

# Synthesis

# Reading Reflection

Discuss in groups

- How would the different synthesis approaches described in the reading affect the user interaction model?
- How would the approaches described in the reading apply or not apply to the various synthesis project ideas you brainstormed during our first synthesis week?

# Reading Key Takeaways

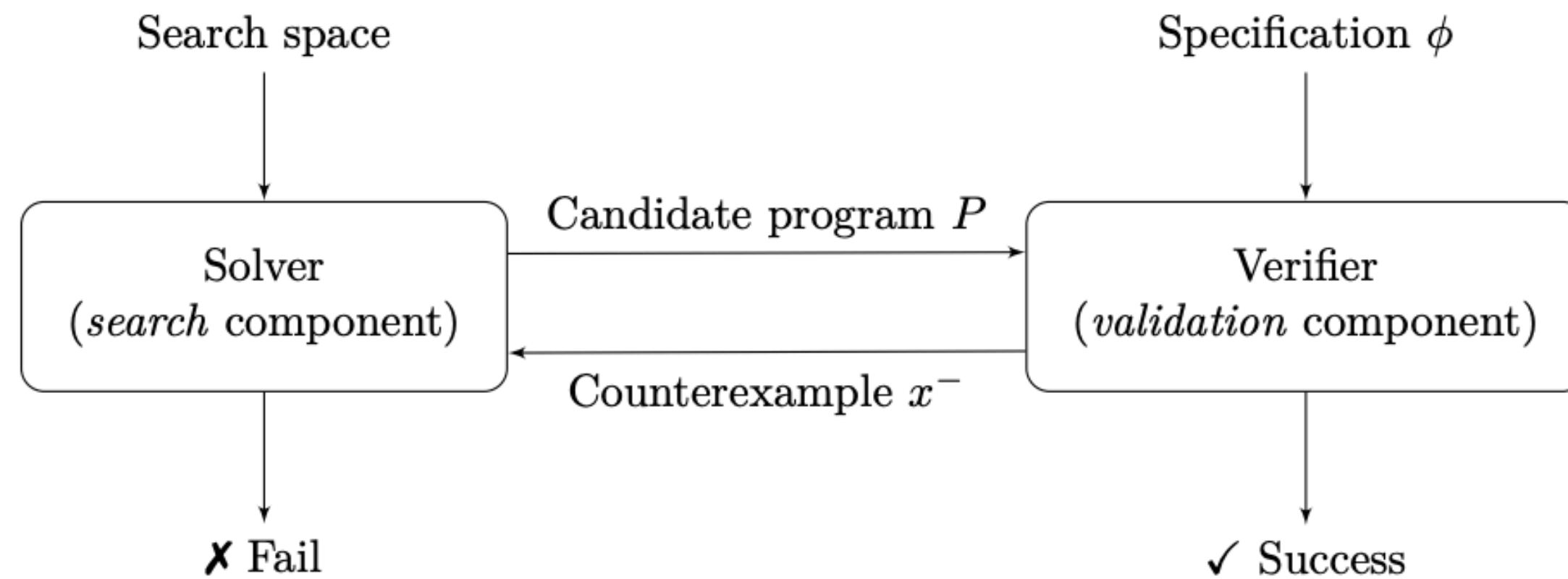


Figure 3.2: Counterexample-guided inductive synthesis.

- CEGIS!

- Distinguishing inputs—2 programs match our spec. How will we find the one we want? Ask the user what we should do on this next input, for which the programs produce different outputs.
- Syntactic bias—as we’ve already discussed, language shapes the search space
- SyGuS—SyGuS solvers can be a really useful starting point for a new synthesis project! See Fig 3.10 for how nice the programs are.

# SyGuS string example

```
(set-logic SLIA)

(synth-fun f ((name String)) String
  ((Start String (ntString))
   (ntString String (name " " "." "Dr." (str.++ ntString ntString)
    (str.replace ntString ntString ntString) (str.at ntString ntInt) (int.to.str ntInt)
    (str.substr ntString ntInt ntInt)))
   (ntInt Int (0 1 2 (+ ntInt ntInt) (- ntInt ntInt) (str.len ntString)
    (str.to.int ntString) (str.indexof ntString ntString ntInt)))
   (ntBool Bool (true false (str.prefixof ntString ntString)
    (str.suffixof ntString ntString) (str.contains ntString ntString))))))

(declare-var name String)
(constraint (= (f "Nancy FreeHafer") "Dr. Nancy"))
(constraint (= (f "Andrew Cencici") "Dr. Andrew"))
(constraint (= (f "Jan Kotas") "Dr. Jan"))
(constraint (= (f "Mariya Sergienko") "Dr. Mariya"))

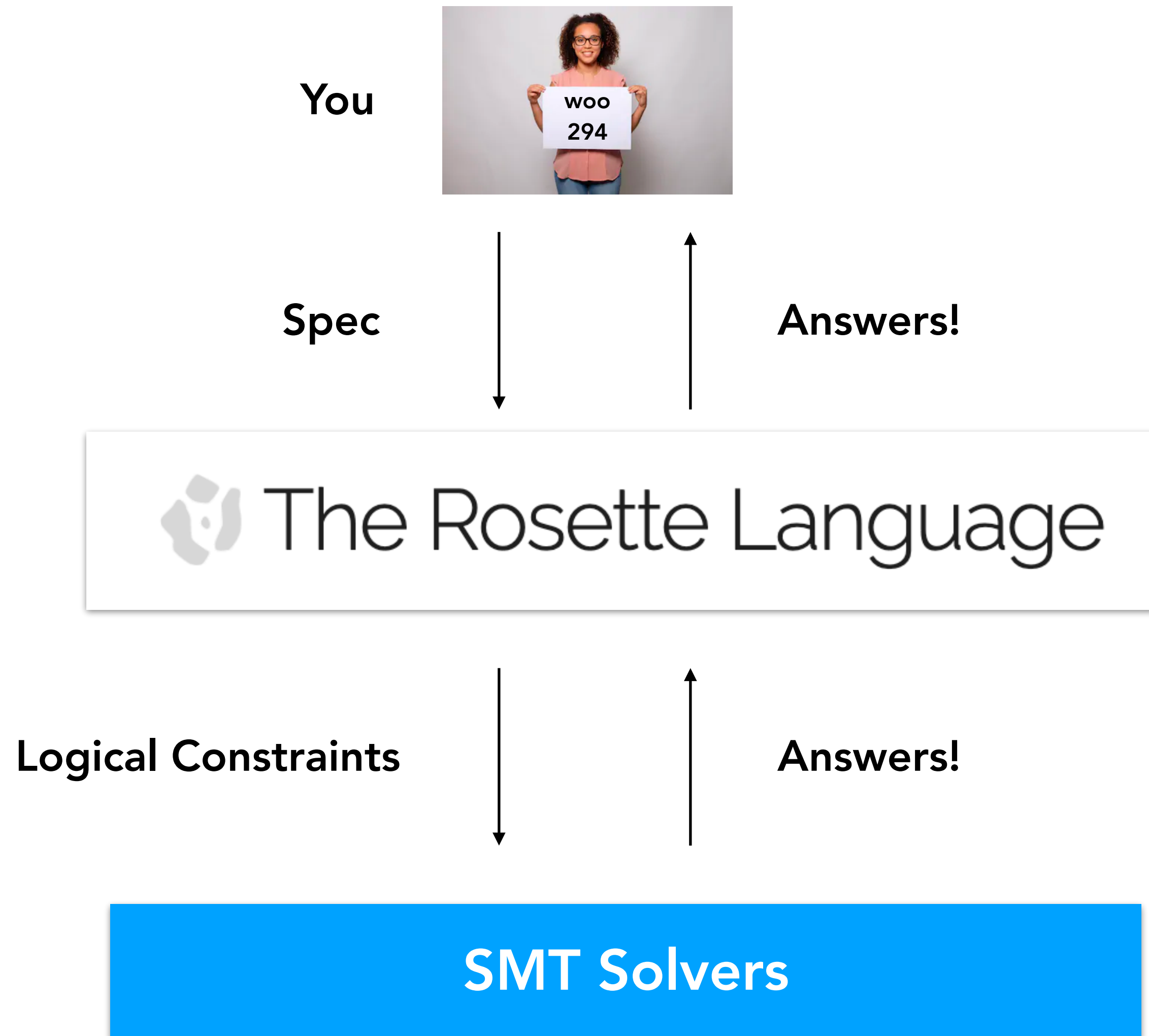
(check-synth)
```

# Reflections on Rosette

- Concise program -> quite complex and sophisticated synthesizers
- Opacity
- Control



# Today's topic: SMT



OK, what's SMT?

Satisfiability Modulo Theories



ok, and what's  
satisfiability??



# Let's back up

- **SAT: Boolean satisfiability problem**; also sometimes called SATISFIABILITY
  - Given a Boolean formula, is there an interpretation of the formula that satisfies it?  
**Can we replace the variables of the Boolean formula with the values TRUE or FALSE such that the formula evaluates to TRUE?**
    - If yes, the formula is satisfiable
    - If no assignment out of all possible assignments makes the formula TRUE, it's unsatisfiable
- Examples:
  - $p \wedge q$  is satisfiable; ( $p=\text{TRUE}$ ,  $q=\text{TRUE}$ )
  - $p \wedge \neg p$  is unsatisfiable
- SAT is NP-complete
  - ...but that hasn't stopped folks from building some seriously efficient SAT solvers and using them to solve real problems

Next few slides shamelessly  
lifted from...

Computer-Aided Reasoning for Software

**SAT Solving Basics**

CSE507

**Emina Torlak**

emina@cs.washington.edu

See <https://courses.cs.washington.edu/courses/cse507/19au/calendar.html> for more

## Syntax of propositional logic

$$(\neg p \wedge \top) \vee (q \rightarrow \perp)$$

## Syntax of propositional logic

$$(\neg p \wedge \top) \vee (q \rightarrow \perp)$$

**Atom**

**truth symbols:**  $\top$  (“true”),  $\perp$  (“false”)

**propositional variables:**  $p, q, r, \dots$

## Syntax of propositional logic

$$(\neg p \wedge \top) \vee (q \rightarrow \perp)$$

**Atom**            **truth symbols:**  $\top$  (“true”),  $\perp$  (“false”)  
**propositional variables:**  $p, q, r, \dots$

**Literal**            an atom  $\alpha$  or its negation  $\neg\alpha$

## Syntax of propositional logic

$$(\neg p \wedge \top) \vee (q \rightarrow \perp)$$

### Atom

**truth symbols:**  $\top$  (“true”),  $\perp$  (“false”)

**propositional variables:**  $p, q, r, \dots$

### Literal

an atom  $\alpha$  or its negation  $\neg\alpha$

### Formula

an atom or the application of a **logical connective** to formulas  $F_1, F_2$ :

$\neg F_1$	“not”	(negation)
$F_1 \wedge F_2$	“and”	(conjunction)
$F_1 \vee F_2$	“or”	(disjunction)
$F_1 \rightarrow F_2$	“implies”	(implication)
$F_1 \leftrightarrow F_2$	“if and only if”	(iff)

## Semantics of propositional logic: interpretations

An **interpretation**  $I$  for a propositional formula  $F$  maps every variable in  $F$  to a truth value:

$$I : \{ p \mapsto \text{true}, q \mapsto \text{false}, \dots \}$$

## Semantics of propositional logic: interpretations

An **interpretation**  $I$  for a propositional formula  $F$  maps every variable in  $F$  to a truth value:

$$I : \{ p \mapsto \text{true}, q \mapsto \text{false}, \dots \}$$

$I$  is a **satisfying interpretation** of  $F$ , written as  $I \models F$ , if  $F$  evaluates to true under  $I$ .

$I$  is a **falsifying interpretation** of  $F$ , written as  $I \not\models F$ , if  $F$  evaluates to false under  $I$ .



## Semantics of propositional logic: interpretations

An **interpretation**  $I$  for a propositional formula  $F$  maps every variable in  $F$  to a truth value:

$$I : \{ p \mapsto \text{true}, q \mapsto \text{false}, \dots \}$$

$I$  is a **satisfying interpretation** of  $F$ , written as  $I \models F$ , if  $F$  evaluates to true under  $I$ .

$I$  is a **falsifying interpretation** of  $F$ , written as  $I \not\models F$ , if  $F$  evaluates to false under  $I$ .

A satisfying interpretation is also called a **model**.

## Satisfiability & validity of propositional formulas

$F$  is **satisfiable** iff  $I \models F$  for some  $I$ .

$F$  is **valid** iff  $I \models F$  for all  $I$ .

## Satisfiability & validity of propositional formulas

$F$  is **satisfiable** iff  $I \models F$  for some  $I$ .

$F$  is **valid** iff  $I \models F$  for all  $I$ .

**Duality** of satisfiability and validity:

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

## Satisfiability & validity of propositional formulas

$F$  is **satisfiable** iff  $I \models F$  for some  $I$ .

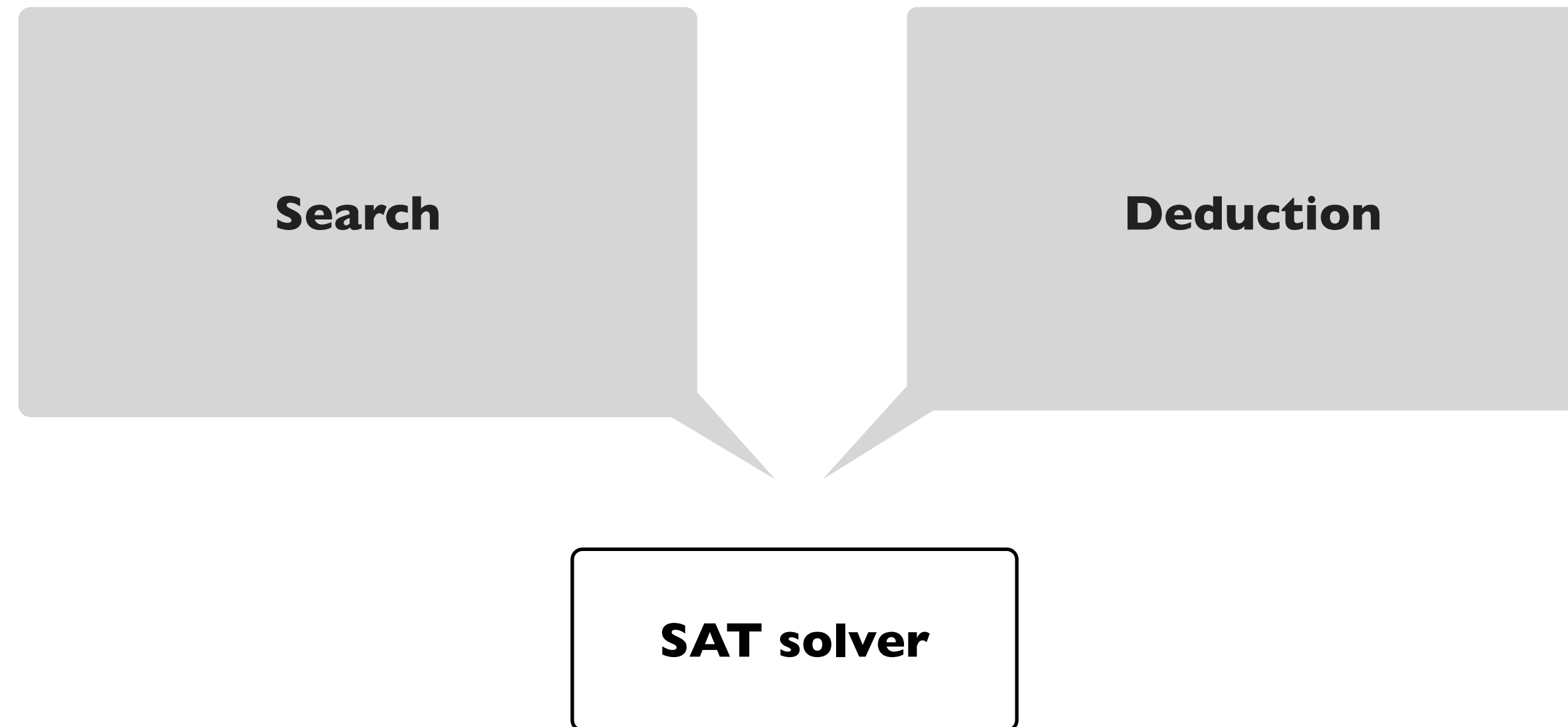
$F$  is **valid** iff  $I \models F$  for all  $I$ .

**Duality** of satisfiability and validity:

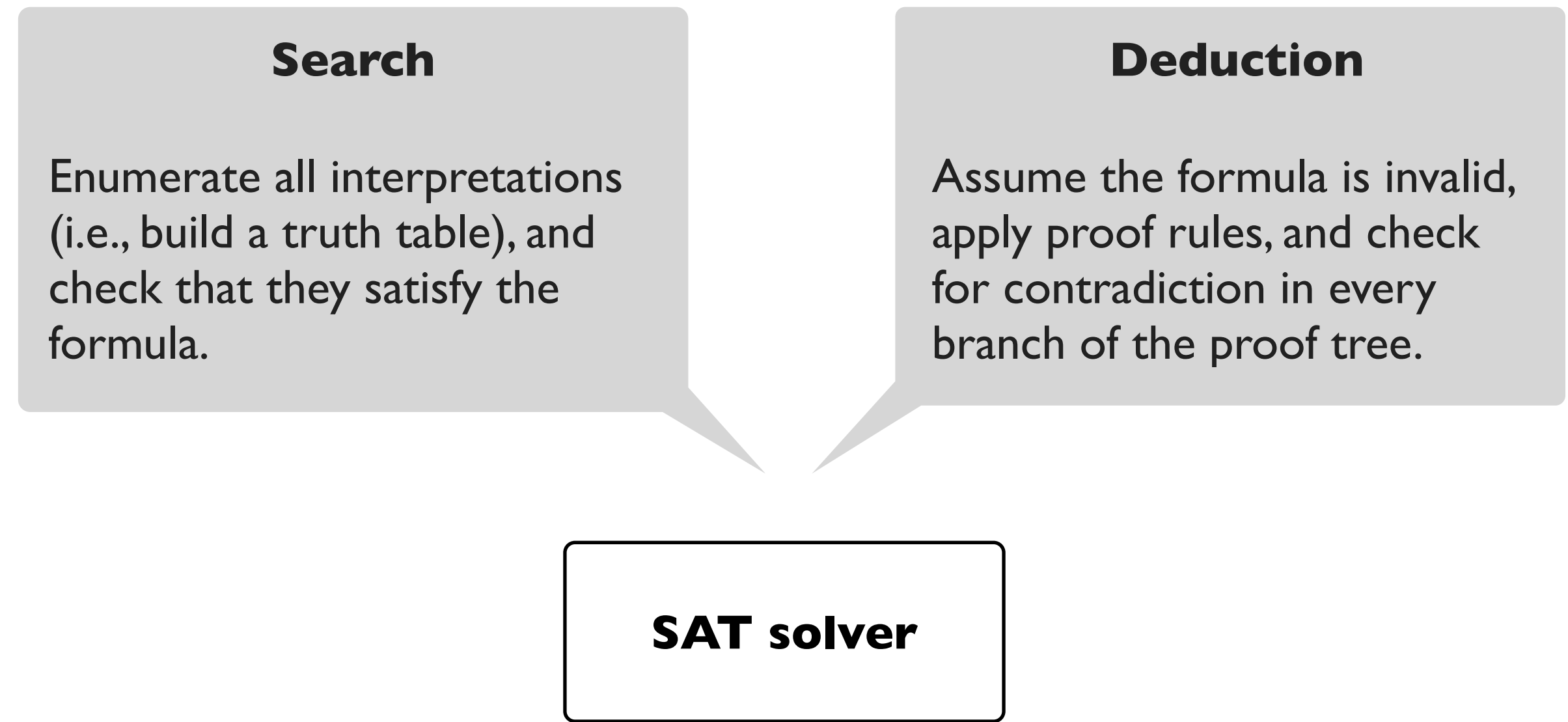
$F$  is valid iff  $\neg F$  is unsatisfiable.

If we have a procedure for checking satisfiability, we can also check validity of propositional formulas, and vice versa.

## **Techniques for deciding satisfiability & validity**



## Techniques for deciding satisfiability & validity



## Proof by search: enumerating interpretations

$$F: (p \wedge q) \rightarrow (p \vee \neg q)$$

$p$	$q$	$p \wedge q$	$\neg q$	$p \vee \neg q$	$F$
0	0	0	1	1	1
0	1	0	0	0	1
1	0	0	1	1	1
1	1	1	0	1	1

Valid.

5 minute break



# Now that we know about SAT...

- Ok so...what's SMT?
- Satisfiability (SAT) **Modulo Theories**

Next few slides shamelessly  
lifted from...

Computer-Aided Reasoning for Software

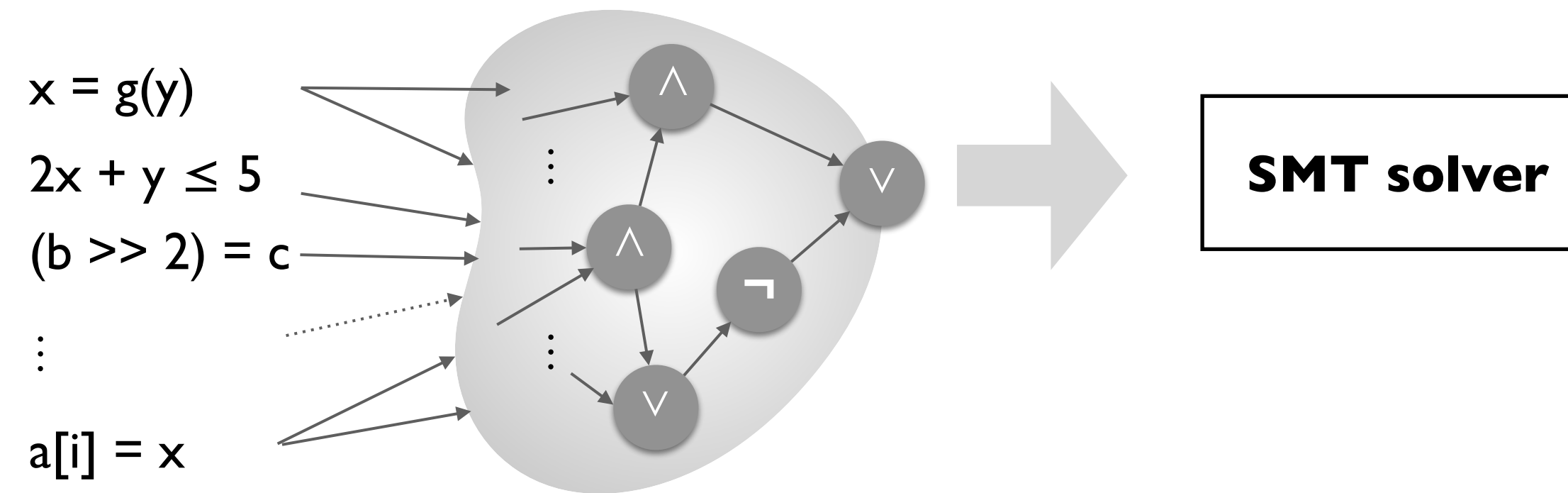
# **Satisfiability Modulo Theories**

**Emina Torlak**

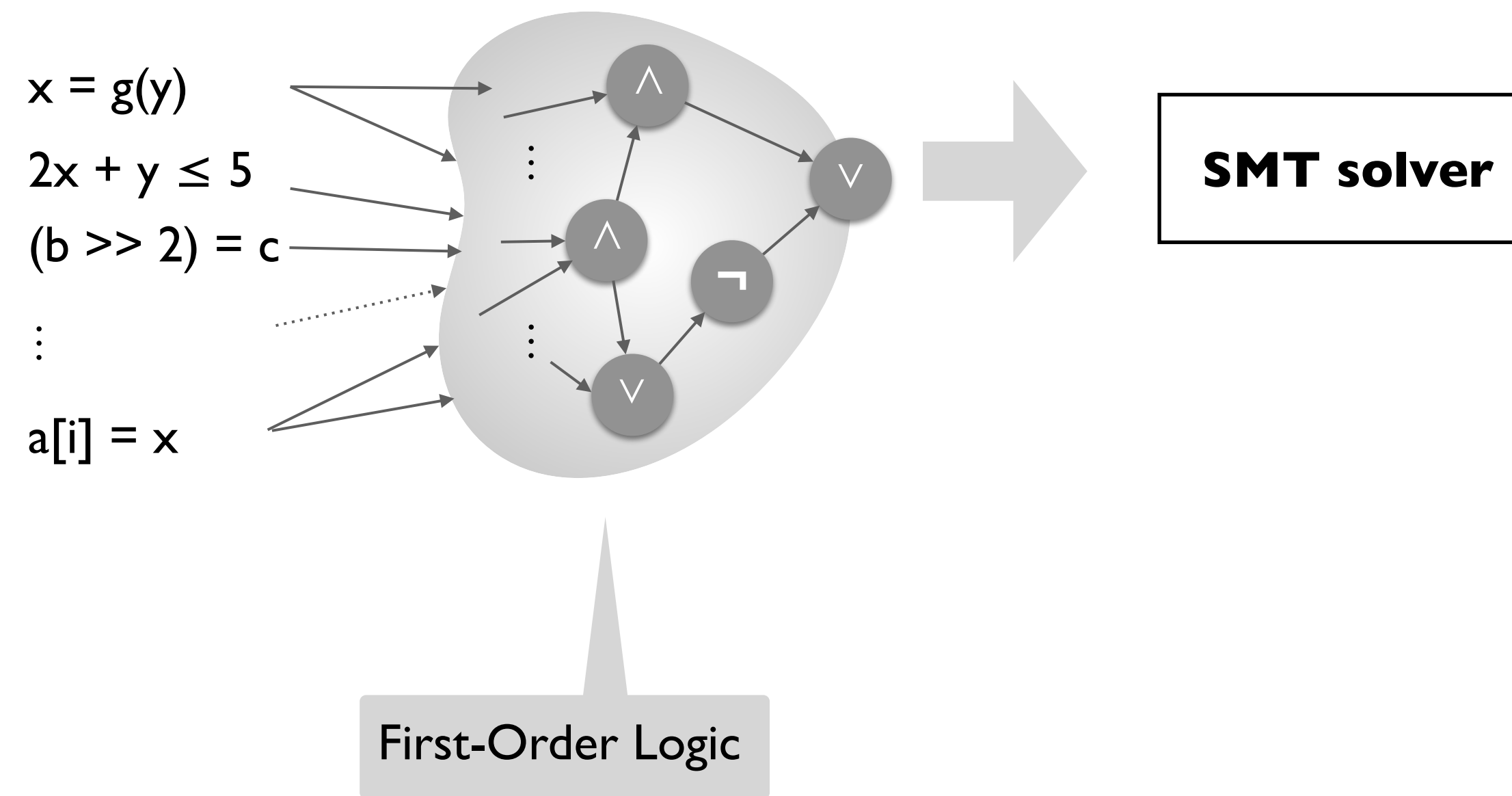
emina@cs.washington.edu

Again, see <https://courses.cs.washington.edu/courses/cse507/19au/calendar.html> for more

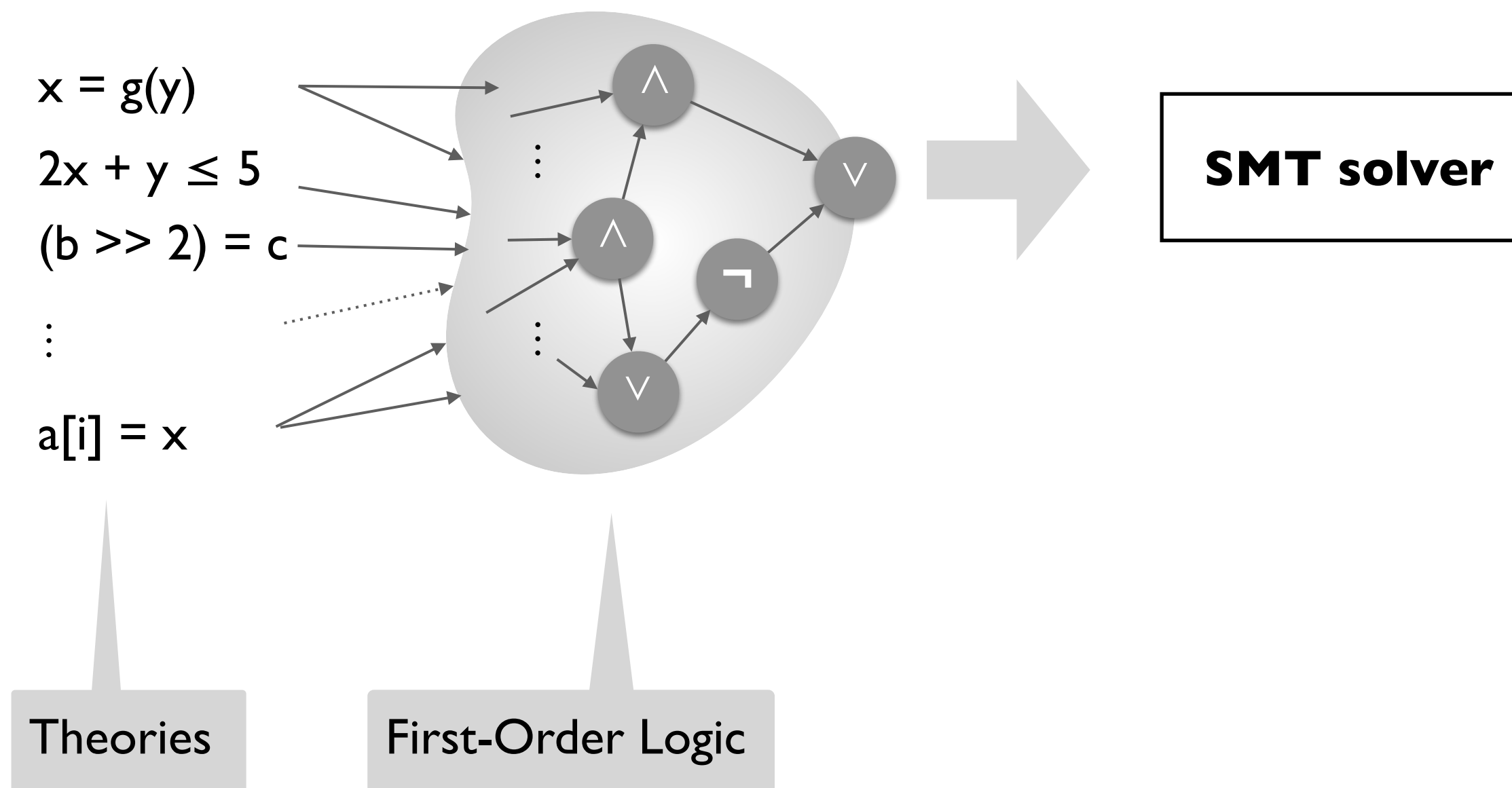
## Satisfiability Modulo Theories (SMT)



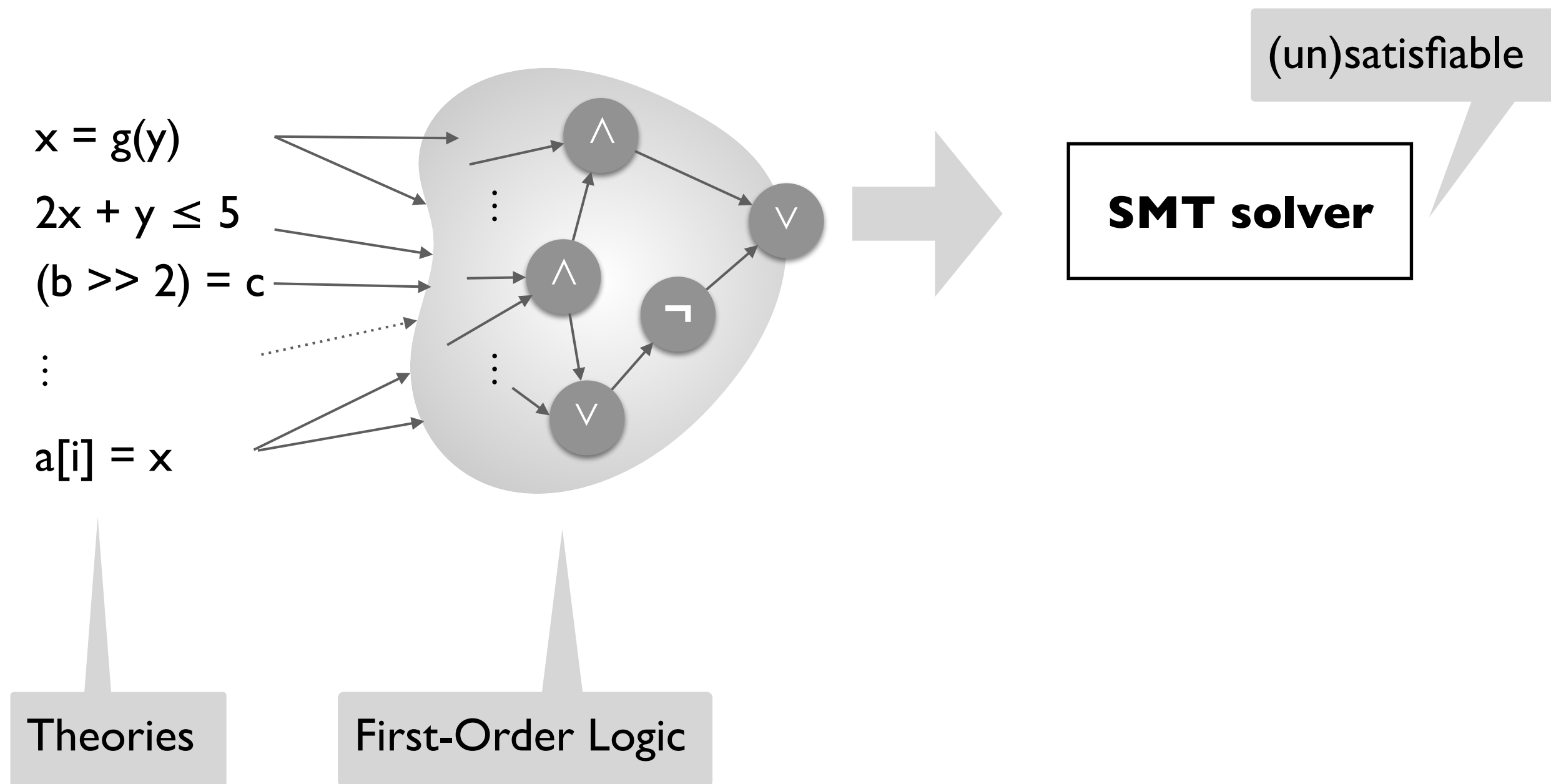
## Satisfiability Modulo Theories (SMT)



# Satisfiability Modulo Theories (SMT)



# Satisfiability Modulo Theories (SMT)



# Syntax of First-Order Logic (FOL)

## Logical symbols

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

## Non-logical symbols

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

We will only consider the **quantifier-free** fragment of FOL.

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

## 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  $p_I \subseteq U^n$

$$U = \{\odot, \clubsuit\}$$

$$I[x] = \odot$$

$$I[y] = \clubsuit$$

$$I[f] = \{\odot \mapsto \clubsuit, \clubsuit \mapsto \odot\}$$

$$I[p] = \{\langle \odot, \odot \rangle, \langle \odot, \clubsuit \rangle\}$$

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

You decide!  
Take 1 min.



## 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.

## **Common theories**

### **Equality (and uninterpreted functions)**

- $x = g(y)$

### **Fixed-width bitvectors**

- $(b \gg l) = c$

### **Linear arithmetic (over $\mathbf{R}$ and $\mathbf{Z}$ )**

- $2x + y \leq 5$

### **Arrays**

- $a[i] = x$

# Theory of equality with uninterpreted functions

**Signature:**  $\{=, x, y, z, \dots, f, g, \dots, p, q, \dots\}$

- The binary predicate  $=$  is *interpreted*.
- All constant, function, and predicate symbols are *uninterpreted*.

## Axioms

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

## 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

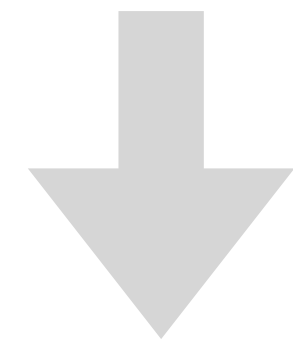
- Integers (or reals)
- Arithmetic operations: multiplication by an integer (or real) number, +, -.
- Predicates: =,  $\leq$ .
- Expanded with all constant symbols:  $x, y, z, \dots$

## 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];  
}
```

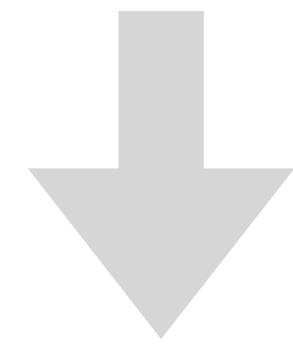


```
int v = a[j];  
for (i=1; i<=10; i++) {  
    a[j+i] = v;  
}
```

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

## 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:

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

# Theory of arrays

## Signature

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

## Axioms

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

## Deciding $T_A$

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

# Basically...

- SAT lets us say simple things
- SMT lets us say...other simple things. But more complicated than SAT!
- And it's enough that we can get to some interesting tasks

SMT activity