolean Constraint Propagation and clause-learning. We will now see how DPLL can be extended to account for first-order theories via the DPLL(T) framework. This approach is used in nearly all modern SMT solvers.

The key idea behind this framework is to decompose the SMT problem into parts we can deal with efficiently:

- Use SAT solver to cope with the **Boolean structure** of the formula;
- Use dedicate conjunctive **theory solver** to decide satisfiability in the background theory.

3.1 Propositional abstractions

The first insight needed to understand how DPLL(T) works is that it is possible to "abstract" a first-order theory formula P as a propositional formula $B(P)$. The way that we go about accomplishing this is motivated by the way that we ultimately want to make use of DPLL, which only understands propositional formulas. We aim for two key properties.

- If P is satisfiable, then $B(P)$ should be satisfiable also.
- If $B(P)$ is unsatisfiable, then P should be as well.

Why is $B(P)$ an abstraction? Note that if P is *not* satisfiable, then $B(P)$ could be either satisfiable or unsatisfiable. In this sense, $B(P)$ has lost information that was present in P, so it is an abstraction. However, these properties do allow us to determine unsatisfiability in some cases by applying DPLL to $B(P)$ (i.e., $B(P)$ is unsat). In the case where $B(P)$ is satisfiable, we will see that the abstraction, plus DPLL's sat decision, provides enough information to continue making progress on deciding P.

The propositional abstraction of a Σ-formula P recursively. Note that below, l refers to a literal in the first-order theory.

$$\begin{array}{lcl} B(l) & = & p_i \text{ (a fresh propositional variable)} \\ B(\neg P) & = & \neg B(P) \\ B(P \land Q) & = & B(P) \land B(Q) \\ B(P \lor Q) & = & B(P) \lor B(Q) \\ B(P \to Q) & = & B(P) \to B(Q) \end{array}$$

For example, given the formula:

$$P : g(a) = c \land (f(g(a)) \neq f(c) \lor g(a) = d) \land c \neq d$$

The propositional abstraction of P is the following:

$$\begin{aligned} B(P) &= B(g(a) = c) \land B(f(g(a)) \neq f(c) \lor g(a) = d) \land c \neq d) \\ &= B(g(a) = c) \land B(f(g(a)) \neq f(c) \lor g(a) = d)) \land B(c \neq d) \\ &= B(g(a) = c) \land B(f(g(a)) \neq f(c)) \lor B(g(a) = d) \land B(c \neq d) \\ &= P_1 \land (\neg P_2 \lor P_3) \land \neg P_4 \end{aligned}$$

Note that we can also define B^{-1} which maps from the Boolean variables back to the atoms in the original formula. For example, $B^-1(P_1 \wedge P_3 \wedge P_4)$ corresponds to the formula $g(a) = c \wedge g(a) = d \wedge c = d$.

3.2 Combining theory solvers with DPLL

The propositional abstraction provides us with a "lazy" way to solve SMT. Given a Σ-formula P, we can determine its satisfiability by performing the following procedure:

1. Construct the propositional abstraction $B(P)$;
2. If $B(P)$ is unsatisfiable then P is unsatisfiable;
3. Otherwise, get a satisfying assignment M for $B(P)$;
4. Construct $R = \bigwedge_{i=1}^{n} P_i \leftrightarrow M(P_i)$;
5. Send $B^{-1}(R)$ to the T-solver;
6. If T-solver reports that $P \cup B^{-1}(R)$ is satisfiable then P is satisfiable;
7. Otherwise, update $B(P) := B(P) \wedge \neg R$ and return to step 2.

This procedure terminates when: (i) $B(P)$ becomes unsatisfiable which implies that P is also unsatisfiable or (ii) T-solver reports that $P \cup B^{-1}(R)$ is satisfiable which implies that $B(P)$ is satisfiable and that there exists an assignment M that satisfies all axioms in the theory T.

Note that if $P \cup B^{-1}(R)$ is unsatisfiable we cannot terminate since there may be another assignment to $B(P)$ that would make $P \cup B^{-1}(R)$ satisfiable. Therefore, we need to exhaust all assignments for $B(P)$ before deciding that P is unsatisfiable.

On step 7 we add $\neg R$ to $B(P)$ since if we did not, we would get the same assignment M for $B(P)$. We denote $\neg R$ as a **theory conflict clause** that prevents the SAT solver from going down the same path in future iterations.

Example 4. Suppose we want to find if the Σ-formula P is satisfiable:

$$P : g(a) = c \wedge (f(g(a)) \neq f(c) \vee g(a) = d) \wedge c \neq d$$

We start by building its propositional abstraction $B(P)$:

$$B(P) : P_1 \wedge (\neg P_2 \vee P_3) \wedge \neg P_4$$

Table 1 shows the step 1 of the procedure with P and the corresponding propositional abstraction $B(P)$. Next, we query the SAT solver for an assignment to $B(P)$. Assume that the SAT solver returns the following assignment $M = \{P_1, \neg P_2, P_3, \neg P_4\}$. We construct $R = (P_1 \wedge \neg P_2 \wedge P_3 \wedge \neg P_4)$ and send $B^{-1}(R)$ to T-solver. Note that $B^{-1}(R)$ corresponds to:

$$B^{-1}(R) : g(a) = c \wedge f(g(a)) \neq f(c) \wedge g(a) = d \wedge c \neq d$$

Theory solver	SAT solver
$g(a) = c \wedge$ $(f(g(a)) \neq f(c) \vee g(a) = d) \wedge$ $c \neq d$	$P_1 \wedge (\neg P_2 \vee P_3) \wedge \neg P_4$

Table 1: P and $B(P)$.

Theory solver	SAT solver
$g(a) = c \wedge$ $(f(g(a)) \neq f(c) \vee g(a) = d) \wedge$ $c \neq d$	$P_1 \wedge (\neg P_2 \vee P_3) \wedge \neg P_4$ $(\neg P_1 \vee P_2 \vee \neg P_3 \vee P_4)$

Table 2: Updated $B(P)$ after checking that the assignment $M = \{P_1, \neg P_2, P_3, \neg P_4\}$ does not satisfy P

$B^{-1}(R) \cup P$ is unsatisfiable since if $g(a) = d$ and $g(a) = c$ then $c = d$ but P states that $c \neq d$. Therefore, we know that this assignment is not satisfiable but there may exist another assignment M that satisfies P. We update $B(P)$ with $\neg R$ as shown in Table 2 and query the SAT solver for another assignment.

Assume that the SAT solver returns a new assignment $M = \{P_1, P_2, P_3, \neg P_4\}$. We construct $R = (P_1 \wedge P_2 \wedge P_3 \wedge \neg P_4)$ and send $B^{-1}(R)$ to T-solver. Note that in this case B^{-1} corresponds to:

$$B^{-1}(R) : g(a) = c \wedge f(g(a)) = f(c) \wedge g(a) = d \wedge c \neq d$$

We can see that $B^{-1}(R) \cup P$ is unsatisfiable for the same reason as before. We update $B(P)$ with $\neg R$ as shown in Table 3 and perform another query to the SAT solver.

Assume that the SAT solver returns a new assignment $M = \{P_1, \neg P_2, \neg P_3, \neg P_4\}$. We construct $R = (P_1 \wedge \neg P_2 \wedge \neg P_3 \wedge \neg P_4)$ and send $B^{-1}(R)$ to T-solver. Note that in this case B^{-1} corresponds to:

$$B^{-1}(R) : g(a) = c \wedge f(g(a)) \neq f(c) \wedge g(a) \neq d \wedge c \neq d$$

We can see that $B^{-1}(R) \cup P$ is unsatisfiable since $g(a) = c$ but $f(g(a)) \neq f(c)$. We update $B(P)$ with $\neg R$ as shown in Table 4 and observe that $B(P)$ becomes unsatisfiable after adding $\neg R$. Since $B(P)$ is unsatisfiable, we can conclude that P is also unsatisfiable.

Theory solver	SAT solver
$g(a) = c \wedge$ $(f(g(a)) \neq f(c) \vee g(a) = d) \wedge$ $c \neq d$	$P_1 \wedge (\neg P_2 \vee P_3) \wedge \neg P_4$ $(\neg P_1 \vee P_2 \vee \neg P_3 \vee P_4)$ $(\neg P_1 \vee \neg P_2 \vee \neg P_3 \vee P_4)$

Table 3: Updated $B(P)$ after checking that the assignment $M = \{P_1, P_2, P_3, \neg P_4\}$ does not satisfy P.

Theory solver	SAT solver
$g(a) = c\wedge$	$P_1 \wedge (\neg P_2 \vee P_3) \wedge \neg P_4$
$(f(g(a)) \neq f(c) \vee g(a) = d)\wedge$	$(\neg P_1 \vee P_2 \vee \neg P_3 \vee P_4)$
$c \neq d$	$(\neg P_1 \vee \neg P_2 \vee \neg P_3 \vee P_4)$
	$(\neg P_1 \vee P_2 \vee P_3 \vee P_4)$
	unsat

Table 4: Updated $B(P)$ after checking that the assignment $M = \{P_1, \neg P_2, \neg P_3, \neg P_4\}$ does not satisfy P. $B(P)$ becomes unsatisfiable after adding the negation of M.

3.3 Improving DPLL(T) framework

Consider the Σ-formula P defined over $\mathsf{T}_{\mathbb{Z}}$:

$$P : 0 < x \wedge x < 1 \wedge x < 2 \wedge \ldots x < 99$$

The propositional abstraction $B(P)$ is the following:

$$B(P) : P_0 \wedge P_1 \wedge \ldots \wedge P_{99}$$

Note that $B(P)$ has 2^{98} assignments containing $P_0 \wedge P_1$ and none of them satisfies P. The procedure described in the previous section will enumerate all of them one by one and add a blocking conflict clause that only covers a single assignment! A potential solution to this issue is to not treat the SAT solver as a black box but instead incrementally query the theory solver as assignments are made in the SAT solver. If we would perform this integration then we would be able to stop after adding $\{0 < x, x < 1\}$ and would not need to explore the 2^{98} infeasible assignments. This can be done by pushing the T-solver into the DPLL algorithm as follows:

1. After Boolean Constraint Propagation (BCP), invoke the T-solver on the partial assignment;
2. If the T-solver returns unsatisfiable then we can stop the search of the SAT solver and immediately add $\neg R$ to BP;
3. Otherwise, continue as usual until we have a new partial assignment.

Recall the example:

$$P : g(a) = c \wedge (f(g(a)) \neq f(c) \vee g(a) = d) \wedge c \neq d$$

And its propositional abstraction $B(P)$:

$$B(P) : P_1 \wedge (\neg P_2 \vee P_3) \wedge \neg P_4$$

DPLL with being by propagating P_1 and $\neg P_4$ since they are unit clauses. At this point the theory axioms imply more propagations:

$$g(a) = c \rightarrow f(g(a)) = f(c)$$
$$g(a) = c \land c \neq d \rightarrow g(a) \neq d$$

Deciding $\neg P_2$ or P_3 would be wasteful, so we can add the **theory lemmas**:

$$(P_1 \rightarrow P_2)$$
$$(P_1 \land \neg P_3) \rightarrow \neg P_3$$

This procedure is called **theory propagation** and can guarantee that every Boolean assignment is T-satisfiable. However, in practice doing this at every step can be expensive and theory propagation is only applied when it is "likely" (using heuristics) to derive useful implications.