

Lecture 16:

Satisfiability Modulo Theories

17-355/17-665/17-819: Program Analysis

Rohan Padhye

October 30, 2025

* Course materials developed with Jonathan Aldrich and Claire Le Goues

Sometimes we need to reason about formulas

- Verification: verification condition generation turns a Hoare triple into a formula
 - Is that formula valid (i.e. always true – the precondition always implies the postcondition)?
- Symbolic execution: builds path conditions as execution proceeds
 - Is that path condition satisfiable (i.e. potentially true given the right inputs)?
- More applications: test generation, program synthesis, program repair, ...
- Can tools automatically reason about formula validity or satisfiability?

First step: reduce validity to satisfiability

- Formula validity: $\forall x . F(x)$ is true
 - (x stands for the *free variables* of F)
- Equivalent to $\neg \exists x . F(x)$ is false
- Equivalent to $\neg \exists x . \neg F(x)$ is true
 - This is asking whether $\neg F(x)$ is *satisfiable*

Satisfiability *modulo theories*

- Satisfiability is for Boolean formulas
 - Variables, Boolean operators such as $\wedge \vee \neg$
- Verification conditions, path conditions, etc. have other elements
 - Integer, real constants and variables
 - Operations over numbers like $< > + -$
- We can enhance satisfiability checkers to incorporate theories
 - Presburger arithmetic can prove that $2 * x = x + x$
 - The theory of arrays can prove that assigning $x[y]$ to 3 and then looking up $x[y]$ yields 3

Satisfiability (SAT) solving

- Let's start by considering Boolean formulas: variables with $\wedge \vee \neg$
- First step: convert to conjunctive normal form (CNF)
 - A conjunction of disjunctions of (possibly negated) variables
 $(a \vee \neg b) \wedge (\neg a \vee c) \wedge (b \vee c)$
- If formula is not in CNF, we transform it: use De Morgan's laws, the double negative law, and the distributive laws:

$$\begin{aligned}\neg(P \vee Q) &\iff \neg P \wedge \neg Q \\ \neg(P \wedge Q) &\iff \neg P \vee \neg Q \\ \neg\neg P &\iff P \\ (P \wedge (Q \vee R)) &\iff ((P \wedge Q) \vee (P \wedge R)) \\ (P \vee (Q \wedge R)) &\iff ((P \vee Q) \wedge (P \vee R))\end{aligned}$$

SAT solving goal

- Prove that a formula is *satisfiable* by giving a satisfying assignment
 - A map from formula variables to Boolean values
- Example: $X \vee Y$ is satisfiable
 - A satisfying assignment is $X \mapsto \text{true}, Y \mapsto \text{false}$
- Example: $X \wedge \neg X$ is unsatisfiable
 - No satisfying assignment exists

SAT is NP-complete

- Cook-Levin theorem [1970s] proved NP-completeness
 - In NP, because can verify a satisfying assignment by evaluating the formula
 - NP-hard by reduction to polynomial-time acceptance by a nondeterministic Turing machine
- Simple solution approach: try all satisfying assignments
 - Takes $O(2^n)$ time for an n -variable formula

DPLL: Efficient SAT solving in practice

- Developed by Davis, Putnam, Logemann, and Loveland [1961]
 - Still exponential in theory, but on many problems is much faster than trying all assignments
- Key innovation #1: *unit propagation*
 $(b \vee c) \wedge (\textcolor{red}{x}) \wedge (\textcolor{red}{x} \vee c \vee d) \wedge (\neg c \vee d) \wedge (\neg c \vee \neg d \vee \textcolor{red}{x}) \wedge (b \vee d)$
 - In this example, a appears alone. It must be true.

DPLL: Efficient SAT solving in practice

- Developed by Davis, Putnam, Logemann, and Loveland
 - Still exponential in theory, but on many problems is much faster than trying all assignments
- Key innovation #1: *unit propagation*
- Key innovation #2: *pure literal elimination*
$$\cancel{(b \vee c)} \wedge (c \vee d) \wedge (\neg c \vee d) \wedge (\neg c \vee \neg d) \wedge \cancel{(b \vee d)}$$
 - This example is simplified from the previous slide, based on unit propagation
 - Note that b appears only positively. **Setting b to true** can only help us, not hurt us!

DPLL: Efficient SAT solving in practice

- Developed by Davis, Putnam, Logemann, and Loveland
 - Still exponential in theory, but on many problems is much faster than trying all assignments
- Key innovation #1: *unit propagation*
- Key innovation #2: *pure literal elimination*
- When we are stuck, we guess (and backtrack later if necessary)
 - $(c \vee d) \wedge (\neg c \vee d) \wedge (\neg c \vee \neg d)$
 - Let's guess that c is true! Then we get $(d) \wedge (\neg d)$
 - We apply unit propagation to set $d=\text{true}$. Unfortunately the result is so we failed to find a satisfying assignment

(true) \wedge (false)

DPLL: Efficient SAT solving in practice

- Developed by Davis, Putnam, Logemann, and Loveland
 - Still exponential in theory, but on many problems is much faster than trying all assignments
- Key innovation #1: *unit propagation*
- Key innovation #2: *pure literal elimination*
- When we are stuck, we guess (and backtrack later if necessary)
 - $(c \vee d) \wedge (\neg c \vee d) \wedge (\neg c \vee \neg d)$
 - Now let's guess that c is false! Then we get (d)
 - We apply unit propagation to set $d=\text{true}$ and the formula is satisfied

The Full DPLL Algorithm

```
function DPLL( $\phi$ )
  if  $\phi = \mathbf{true}$  then
    return true
  end if
  if  $\phi$  contains a false clause then
    return false
  end if
  for all unit clauses  $l$  in  $\phi$  do
     $\phi \leftarrow \text{UNIT-PROPAGATE}(l, \phi)$ 
  end for
  for all literals  $l$  occurring pure in  $\phi$  do
     $\phi \leftarrow \text{PURE-LITERAL-ASSIGN}(l, \phi)$ 
  end for
   $l \leftarrow \text{CHOOSE-LITERAL}(\phi)$ 
  return  $\text{DPLL}(\phi \wedge l) \vee \text{DPLL}(\phi \wedge \neg l)$ 
end function
```

Heuristic: Apply unit propagation first because it creates more units and pure literals. Pure literal assignment only removes entire clauses.

Try both assignments of the chosen literal. If we assume \vee is short-circuiting, then this implements backtracking.

Practice: Applying DPLL

- Show how DPLL (unit propagation, pure literal elimination, choosing a literal, backtracking) applies to the following formula: $(a \vee b) \wedge (a \vee c) \wedge (\neg a \vee c) \wedge (a \vee \neg c) \wedge (\neg a \vee \neg c) \wedge (\neg d)$

From SAT to SMT

- We'd like to check the satisfiability of formulas like
$$\begin{array}{l} f(f(x) - f(y)) = a \quad \wedge \\ f(0) = a + 2 \quad \wedge \\ x = y \end{array}$$
- Includes arithmetic and the theory of unknown functions
 - E.g. we assume f is some mathematical function
- We may have solvers for each theory, but how can we combine them?
 - Note that separate satisfying assignments for two theories might not be compatible!
- SMT's solution: solve theories separately, use SAT to combine them

The running example is due to Oliveras and Rodriguez-Carbonell

Nelson-Oppen replaces expressions with variables

$$f(f(x) - f(y)) = a \quad \wedge \quad f(0) = a + 2 \quad \wedge \quad x = y$$

Now we have formulas in two theories

- Theory of uninterpreted functions

$$f(e1) = a$$

$$e2 = f(x)$$

$$e3 = f(y)$$

$$f(e4) = e5$$

$$x = y$$

- Theory of arithmetic

$$e1 = e2 - e3$$

$$e4 = 0$$

$$e5 = a + 2$$

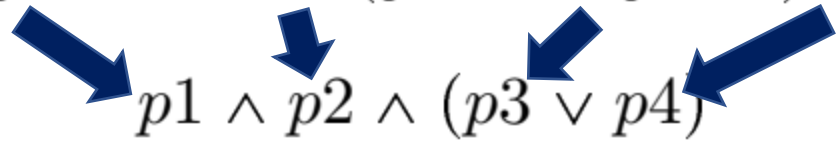
$$x = y$$

- Congruence closure:

for all f, x , and y , if $x = y$ then $f(x) = f(y)$

Theories communicate
using equalities

Combining Theories using DPLL

- Consider the following source formula: $x \geq 0 \wedge y = x + 1 \wedge (y > 2 \vee y < 1)$
- We can convert each subformula to a variable:

$$p1 \wedge p2 \wedge (p3 \vee p4)$$
- Now we solve with DPLL and get a satisfying assignment: $p1, p2, \neg p3, p4$
- We ask the theories if this assignment is feasible
 - The theory of arithmetic says no. $p1, p2$, and $p4$ can't all be true, because $p1$ and $p2$ together imply $y \geq 1$
- We add a clause expressing this and run DPLL again on
$$p1 \wedge p2 \wedge (p3 \vee p4) \wedge (\neg p1 \vee \neg p2 \vee \neg p4)$$
- One satisfying assignment is $p1, p2, p3, \neg p4$
 - We check this against the theories and it succeeds

Details on equality

- Sometimes a theory doesn't tell us an equality, but rather that one of two equalities are true
 - That's fine—we just encode this as a formula and give it to DPLL. For exam, $(e1 = e2 \vee e1 \neq e2) \wedge (e2 = e3 \vee e2 \neq e3)$
- DPLL will choose which equalities are true, and we try those with other theories.

SMT uses a variant of DPLL called DPLL(T)

- T is for Theory
- Differences vs. plain DPLL
 - DPLL(T) doesn't use pure literal elimination
 - Variables may not be independent when they represent a formula – so setting x to true can hurt you, even when x is a pure literal!
 - For example: $(x > 10 \vee x < 3) \wedge (x > 10 \vee x < 9) \wedge (x < 7)$
 - Can't just set $x > 10$ to true, because $x < 7$ will be false
 - DPLL(T) supports adding clauses to the formula
 - To represent knowledge gained from theories, as mentioned above

How to solve arithmetic

- Approach #1: Substitution
 - If we have $y = x+1$, we can eliminate y by substituting it with $x+1$ everywhere
 - High school math!
- Approach #2: Fourier-Motzkin Elimination
 - Applies when we have inequalities rather than equalities
 - Transform all inequalities mentioning x into $A \leq x$ or $x \leq B$
 - Then eliminate x , replacing the inequalities with $A \leq B$
 - Detail: if there are multiple inequalities, we conjoin the cross product of them

Modern tooling: SMT-lib

SMT-LIB

THE SATISFIABILITY MODULO THEORIES LIBRARY

Theories

SMT-LIB logs refer to one or more theories below. Click on a theory's name to see its declaration in Version 2.x of the format.

ArraysEx

Functional arrays with extensionality

FixedSizeBitVectors

Bit vectors with arbitrary size

Core

Core theory, defining the basic Boolean operators

FloatingPoint

Floating point numbers

Ints

Integer numbers

Reals

Real numbers

Reals_Ints

Real and integer numbers

Strings

Unicode character strings and regular expressions

SMT-COMP

The International
Satisfiability Modulo
Theories (SMT)
Competition.

[GitHub](#)

[Home](#)
[Introduction](#)
[Benchmark Submission](#)
[Publications](#)
[SMT-LIB](#)
[Previous Editions](#)

SMT-COMP 2021

[Rules](#)
[Benchmarks](#)
[Tools](#)
[Specs](#)
[Parallel & Cloud Tracks](#)
[Participants](#)
[Results](#)
[Slides](#)

SAT Performance

Solver	Correct Score	Time Score	Division
cvc5	0.10587024	0.00161216	Equality+LinearArith
UltimateEliminator+MathSAT	0.08416589	0.00358447	Equality+NonLinearArith
Vampire	0.02936727	0.00424783	Equality
cvc5	0.00616228	0.00599641	QF_NonLinearIntArith
Yices2-QS	0.00553133	0.00393864	Arith
cvc5	0.00262204	0.00188031	QF_NonLinearRealArith
cvc5	0.00157784	0.00059486	QF_Equality
cvc5	0.00145186	0.0022273	QF_LinearIntArith
cvc5	0.00125507	0.00027314	Bitvec
Bitwuzla	0.00093358	0.00069896	QF_Bitvec
cvc5	0.00093134	-0.00090887	QF_Equality+NonLinearArith
cvc5	0.00055932	-0.00228172	QF_FPArith
cvc5	0.00013178	0.00015434	QF_Equality+LinearArith
Yices2	0.00010454	0.00020464	QF_Equality+Bitvec
Yices2	9.601e-05	0.00035558	QF_LinearRealArith

UNSAT Performance

Solver	Correct Score	Time Score	Division
cvc5	0.05633672	0.03589274	Equality+NonLinearArith
Yices2	0.02120632	0.0072311	QF_NonLinearIntArith
cvc5	0.01061534	0.04760987	Equality+LinearArith

Modern tooling: Z3 w/ SMT-lib

[Z3 Online Demonstrator](#) [SMT-LIB 2 Standard](#) [Z3 sources](#)

Input

SMT-LIB 2 script

```
; Variable declarations
(declare-fun a () Int)
(declare-fun b () Int)
(declare-fun c () Int)

; Constraints
(assert (> a 0))
(assert (> b 0))
(assert (> c 0))
(assert (= (+ (* a a) (* b b)) (* c c)))

; Solve
(check-sat)
(get-model)
```

Reset **Execute**

Output

Z3 output

```
sat
(model
  (define-fun c () Int
    15)
  (define-fun b () Int
    9)
  (define-fun a () Int
    12)
)
```

Summary

Command	z3 -in -T:30
Execution time	0.083 s
Version	z3-4.4.1