SMT Solvers

- An SMT (satisfiability modulo theory) solver is a tool that decides satisfiability of formulas in combination of various first-order theories.
- SMT solvers are generalizations of SAT solvers.
- Common first-order theories SMT solvers reason about:
  - Theory of equality
  - Theory of rationals
  - Theory of bitvectors
  - Theory of arrays
  - Theory of integers

Applications of SMT Solvers

- SMT solving: active research topic and lots of clients.
- Many applications: software verification, programming languages, test case generation, planning and scheduling, ...
- Slogan: “Whatever SAT solvers can do, SMT solvers can do better!”
- SMT solvers generalize SAT solvers; they can handle much richer theories than propositional logic.

Existing SMT Solvers

- Many existing off-the-shelf SMT solvers:
  - Yices (SRI)
  - Z3 (Microsoft Research)
  - CVC4 (NYU, U Iowa)
  - MathSAT (U Trento, Italy)
  - STP (Stanford)
  - Barcelogic (Catalonia, Spain)
- Annual competition SMT-COMP between solvers; tools ranked in various categories.
- All of these SMT solvers have many users – at least two dozen projects at Microsoft that rely on the Z3.

Overview

- Plan for today: Learn about how SMT solvers actually work.
- We’ve already learned about some aspects of SMT solvers.
- Already know how to decide satisfiability of several first-order theories and how to combine them using Nelson-Oppen technique.
- Big missing piece: How to handle boolean structure of SMT formulas including disjunctions (without conversion to DNF).

DPLL(T) Overview

- Key idea underlying SMT solvers: Combine DPLL algorithm for SAT solving with theory solver.
- Theory solver: Decision procedure for checking satisfiability in conjunctive fragment.
- This architecture where we combine DPLL-based SAT solvers with theory solvers is known as DPLL(T) framework.
- Called DPLL(T) because we combine DPLL algorithm with solver for theory T.
- However, T can be a combination theory, such as $\mathcal{T}_1 \cup \mathcal{T}_2$.
- As before, solver for $\mathcal{T}_1 \cup \mathcal{T}_2$ is obtained by using Nelson-Oppen technique.
Main Idea of DPLL(T)

- In the DPLL(T) framework, SAT solver handles boolean structure of formula
- For this, treat each atomic formula as a propositional variable ⇒ resulting formula called boolean abstraction
- Now, use SAT solver to decide satisfiability of boolean abstraction

Main Idea of DPLL(T), cont.

- If there is no satisfying assignment to boolean abstraction, formula is UNSAT
- If there is satisfying assignment to boolean abstraction, formula may not be SAT
- Main job of the theory solver is to check whether assignments made by SAT solver is satisfiable modulo theory
- If SAT solver finds assignment that is consistent with theory, then SMT formula is satisfiable

SMT Formulas and Boolean Abstraction

- SMT formula in theory T formed according to CFG:
  \[ F := a_F^1 \mid F_1 \land F_2 \mid F_1 \lor F_2 \mid \neg F \]
- For each SMT formula, define a bijective function \( B \), called boolean abstraction function, that maps SMT formula to overapproximate SAT formula
- Function \( B \) defined inductively as follows:
  \[
  B(a_F^1) = b_i \\
  B(F_1 \land F_2) = B(F_1) \land B(F_2) \\
  B(F_1 \lor F_2) = B(F_1) \lor B(F_2) \\
  B(\neg F) = \neg B(F_1)
  \]

Example

- What is the boolean abstraction of this formula?
  \[ F : x = z \land ((y = z \land x = z + 1) \lor \neg(x = z)) \]
- \( B(F) = b_1 \land ((b_2 \land b_3) \lor \neg b_1) \)
- Boolean abstraction is also called boolean skeleton
- Since \( B \) is a bijective function, \( B^{-1} \) also exists
- What is \( B^{-1}(b_2 \lor b_3) \)? \( y = z \lor \neg(x = z) \)

Boolean Abstraction as Overapproximation

- Observe: The boolean abstraction constructed this way overapproximates satisfiability of the formula
- Is this formula satisfiable? No
  \[ F : x = z \land ((y = z \land x = z + 1) \lor \neg(x = z)) \]
- Boolean abstraction: \( B(F) = b_1 \land ((b_2 \land b_3) \lor \neg b_1) \)
- Is this satisfiable? Yes
- What is a sat assignment? \( A = b_1 \land b_2 \land b_3 \)
- What is \( B^{-1}(A) \)? \( x = z \land y = z \land x = z + 1 \)
- Is \( B^{-1}(A) \) satisfiable? No

SMT Solving: Simplest Version

- In simplest version of SMT solvers, construct boolean abstraction \( B(F) \) of SMT formula \( F \)
- If \( B(F) \) is unsat, return unsat
- If \( B(F) \) is sat, get sat assignment \( A \) (conjunction of propositional literals)
- Construct \( B^{-1}(A) \); this is conjunction of atomic \( T \)-formulas
- Query \( T \)-solver for satisfiability of \( B^{-1}(A) \)
SMT Solving: Simplest Version, cont

- If $T$-solver decides $B^{-1}(A)$ is sat, return SAT
- Why? Because we found an assignment that (i) both satisfies boolean structure, and (ii) consistent with theory axioms
- If $B^{-1}(A)$ is unsat, does this mean original formula is UNSAT?
- No b/c might be other ways of satisfying boolean structure
- In this case, construct new boolean abstraction $B(F) \land \neg A$
- Repeat until we find assignment consistent with theory or until boolean abstraction is unsat

Example

- Consider example from before:
  \[ F : x = z \land ((y = z \land x = z + 1) \lor \neg(x = z)) \]
  \[ B(F) : b_1 \land ((b_2 \land b_3) \lor \neg b_1) \]
  Sat assignment to $B(F)$: $A : b_1 \land b_2 \land b_3$
- $B^{-1}(A)$ is unsat
- What is new boolean abstraction?
  \[ (b_1 \land ((b_2 \land b_3) \lor \neg b_1)) \land \neg(b_1 \land b_2 \land b_3) \]
- Is this formula SAT? No, thus original formula UNSAT

SMT Solving, Simplest Version: Correctness

- If $B^{-1}(A)$ is unsat, construct new abstraction as $B(F) \land \neg A$
- Does $B(F) \land \neg A$ still overapproximate satisfiability?
  \[ \text{Yes because since } B^{-1}(A) \text{ is unsat } B^{-1}(\neg A) \text{ is valid} \]
- Thus, $F \land B^{-1}(\neg A)$ is equivalent to $F$
- Hence, $B(F \land B^{-1}(\neg A))$ (i.e., $B(F) \land \neg A$) still overapproximates satisfiability
- Formulas such as $\neg A$ that are $T$-valid are called theory conflict clauses

SMT Solving, Simplest Version: Termination

- Approach is sound, but is it guaranteed to terminate? Yes
- Suppose SAT solver gives assignment $A$ s.t. $B^{-1}(A)$ is unsat
  \[ \text{We’ll never obtain same assignment } A \text{ again because formula next time is } B(F) \land \neg A \]
  \[ \text{There are finitely many satisfying assignments to boolean abstraction, and we get different sat assignment every time} \]
- Thus, we’ll eventually either find assignment consistent with theory $\Rightarrow$ SAT
- Or all satisfying assignments contradict theory axioms $\Rightarrow$ UNSAT

Shortcoming of This Approach

- So far, we just add negation of current assignment as theory conflict clause
- Unfortunately, conflict clauses obtained this way are too weak
- Suppose $A$ is a conjunction of 100 literals such that
  \[ B^{-1}(A) = x = y \land x > y \land a_1 \land a_2 \land \ldots \land a_{98} \]
- Theory conflict clause $\neg A$ prevents exact same assignment
- But it doesn’t prevent many other bad assignments involving $x = y \land x > y$ such as:
  \[ B^{-1}(A) = x = y \land x > y \land a_1 \land a_2 \land \ldots \land \neg a_{98} \]
- In fact, there are $2^{98}$ unsat assignments containing $x = y \land x > y$ but $\neg A$ prevents only one of them!

SMT solving, Improvement #1

- Suppose SAT solver makes assignment $A$ s.t. $B^{-1}(A)$ is unsat
  Rather than adding $\neg A$ as a conflict clause, better idea is to find an unsatisfiable core of $B^{-1}(A)$
  \[ \text{An unsatisfiable core } C \text{ of } A \text{ contains a subset of atoms in } A \text{ and } B^{-1}(C) \text{ is still unsatisfiable.} \]
- Ideally, we would like to find the minimal unsatisfiable core
  \[ \text{Minimal unsatisfiable core } C^* \text{ has property that if you drop any single atom of } C^*, \text{ result is satisfiable} \]
  \[ \text{What is a minimal unsat core of } x = y \land x > y \land a_1 \land a_2 \land \ldots \land a_{98}? \text{ x = y \land x > y} \]
Computing Minimal Unsat Core

- How can we compute **minimal unsat core** of conjunctive $T$ formula without modifying theory solver?
- Let $\phi$ be original unsatisfiable conjunct
- Drop one atom from $\phi$, call this $\phi'$
- If $\phi'$ is still **unsat**, $\phi' := \phi'$
- Repeat this for every atom in $\phi$
- Clearly, resulting $\phi$ is **minimal unsat core** of original formula

SMT Solving Using Unsat Cores

- Given formula $F$, construct boolean abstraction $B(F)$
- Use SAT solver to decide if $B(F)$ is unsat; if so $F$ also **unsat**
- Otherwise, get satisfying assignment $A$ to $B(F)$
- Query theory solver if $B^{-1}(A)$ is sat; if so $F$ is **sat**
- Otherwise, compute **minimal unsat core** $C$ of $B^{-1}(A)$
- Use $\neg C$ as theory conflict clause
- i.e., construct new boolean abstraction as $B(F \land \neg C)$
- Repeat until we decide sat or unsat

Motivation for Integration with DPLL

- Consider **very large** formula $F$ containing $x = y$ and $x > y$ with corresponding boolean variables $b_1$ and $b_2$
- Also, suppose $B(F)$ contains hundreds of boolean variables
- As soon as sat solver makes assignment $b_1 = T$, $b_2 = T$, we are doomed because this is unsatisfiable in theory
- Thus, no need to continue with SAT solving after this bad partial assignment
- Idea: Don’t use SAT solver as "blackbox"
- Instead, integrate theory solver right into the DPLL algorithm

Example

- Let’s compute minimal unsat core of
  $\phi : x = y \land f(x) + z = 5 \land f(x) \neq f(y) \land y \leq 3$
- Drop $x = y$ from $\phi$. Is result unsat? **no, so keep $x = y$**
- Drop $f(x) + z = 5$. Is result unsat? **yes, so drop $f(x) + z = 5$**
- New formula: $\phi : x = y \land f(x) \neq f(y) \land y \leq 3$
- Drop $f(x) \neq f(y)$. Is result unsat? **no, keep $f(x) \neq f(y)$**
- Finally, drop $y \leq 3$. Is result unsat? **yes, drop $y \leq 3$**
- What is minimal unsat core? $x = y \land f(x) \neq f(y)$

Discussion

- This strategy is much better than simplest strategy where we add $B^{-1}(A)$ as theory conflict clause.
- Using simple strategy, we block just one assignment
- Using minimal unsat cores, we block many assignments using one theory conflict clause
- However, our strategy still not ideal because it waits for full assignment to boolean abstraction to generate conflict clause

DPLL-Based SAT Solver Architecture

- **Idea**: Integrate theory solver right into this SAT solving loop!
**DPLL(T) Framework**

Suppose SAT solver has made assignment in Decide step and performed BCP
- If no conflict detected, immediately invoke theory solver
- Specifically, suppose $A$ is current partial assignment to boolean abstraction
- Use theory solver to decide if $B^{-1}(A)$ is unsat
- If $B^{-1}(A)$ unsat, add theory conflict clause $\neg A$ to clause database
- Or better, add negation of unsat core of $A$ to clause database

---

**Theory Propagation**

- What we described so far is sufficient to solve SMT formula, but we can be even more clever!
- Suppose original formula contains literals $x = y, y = z, x \neq z$ with corresponding boolean variables $b_1, b_2, b_3$
- Suppose SAT solver makes partial assignment $b_1 : T, b_2 : T$
- In next Decide step, free to assign $b_3 : T$ or $b_3 : \bot$
- But assignment $b_3 : T$ is stupid b/c will lead to conflict in $T$

---

**Theory Propagation Lemma, cont**

- Idea: Theory solver can communicate which literals are implied by current partial assignment
- In our example, $\neg x \neq z$ implied by current partial assignment $x = y \land y = z$
- Thus, can safely add $b_1 \land b_2 \rightarrow b_3$ to clause database
- These kinds of clauses implied by theory are called theory propagation lemmas

---

**DPLL(T) Framework**

After adding theory propagation lemma, continue doing BCP
- Adding theory propagation lemmas prevents bad assignments to boolean abstraction

---

**DPLL(T) Framework**

Combination of DPLL-based SAT solver and decision procedure for conjunctive $T$ formula called DPLL(T) framework
- As before, AnalyzeConflict decides what level to backtrack to
- Or better, add negation of unsat core of $A$ to clause database

---

**Theory Propagation**

Suppose SAT solver makes partial assignment $b_1 : T, b_2 : T$
- In next Decide step, free to assign $b_3 : T$ or $b_3 : \bot$
- But assignment $b_3 : T$ is stupid b/c will lead to conflict in $T$

---

**DPLL(T) Framework**

- Decide
- BCP
- Theory

---

**DPLL(T) Framework**

- Decide
- BCP
- Theory
Inferring Theory Propagation Lemmas

- How do we obtain theory propagation lemmas?
- Option #1: Treat theory solver as blackbox, query whether particular literal \( a \) is implied by current partial assignment?
- Option #2: Modify theory solver so that it can figure out implied literals
- Second option is considered more efficient, but have to figure out how to do this for each different theory

Which Theory Propagation Lemmas to Add

- Which theory propagation lemmas do we add?
- Option #1: Figure out and add all literals implied by current partial assignment; called **exhaustive theory propagation**
- Option #2: Only figure out literals “obviously” implied by current partial assignment
- Exhaustive theory propagation can be very expensive; second option considered preferable
- There isn’t much of a science behind which literals are “obviously” implied
- Solvers use different strategies to obtain simple-to-find implications

Summary

- SMT solvers decide satisfiability in boolean combinations of different theories
- Instead of converting to DNF, they handle boolean structure using SAT solving techniques
- Most common approach is to construct boolean abstraction and lazily infer **theory conflict clauses**
- To do this, can either consider SAT solver as blackbox or can integrate with it
- Latter strategy considered superior and known as **DPLL(\(T\)) framework**