Computer-Aided Reasoning for Software

CSE507

Satisfiability Modulo Theories

courses.cs.washington.edu/courses/cse507/14au/

Emina Torlak
emina@cs.washington.edu

Today

Today

Last lecture

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

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

Today

Last lecture

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

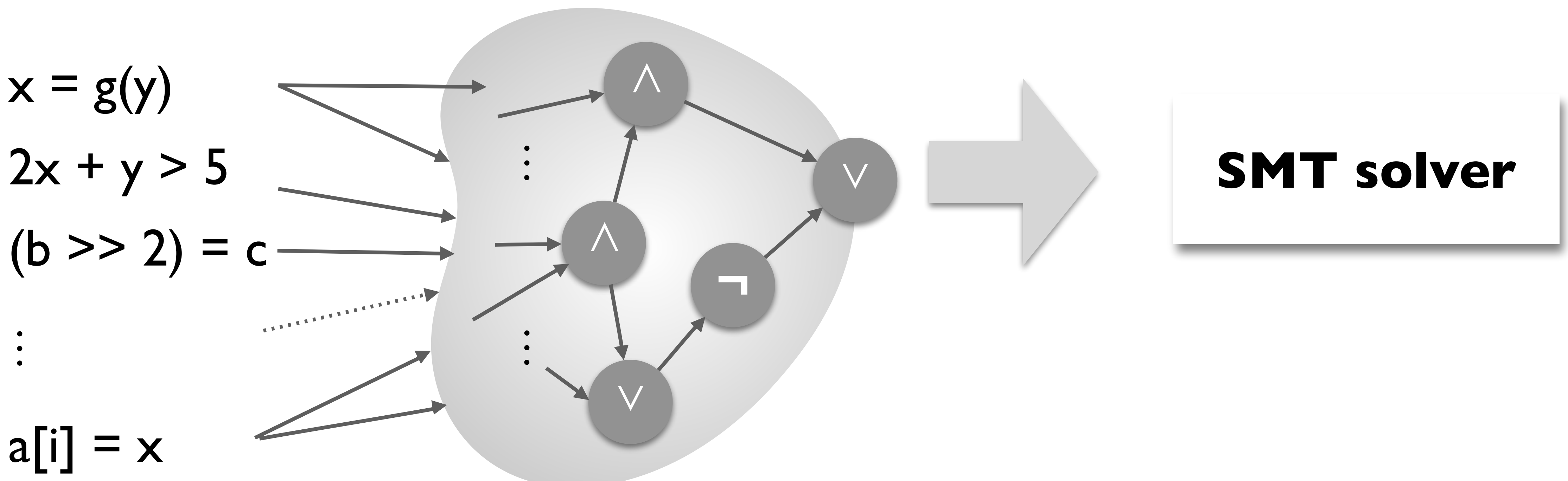
Today

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

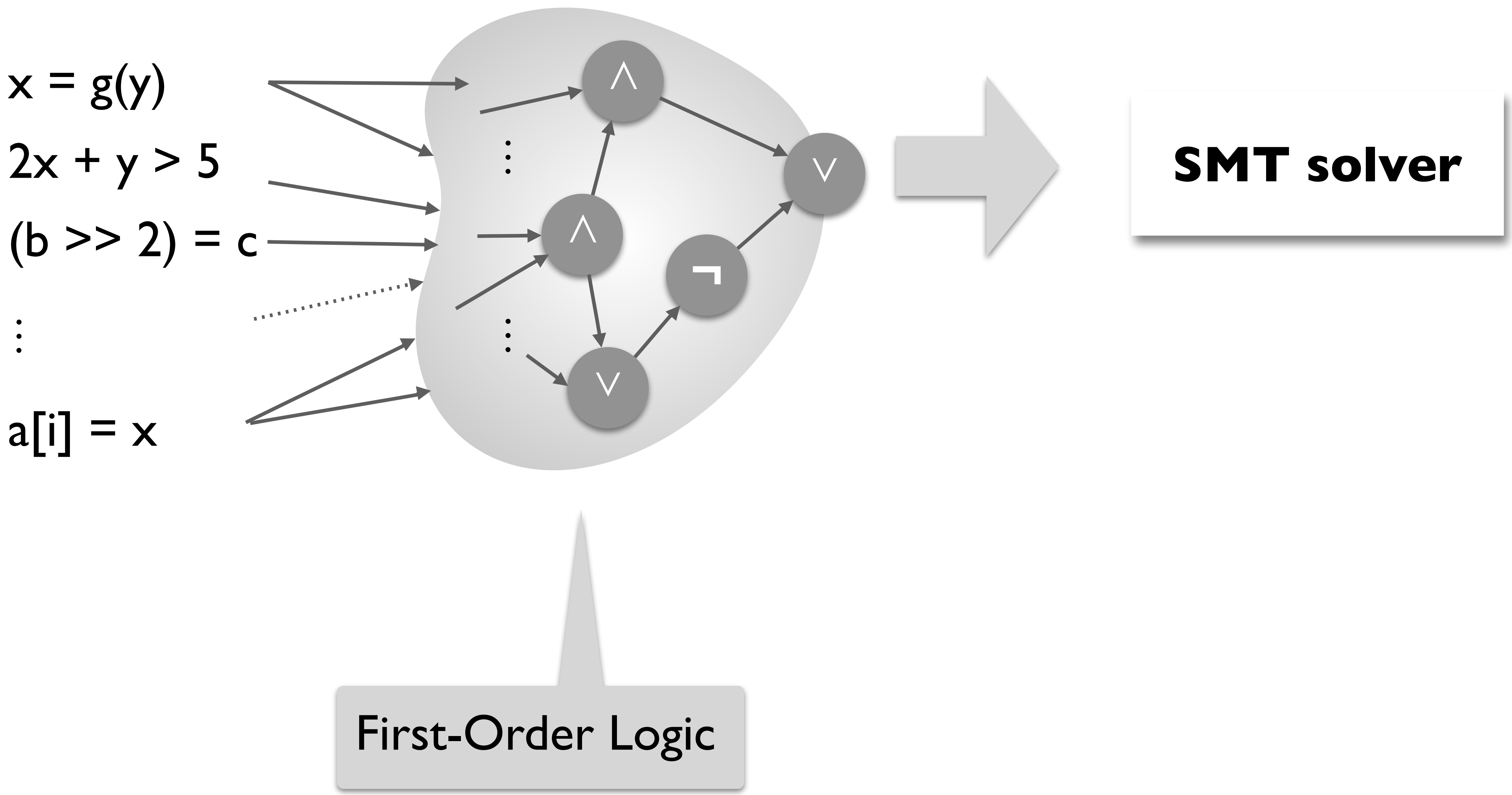
Reminder

- Email us the names of your team members by 11pm today

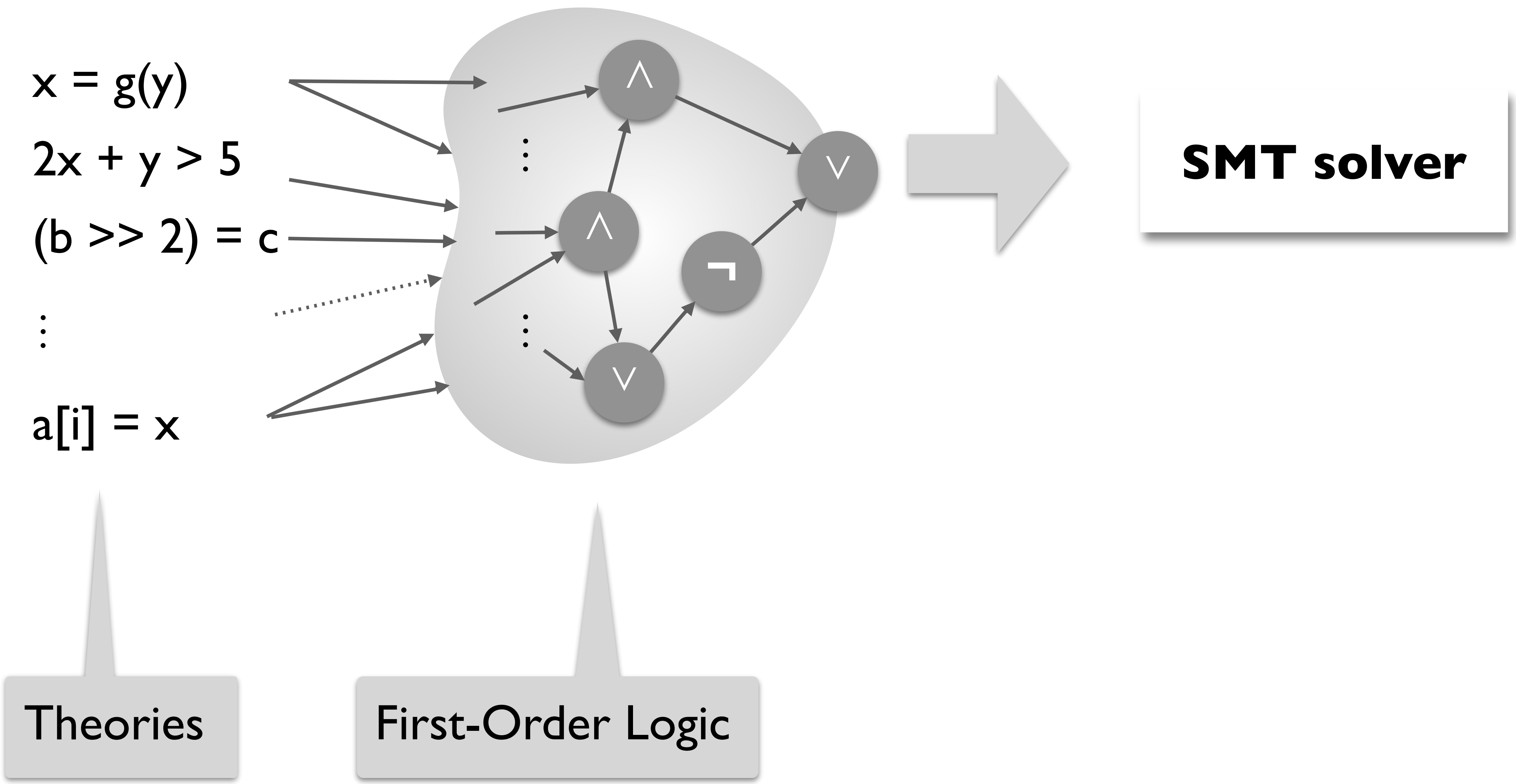
Satisfiability Modulo Theories (SMT)



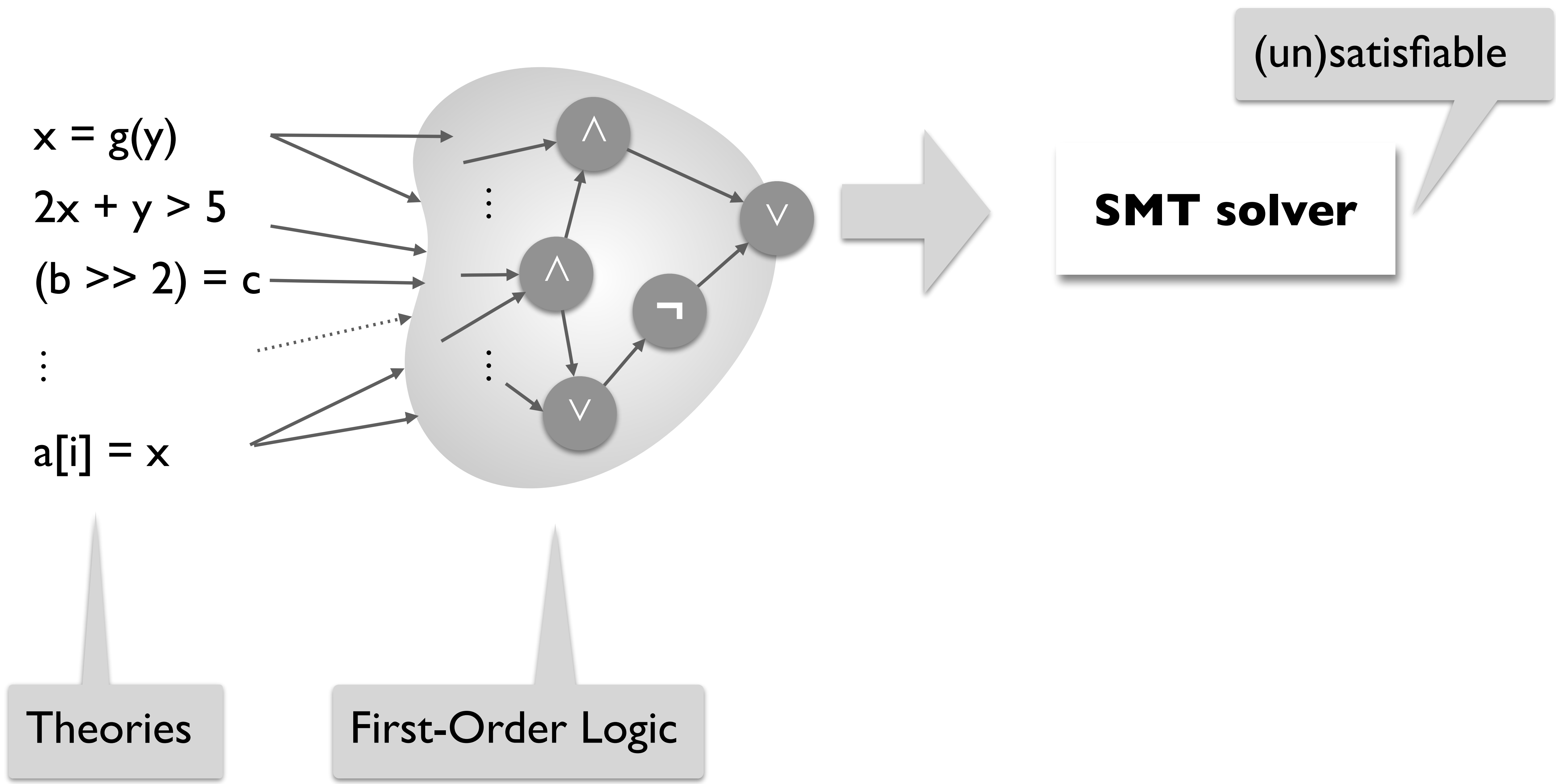
Satisfiability Modulo Theories (SMT)



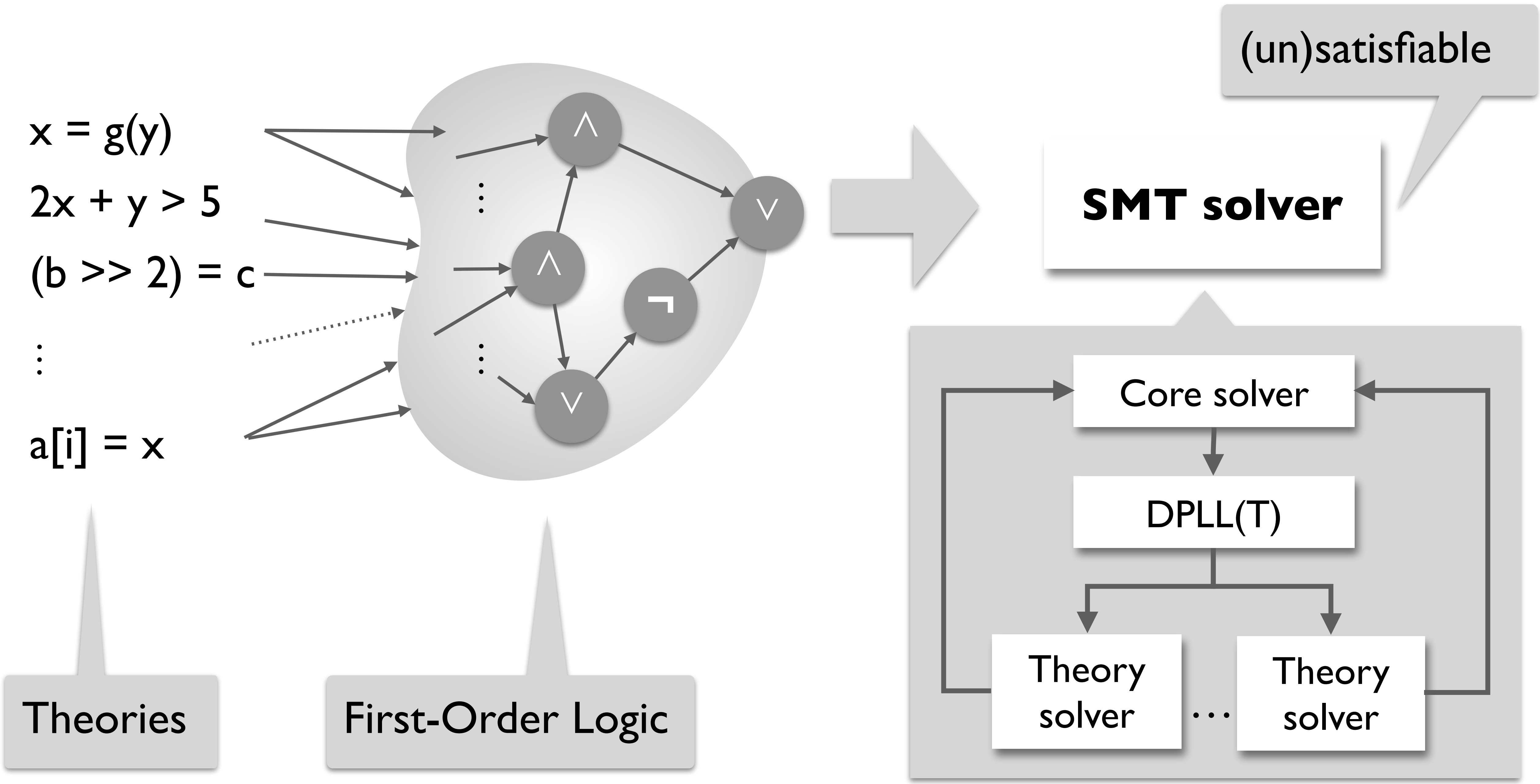
Satisfiability Modulo Theories (SMT)



Satisfiability Modulo Theories (SMT)



Satisfiability Modulo Theories (SMT)



First-Order Logic (FOL)

Logical symbols

- Connectives: $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$
- Parentheses: ()
- Quantifiers: $\forall, \exists$

Non-logical symbols

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

First-Order Logic (FOL)

Logical symbols

- Connectives: $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$
- Parentheses: ()
- ✗ ~~Quantifiers: $\forall, \exists$~~

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

First-Order Logic (FOL)

Logical symbols

- Connectives: $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$
- Parentheses: ()
- ✗ Quantifiers: $\forall, \exists$

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

No variables, just constants.

First-Order Logic (FOL)

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) **formula** is a literal or the application of logical connectives to formulas.

First-Order Logic (FOL)

Logical symbols

- Connectives: $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$
- Parentheses: ()

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

Non-logical symbols

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

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

Universe

Interpretation

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

Universe

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

Interpretation

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

Universe

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

Interpretation

- Maps a constant symbol c to an element of I: $I[c] \in U$
- 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 $p_I \subseteq U^n$

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

Universe

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

Interpretation

- Maps a constant symbol c to an element of I: $I[c] \in U$
- 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 $p_I \subseteq U^n$

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

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

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

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

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

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

…

Semantics of FOL: first-order structures ⟨U, I⟩

Universe

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

Interpretation

- Maps a constant symbol c to an element of I: $I[c] \in U$
- 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) = {⟨☀,☀⟩, ⟨☀,☁⟩}

⟨U, I⟩ ⊨ p(f(y), f(f(x))) ?

FOL satisfiability and validity

F is **satisfiable** iff M ⊨ F for some structure M = ⟨U, I⟩.

F is **valid** iff M ⊨ F for all structures M = ⟨U, I⟩.

Duality of satisfiability and validity:

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

Components of a first-order theory T

Signature Σ_T

Set of T-models

Components of a first-order theory T

Signature Σ_T

- Set of constant, predicate, and function symbols

Set of T-models

Components of a first-order theory T

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 Σ_T-models are the models of the theory axioms)

Components of a first-order theory T

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 Σ_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.

Common theories

Equality (and uninterpreted functions)

- $x = g(y)$

Fixed-width bitvectors

- $(b >> 1) = c$

Linear arithmetic (over R and Z)

- $2x + y > 5$

Arrays

- $a[i] = x$

Theory of equality (QF_UF)

Extends FOL with the equality predicate =

Theory of equality (QF_UF)

Extends FOL with the equality predicate =

Signature (all symbols)

- {=, x, y, z, …, f, g, …, p, q, …}

Theory of equality (QF_UF)

Extends FOL with the equality predicate =

Signature (all symbols)

- $\{=, x, y, z, \ldots, f, g, \ldots, p, q, \ldots\}$

Axioms

- = is reflexive, symmetric, transitive
- $\forall x_1, \ldots, x_n, y_1, \ldots, y_n.\ (x_1 = y_1 \land \ldots \land x_n = y_n) \rightarrow (f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n))$
- $\forall x_1, \ldots, x_n, y_1, \ldots, y_n.\ (x_1 = y_1 \land \ldots \land x_n = y_n) \rightarrow (p(x_1, \ldots, x_n) \leftrightarrow p(y_1, \ldots, y_n))$

Theory of equality (QF_UF)

Extends FOL with the equality predicate =

Signature (all symbols)

- $\{=, x, y, z, \ldots, f, g, \ldots, p, q, \ldots\}$

Axioms

- = is reflexive, symmetric, transitive
- $\forall x_1, \ldots, x_n, y_1, \ldots, y_n.\ (x_1 = y_1 \wedge \ldots \wedge x_n = y_n) \rightarrow (f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n))$
- $\forall x_1, \ldots, x_n, y_1, \ldots, y_n.\ (x_1 = y_1 \wedge \ldots \wedge x_n = y_n) \rightarrow (p(x_1, \ldots, x_n) \leftrightarrow p(y_1, \ldots, y_n))$

Decidable in polynomial time

QF_UF 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;
}
```

An QF_UF formula that is satisfiable iff programs are not equivalent:

Example from Sanjit Seshia

QF_UF 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;
}
```

An QF_UF formula that is satisfiable iff programs are not equivalent:

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

$(r_2 = y_0 * y_0) \wedge$

$\neg(r_2 = r_1)$

Example from Sanjit Seshia

QF_UF 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;
}
```

An QF_UF formula that is satisfiable iff programs are not 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.

Example from Sanjit Seshia

QF_UF 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;
}
```

An QF_UF formula that is satisfiable iff programs are not equivalent:

$(z_1 = y_0 \land y_1 = x_0 \land x_1 = z_1 \land r_1 = sq(x_1)) \land$

$(ret_2 = sq(y_0)) \land$

$\neg(ret_2 = ret_1)$

QF_UF 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;
}
```

An QF_UF formula that is satisfiable iff programs are not equivalent:

$(z_1 = y_0 \wedge y_1 = x_0 \wedge x_1 = z_1 \wedge r_1 = sq(x_1)) \wedge$

$(ret_2 = sq(y_0)) \wedge$

$\neg(ret_2 = ret_1)$

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

Example from Sanjit Seshia

QF_UF example: checking program equivalence

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

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

Is the uninterpreted function abstraction going to work in this case?

Example from Sanjit Seshia

QF_UF example: checking program equivalence

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

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

Is the uninterpreted function abstraction going to work in this case?

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

Example from Sanjit Seshia

Theory of fixed-width bitvectors (QF_BV)

Signature

- constants
- fixed-width words (modeling machine ints, longs, etc.)
- arithmetic operations (+, -, *, /, etc.)
- bitwise operations (&, |, ^, etc.)
- comparison operators (<, >, etc.)
- equality (=)

Satisfiability problem: NP-complete

Theory of linear arithmetic (QF_LIA)

Theory of linear arithmetic (QF_LIA)

Signature

- $\{\ldots, -1, 0, 1, \ldots, -2\cdot, 2\cdot, \ldots, +, -, =, >, x, y, z, \ldots\}$
- Constants, integers, multiplication by an integer constant, addition, subtraction, equality, greater-than

Theory of linear arithmetic (QF_LIA)

Signature

- {…, -1, 0, 1, …, -2 · , 2 · , …, +, -, =, >, x, y, z, …}
- Constants, integers, multiplication by an integer constant, addition, subtraction, equality, greater-than

Satisfiability problem: NP-complete

Theory of linear arithmetic (QF_LIA)

Signature

- {…, -1, 0, 1, …, -2 · , 2 · , …, +, -, =, >, x, y, z, …}
- Constants, integers, multiplication by an integer constant, addition, subtraction, equality, greater-than

Satisfiability problem: NP-complete

Theory of reals (QF_LRA) can be decided in polynomial time.

Theory of linear arithmetic (QF_LIA)

Signature

- {…, -1, 0, 1, …, -2 · , 2 · , …, +, -, =, >, x, y, z, …}
- Constants, integers, multiplication by an integer constant, addition, subtraction, equality, greater-than

Satisfiability problem: NP-complete

Theory of reals (QF_LRA) can be decided in polynomial time.

Difference Logic (QF_DIA) can also be decided in polynomial time

- Conjunctions of the form $x - y \leq c$, where c is an integer constant

QF_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;
}
```

An QF_LIA formula that is satisfiable iff this transformation is invalid:

QF_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;
}
```

An QF_LIA formula that is satisfiable iff this transformation is invalid:

$(i \geq 1) \wedge (i \leq 10) \wedge$

$(j + i = j)$

Polyhedral model

Theory of arrays

Theory of arrays

Signature

- {read, write, =, x, y, z, . . .}

Theory of arrays

Signature

- {read, write, =, x, y, z, …}

Axioms

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

Theory of arrays

Signature

- {read, write, =, x, y, z, …}

Axioms

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

Satisfiability problem: NP-complete

Theory of arrays

Signature

- $\{read, write, =, x, y, z, \ldots\}$

Axioms

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

Satisfiability problem: NP-complete

Used in many software verification tools to model memory (e.g., Dafny)

Summary

Today

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

Next lecture

- Survey of theory solvers