Lecture 7: Satisfiability Modulo Theories

Yu Feng

Spring 2024

Summary of previous lecture

- Applications of SAT (Max SAT, Partial Max SAT, etc.)

Outline of this lecture

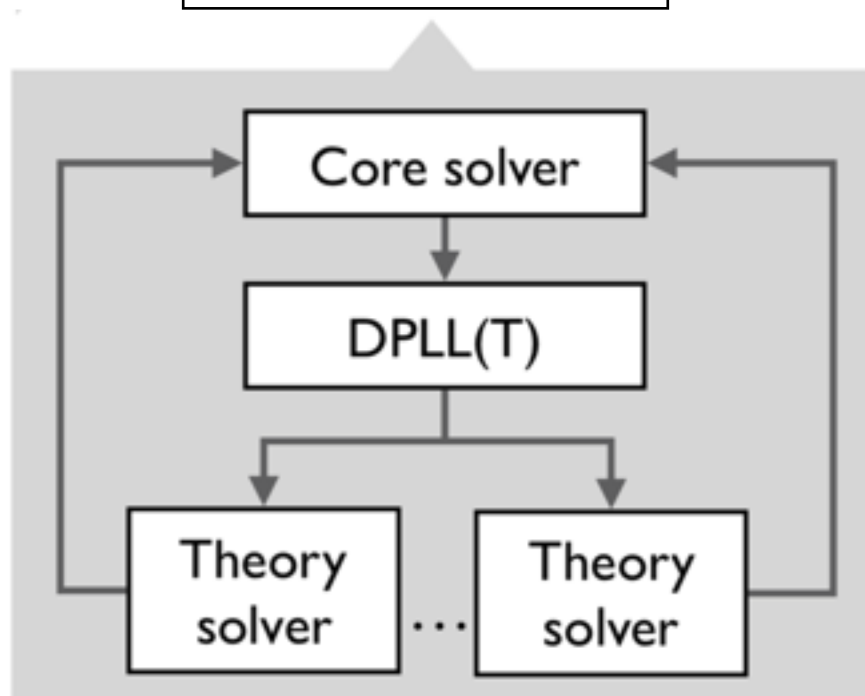
- Introduction to Satisfiability Modulo Theories (SMT)
- Syntax and semantics of first-order logic
- Overview of key theories

Satisfiability Modulo Theories

Theories: $x = g(y)$ $2x + y \leq 5$ $a[i] = x$ $(b >> 2) = c$

First order logic

SMT solver



Syntax of First-Order Logic (FOL)

Logical symbols

- Connectives: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$
- Parentheses: ()
- Quantifiers: $\forall, \exists$

quantifier-free fragment of FOL.

Non-logical symbols

- Constants: x,y,z
- N-ary functions: f, g
- N-ary predicates: p, q
- Variables: u,v,w

quantifier-free **ground** formulas.

Syntax of First-Order Logic (FOL)

Logical symbols

- Connectives: ¬,∧,∨,→,↔
- Parentheses: ()

Non-logical symbols

- Constants: x,y,z
- N-ary functions: f, g
- N-ary predicates: p, q

- A **term** is a constant or an n-ary function with n terms.
- An **atom** is ⊤, ⊥, or an n-ary predicate applied to n terms.
- A **literal** is an atom or its negation.
- A (quantifier-free ground) **formula** is a literal or the application of logical connectives to formulas.

isPrime(x) → ¬ isInteger(sqrt(x))

Semantics of FOL $\langle U, I \rangle$

Universe

- A non-empty set of values
- Finite or (un)countably infinite

Interpretation

- Maps a constant symbol c to an element of U: I[c] in U
- Maps an n-ary function symbol f to a function $f_I : U^n \rightarrow U$
- Maps an n-ary predicate symbol p to an n-ary relation $p \subseteq U^n$

U = {☀, ☁}

I[x] = ☀

I[y] = ☁

I[f] = {☀ $\mapsto$ ☁, ☁ $\mapsto$ ☀}

I[p] = {⟨☀,☀⟩, ⟨☀,☁⟩}

$\langle U, I \rangle \models p(f(y), f(f(x)))$?

Satisfiability and validity of FOL

F is **satisfiable** iff $M \models F$ for some structure $M = \langle U, I \rangle$.

F is **valid** iff $M \models F$ for all structures $M = \langle U, I \rangle$.

Duality of satisfiability and validity:

F is valid iff $\neg F$ is unsatisfiable.

First-order theories

Signature Σ_T

- Set of constant, predicate, and function symbols

Set of T-models

- One or more (possibly infinitely many) models that fix the interpretation of the symbols in Σ_T
- Can also view a theory as a set of axioms over Σ_T (and T-models are the models of the theory axioms)

.A formula F is **satisfiable modulo T** iff $M \models F$ for someT-model M.

A formula F is **valid modulo T** iff $M \models F$ for all T-models M.

Common theories

Equality (and uninterpreted functions)

- $x = g(y)$

Fixed-width bitvectors

- $(b >> 1) = c$

Linear arithmetic (over R and Z)

- $2x + y \leq 5$

Arrays

- $a[i] = x$

Theory of equality with uninterpreted functions

- **Signature: {=, x, y, z, ..., f, g, ..., p, q, ...}**
  - The binary predicate = is *interpreted*.
  - All constant, function, and predicate symbols are *uninterpreted.*
- **Axioms**
  - $\forall x.\ x = x$
  - $\forall x,y.\ x=y \rightarrow y=x$
  - $\forall x,y,z.\ x=y \wedge y=z \rightarrow x=z$
  - $\forall x_1,...,x_n,y_1,...,y_n.(x_1 = y_1 \wedge ... \wedge x_n = y_n) \rightarrow (f(x_1,...,x_n) = f(y_1,\ldots,y_n))$
  - $\forall x_1,...,x_n,y_1,...,y_n.(x_1 = y_1 \wedge ... \wedge x_n = y_n) \rightarrow (p(x_1,...,x_n) \leftrightarrow p(y_1,...,y_n))$
- **Deciding $T_=$**
  - Conjunctions of literals modulo $T_=$ is decidable in polynomial time.

$T_=$ example: checking program equivalence

```
int fun1(int y) {
  int x, z;
  z = y;
  y = x;
  x = z;
  return x * x;
}

int fun2(int y) {
  return y * y
}
```

A formula that is unsatisfiable iff programs are equivalent:

$(z_1 = y_0 \wedge y_1 = x_0 \wedge x_1 = z_1 \wedge r_1 = x_1 * x_1) \wedge$
$(r_2 = y_0 * y_0) \wedge$
$\neg(r_2 = r_1)$

Using 32-bit integers, a SAT solver fails to return an answer in 5 min.

$T_=$ example: checking program equivalence

```
int fun1(int y) {
 int x, z;
 z = y;
 y = x;
 x = z;
 return x * x;
}

int fun2(int y) {
 return y * y
}
```

A formula that is unsatisfiable iff programs are equivalent:

$(z_1 = y_0 \wedge y_1 = x_0 \wedge x_1 = z_1 \wedge r_1 = mul(x_1, x_1)) \wedge$
$(r_2 = mul(y_0, y_0)) \wedge$
$\neg(r_2 = r_1)$

Using T=, an SMT solver proves unsatisfiability in a fraction of a second.

$T_=$ example: checking program equivalence

```
int fun1(int y) {
 int x, y;
  x = x ^ y;
 y = x ^ y;
 x = x ^ y;
 return x * x;
}
```

Is the uninterpreted function abstraction going to work?

- No, we need the theory of fixed-width bitvectors to reason about ^ (xor).

```
int fun2(int y) {
  return y * y
}
```

Theory of fixed-width bitvector

Signature

- Fixed-width words modeling machine ints, longs, ...
- Arithmetic operations: bvadd, bvsub, bvmul, ...
- Bitwise operations: bvand, bvor, bvnot, ...
- Comparison predicates: bvlt, bvgt, …
- Equality: =
- Expanded with all constant symbols: x, y, z, ...

Deciding T_{BV}

- NP-complete.

Theory of linear integer and real

Signature

- Integers (or reals)
- Arithmetic operations: multiplication by an integer (or real) number, +, -.
- Predicates: =, $\leq$.
- Expanded with all constant symbols: x, y, z, ...

Deciding T_{LIA} and T_{LRA}

- NP-complete for linear integer arithmetic (LIA). Polynomial time for linear real arithmetic (LRA).
- Polynomial time for difference logic (conjunctions of the form $x - y \leq c$, where c is an integer or real number).

LIA example: compiler optimization

```
for (i=1; i<=10; i++) {
  a[j+i] = a[j];
}
```

```
int v = a[j];
for (i=1; i<=10; i++) {
  a[j+i] = v;
}
```

A LIA formula that is unsatisfiable iff this transformation is valid:

$(i \geq 1) \wedge (i \leq 10) \wedge$
$(j + i = j)$

Polyhedral model

https://en.wikipedia.org/wiki/Polytope_model

Theory of arrays

Signature

- Array operations: read, write
- Equality: =
- Expanded with all constant symbols: x, y, z, ...

Axioms

- $\forall a, i, v.\ read(write(a, i, v), i) = v$
- $\forall a, i, j, v.\ \neg(i = j) \rightarrow (read(write(a, i, v), j) = read(a, j))$
- $\forall a, b.\ (\forall i.\ read(a, i) = read(b, i)) \rightarrow a = b$

Deciding T_A

- Satisfiability problem: NP-complete.
- Used in many software verification tools to model memory.

TODOs by next lecture

- The 3rd homework will be out
- Start to work on the proposal for your final project