Computer-Aided Reasoning for Software

CSE507

Satisfiability Modulo Theories

Emina Torlak
emina@cs.washington.edu

Today

Last lecture

- Practical applications of SAT and the need for a richer logic

Today

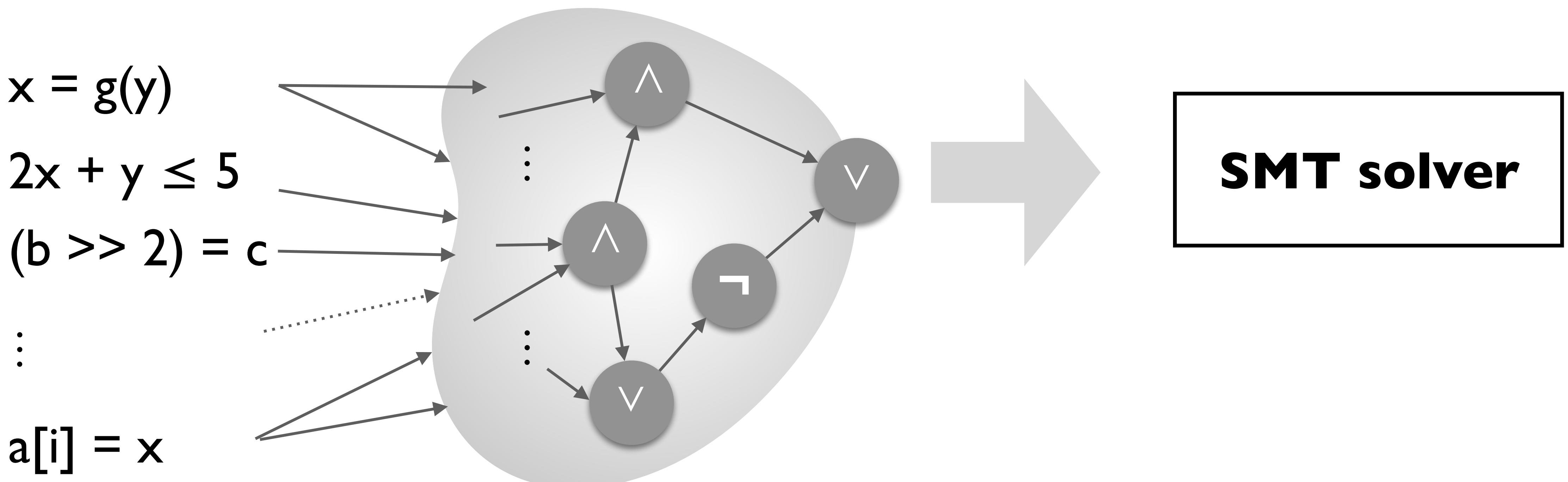
- Introduction to Satisfiability Modulo Theories (SMT)
- Syntax and semantics of (quantifier-free) first-order logic
- Overview of key theories

Reminder

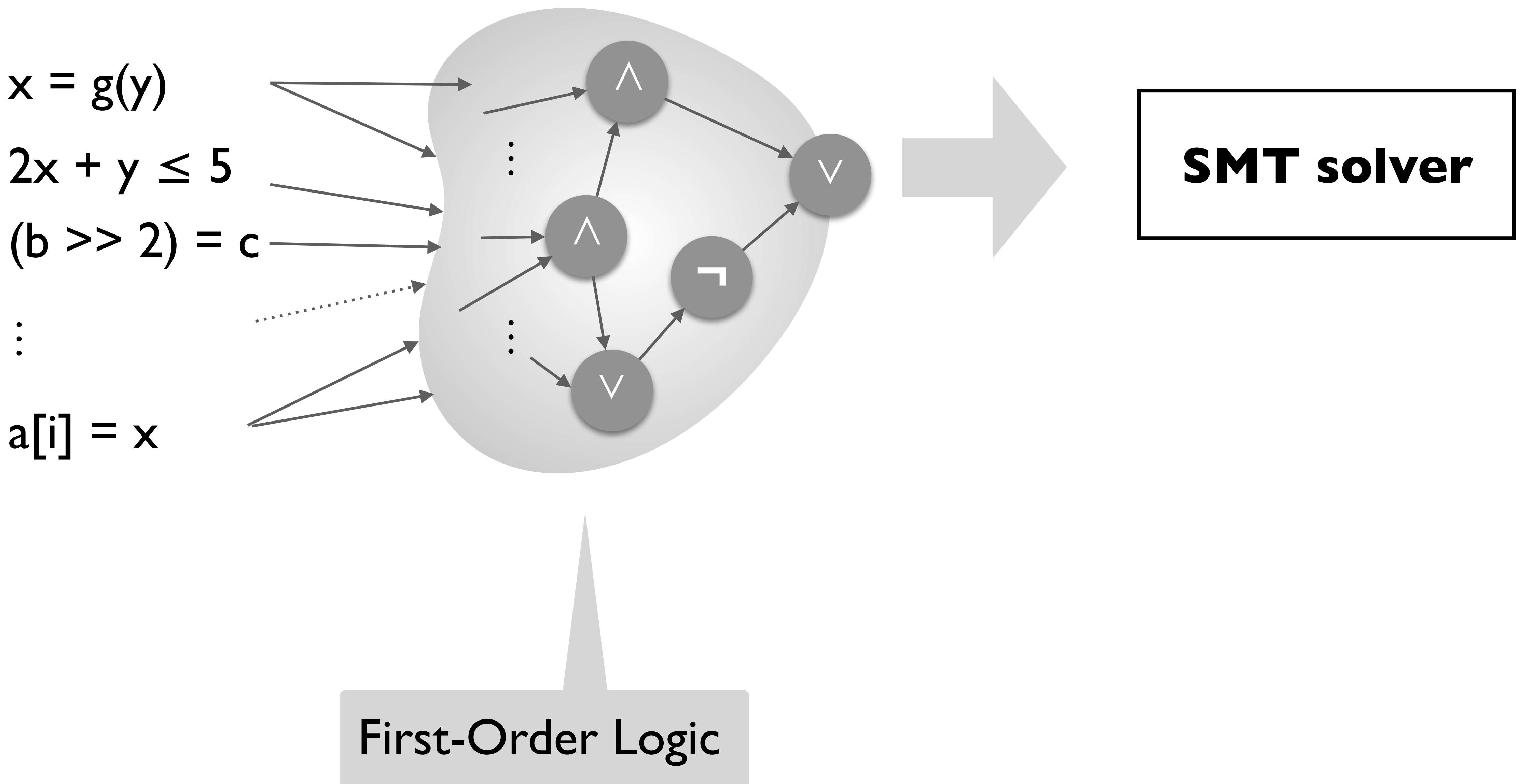
- HW1 due tonight at 11pm

topics

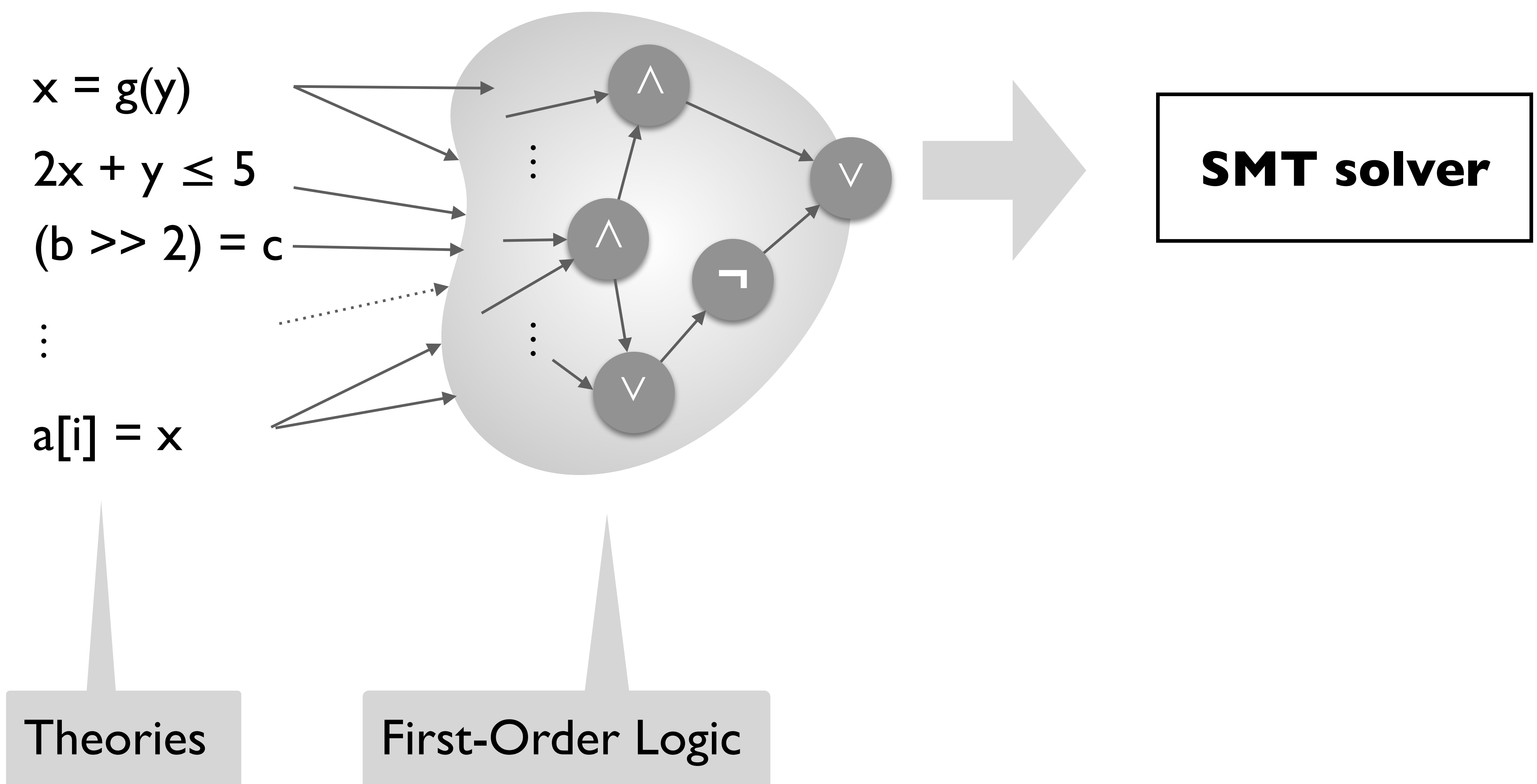
Satisfiability Modulo Theories (SMT)



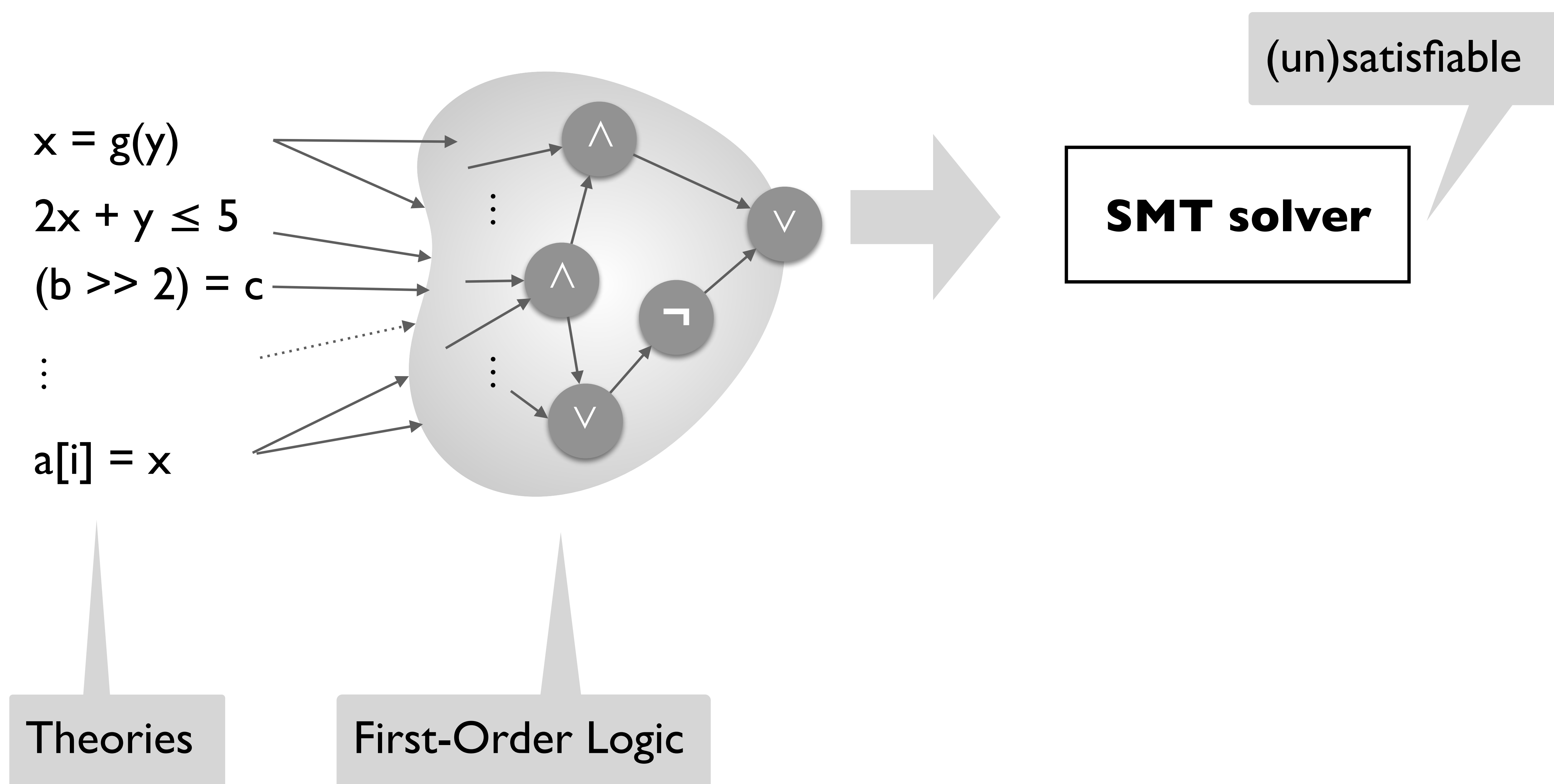
Satisfiability Modulo Theories (SMT)



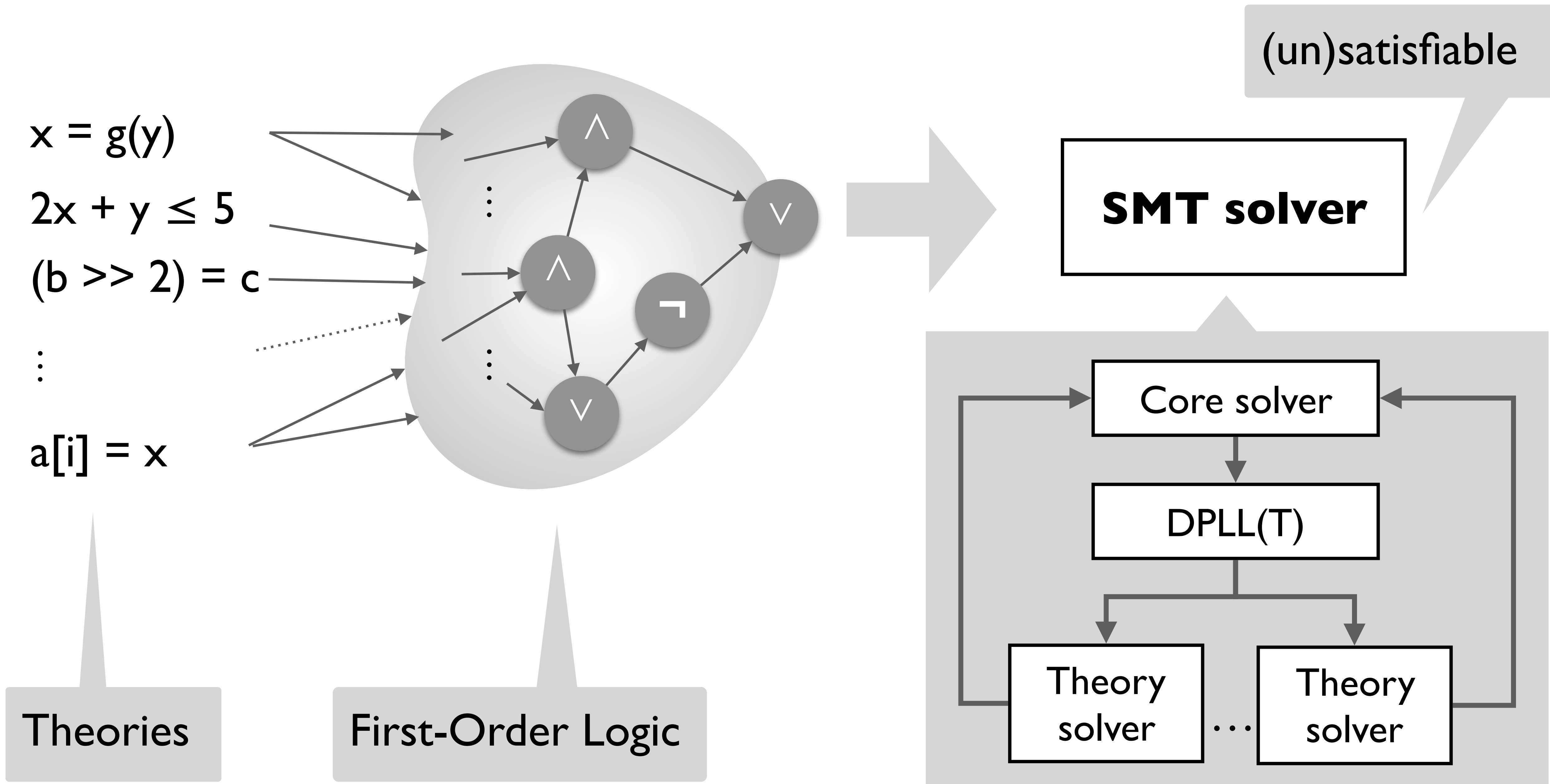
Satisfiability Modulo Theories (SMT)



Satisfiability Modulo Theories (SMT)



Satisfiability Modulo Theories (SMT)



Syntax of First-Order Logic (FOL)

Logical symbols

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

Non-logical symbols

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

Syntax of First-Order Logic (FOL)

Logical symbols

- Connectives: ¬, ∧, ∨, →, ↔
- Parentheses: ()
- ✗ Quantifiers: ∀, ∃

We will only consider the **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

Syntax of First-Order Logic (FOL)

Logical symbols

- Connectives: ¬, ∧, ∨, →, ↔
- Parentheses: ()
- ✗ Quantifiers: ∀, ∃

We will only consider the **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

In particular, we will consider quantifier-free **ground** formulas.

Syntax of quantifier-free ground FOL formulas

Logical symbols

- Connectives: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$
- 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 applied to n terms.
- An **atom** is $\top$, $\bot$, 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.

Syntax of quantifier-free ground FOL formulas

Logical symbols

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

Non-logical symbols

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

isPrime(x) $\rightarrow$ $\neg$ isInteger(sqrt(x))

- A **term** is a constant, or an n-ary function applied to n terms.
- An **atom** is $\top$, $\bot$, 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.

Semantics of FOL: first-order structures $\langle U, I \rangle$

Universe

Interpretation

Semantics of FOL: universe

Universe

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

Interpretation

Semantics of FOL: interpretation

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 $f_I : U^n \to U$
- Maps an n-ary predicate symbol p to an n-ary relation $p_I \subseteq U^n$

Semantics of FOL: inductive definition

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 $p_I \subseteq U^n$

$I[f(t_1, \ldots, t_n)] = I[f](I[t_1], \ldots, I[t_n])$

$I[p(t_1, \ldots, t_n)] = (\langle I[t_1], \ldots, I[t_n] \rangle \in I[p])$

$\langle U, I \rangle \vDash \top$

$\langle U, I \rangle \nvDash \bot$

$\langle U, I \rangle \vDash p(t_1, \ldots, t_n)$ **iff** $I[p(t_1, \ldots, t_n)] = \text{true}$

$\langle U, I \rangle \vDash \neg F$ **iff** $\langle U, I \rangle \nvDash F$

…

Semantics of FOL: inductive definition

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 $p_I \subseteq U^n$

$I[f(t_1, \ldots, t_n)] = I[f](I[t_1], \ldots, I[t_n])$

$I[p(t_1, \ldots, t_n)] = (\langle I[t_1], \ldots, I[t_n] \rangle \in I[p])$

$\langle U, I \rangle \models \top$

$\langle U, I \rangle \not\models \bot$

$\langle U, I \rangle \models p(t_1, \ldots, t_n)$ **iff** $I[p(t_1, \ldots, t_n)] = \text{true}$

$\langle U, I \rangle \models \neg F$ **iff** $\langle U, I \rangle \not\models F$

…

This is the semantics of **unsorted FOL**. SMT solvers work on **many-sorted FOL**, which partitions the universe into different types or sorts, and assigns types to non-logical symbols. SMT interpretations respect these types.

Semantics of FOL: example

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 $p_I \subseteq U^n$

U = {☀, ☁}

I[x] = ☀

I[y] = ☁

I[f] = {☀ ↦ ☁, ☁ ↦ ☀}

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 T-models

First-order theories

Signature Σ_T

- Set of constant, predicate, and function symbols

Set of T-models

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)

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 some T-model M.

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

First-order theories: expansion

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)

We can **expand** a theory's signature to include additional *uninterpreted* symbols (e.g., constants).

If E_T is an expansion of Σ_T, then the T-models of E_T are the set of all possible expansions of the T-models of Σ_T to include interpretations for the symbols in $E_T \setminus \Sigma_T$.

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

- $\forall x.\ x = x$
- $\forall x, y.\ x = y \rightarrow y = x$
- $\forall x, y, z.\ x = y \wedge y = z \rightarrow x = z$
- $\forall x_1, \ldots, x_n, y_1, \ldots, y_n.\ (x_1 = y_1 \wedge \ldots \wedge x_n = y_n) \rightarrow (f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n))$
- $\forall x_1, \ldots, x_n, y_1, \ldots, y_n.\ (x_1 = y_1 \wedge \ldots \wedge x_n = y_n) \rightarrow (p(x_1, \ldots, x_n) \leftrightarrow p(y_1, \ldots, y_n))$

Deciding $T_=$

- Conjunctions of literals modulo $T_=$ is decidable in polynomial time.

$T_=$ example: checking program equivalence

```
int abs(int y) {
  return y<0 ? -y : y;
}

int sq(int y) {
  return y*y;
}

int sqabs(int y) {
  return abs(y)*abs(y);
}
```

$T_=$ example: checking program equivalence

```
int abs(int y) {
  return y<0 ? -y : y;
}

int sq(int y) {
  return y*y;
}

int sqabs(int y) {
  return abs(y)*abs(y);
}
```

Are **sq** and **sqabs** equivalent on all 128-bit integers?

$T_=$ example: checking program equivalence

```
int abs(int y) {
  return y<0 ? -y : y;
}

int sq(int y) {
  return y*y;
}

int sqabs(int y) {
  return abs(y)*abs(y);
}
```

Are **sq** and **sqabs** equivalent on all 128-bit integers?

Yes, but the solver takes a while to return an answer because reasoning about multiplication is expensive.

$T_=$ example: checking program equivalence

```
int abs(int y) {
  return y<0 ? -y : y;
}

int sq(int y) {
  return y*y;
}

int sqabs(int y) {
  return abs(y)*abs(y);
}
```

Are **sq** and **sqabs** equivalent on all 128-bit integers?

Yes, but the solver takes a while to return an answer because reasoning about multiplication is expensive.

What happens if we replace the multiplication with an uninterpreted function?

Theory of fixed-width bitvectors

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.

Theories of linear integer and real arithmetic

Signature

- Integers (or reals)
- Arithmetic operations: multiplication by an integer (or real) number, +, -.
- Predicates: =, ≤.
- 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];
}
```

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

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

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

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.

Summary

Today

- Introduction to SMT
- Quantifier-free FOL (syntax & semantics)
- Overview of common theories

Next lecture

- Survey of theory solvers