Motivation

- In previous lectures, we looked at decision procedures for conjunctive formulas in various first-order theories
- This lecture: How to handle boolean structure when deciding satisfiability modulo theories
- In practice, cannot convert to DNF because causes exponential blow-up in formula size
- SMT (satisfiability modulo theory) solvers use clever techniques to handle boolean structure

SMT solvers

- Key idea underlying SMT solvers: Combine theory solvers with SAT solvers
  - Theory solver: Decision procedure for checking satisfiability in conjunctive fragment
  - SAT solver handles boolean structure, and theory solver handles theory-specific reasoning

The Basic Idea

- To use SAT solver, we construct a propositional formula, called boolean abstraction, that overapproximates satisfiability
- If boolean abstraction is UNSAT, we are done ⇒ also unsat modulo theory
- If boolean abstraction is SAT, doesn’t necessarily mean original formula is SAT
  - Use theory solver to check if assignment returned by SAT solver is satisfiable modulo theory
- If not, add additional boolean constraints (called theory conflict clauses) to guide the search for an assignment that is satisfiable modulo theory

Boolean Abstraction

- SMT formula in theory \( T \) formed according to CFG:
  \[
  F ::= \alpha T \mid F_1 \land F_2 \mid F_1 \lor F_2 \mid \neg F
  \]
- 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:
  \[
  B(\alpha T) = b (b \text{ fresh})
  B(F_1 \land F_2) = B(F_1) \land B(F_2)
  B(F_1 \lor F_2) = B(F_1) \lor B(F_2)
  B(\neg F) = \neg B(F_1)
  \]

Example

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

- 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 $x = y \land x \neq y$ 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 $x = y \land x \neq y$ but $\neg A$ prevents only one of them.

Improvement to Off-line SMT

- Rather than adding $\neg A$ as a conflict clause, better idea is to find an unsatisfiable core of $B^{-1}(A)$.
- Given a set $S$ of clauses, an unsat core of $S'$ is a subset $S''$ such that $S''$ is also unsat.
- Ideally, we would like to find the minimal unsatisfiable core.
- Minimal unsatisfiable core $C^*$ has property that if you drop any single atom of $C^*$, result is satisfiable.
- What is a minimal unsatisfiable core of $x = y \land x < y$?
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 Improved Off-line Version

```plaintext
OfflineSMT(φ) {  ψ := B(φ)
  while(true) {
    A := CDCL(φ)
    if(A = ⊥) return UNSAT;
    res := ...  
  }
```

Example

- Let’s compute minimal unsat core of
  \[
  φ: x = y ∧ f(x) + z = 5 ∧ f(x) ≠ f(y) ∧ y ≤ 3
  \]
- Drop $x = y$ from $φ$. Is result unsat?
- Drop $f(x) + z = 5$. Is result unsat?
- New formula: $φ: x = y ∧ f(x) ≠ f(y) ∧ y ≤ 3$
- Drop $f(x) ≠ f(y)$. Is result unsat?
- Finally, drop $y ≤ 3$. Is result unsat?
- So, minimal unsat core is $x = y ∧ f(x) ≠ f(y)$

Motivation for On-line SMT

- This strategy is much better than simple strategy where we add $¬A$ as theory conflict clause.
- But still need to wait for full assignment from the SAT solver, which can be problematic
- Consider very large formula $F$ containing $x = y$ and $x < y$ with corresponding boolean variables $b_1$ and $b_2$
- 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

On-line SMT

- Idea: Don’t use SAT solver as “blackbox”
- Integrate theory solver right into the CDCL
- In other words, theory conflict clauses become another kind of conflict clause that SAT solvers already learn...

DPLL-Based SAT Solver Architecture

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

Theorem Propagation Lemma, cont

- Idea: Theory solver can communicate which literals are implied by current partial assignment
  - In our example, \( \neg x < z \) 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

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 \( x = y, y = z, x < z \) with corresponding boolean variables \( b_1, b_2, b_3 \)
- Suppose SAT solver makes partial assignment \( b_1 : \top, b_2 : \top \)
- In next Decide step, free to assign \( b_3 : \bot \) or \( b_3 : \top \)
- But assignment \( b_3 : \top \) is stupid b/c will lead to conflict in \( T \)

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

DPLL(T) Framework

- Combination of DPLL-based SAT solver and decision procedure for conjunctive \( T \) formula called DPLL(T) framework

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

- Adding theory propagation lemmas prevents bad assignments to boolean abstraction
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

SMT Solvers Today

- All competitive SMT solvers today are based on the on-line version
- Many existing off-the-shelf SMT solvers: Z3, CVC3, Yices, MathSAT, etc.
- Lots of on-going research on SMT, esp. related to quantifier support
- Annual competition SMT-COMP between solvers; tools ranked in various categories

Summary

- SMT solvers decide satisfiability in boolean combinations of different theories
- Instead of converting to DNF, they handle boolean structure using SAT solving techniques
- Competitive solvers are based on \( \text{DPLL}(T) \) framework