

Lecture 11

From verification to synthesis (I)

Nadia Polikarpova

Map of the unit

Constraint-based synthesis

- How to solve constraints about infinitely many inputs? CEGIS
- How to encode semantics of looping / recursive programs?
 - Bounded reasoning
- • Unbounded / deductive reasoning

Enumerative (and deductive) synthesis

- How to use deductive reasoning to guide the search?

Verification

```
method SumMax (a: array<int>) returns (sum: int, max: int)
  ensures sum <= a.Length * max;
{
  sum, max := 0, 0;
  var i := 0;
  while (i < a.Length)
    invariant 0 <= i <= a.Length && sum <= i * max;
    decreases a.Length - i;
  {
    if (max < a[i]) { max := a[i]; }
    sum := sum + a[i];
    i := i + 1;
  }
}
```

Dafny



correct!



AutoProof



can't proof
correctness

VCC

Verifast

...

Invariant inference

```
method SumMax (a: array<int>) returns (sum: int, max: int)
  ensures sum <= a.Length * max;
{
  sum, max := 0, 0;
  var i := 0;
  while (i < a.Length)
    invariant ??;
    decreases ??;
  {
    if (max < a[i]) { max := a[i]; }
    sum := sum + a[i];
    i := i + 1;
  }
}
```



BLAST

Astrée

FB Infer

LiquidHaskell

...

correct!



invariant $0 \leq i \leq a.Length$
&& $sum \leq i * max$;
decreases $a.Length - i$;



can't find an invariant that
lets me prove correctness

Program synthesis

```
method SumMax (a: array<int>) returns (sum: int, max: int)
  ensures sum <= a.Length * max;
{
  var i;
  ??;
  while (??)
    invariant ??;
    decreases ??;
  {
    ??;
  }
}
```



VS3

Natural Synthesis

(Leon)

(Synquid)



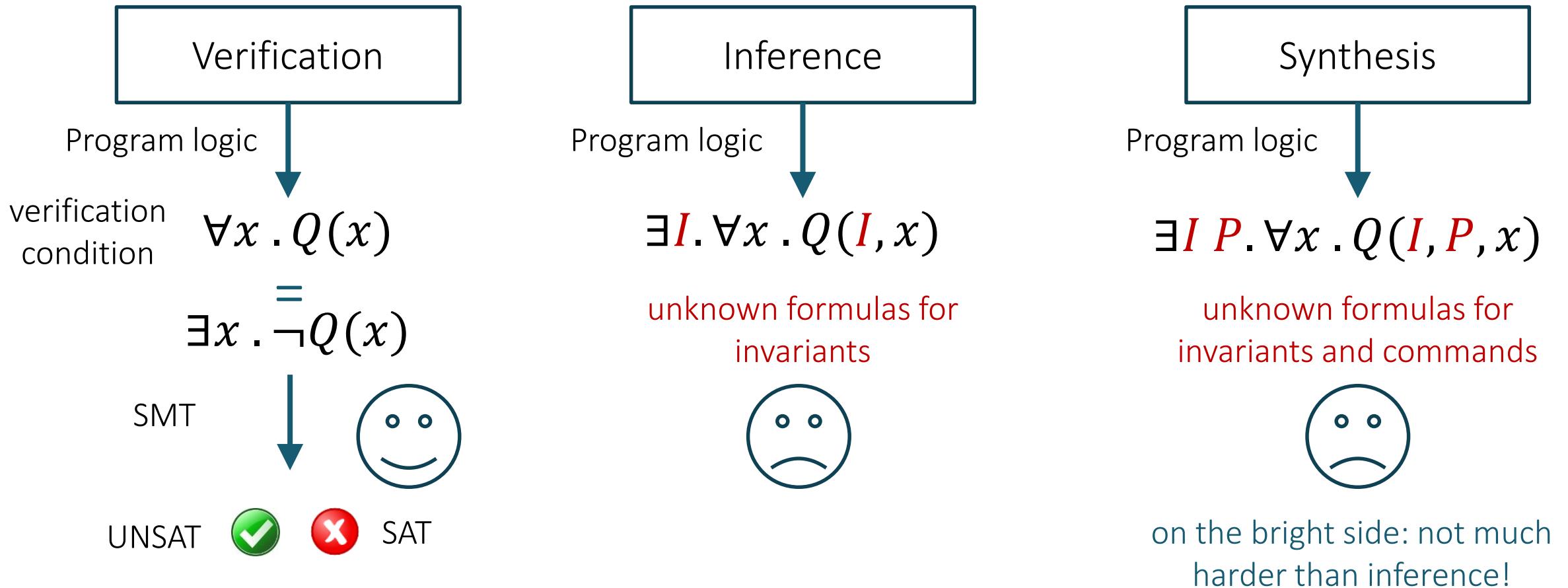
found a correct program!

```
sum, max := 0, 0;
var i := 0;
while (i < a.Length)
  invariant 0 <= i <= a.Length && sum <= i * max;
  decreases a.Length - i;
{
  if (max < a[i]) { max := a[i]; }
  sum := sum + a[i];
  i := i + 1;
}
```



can't find a (program,
invariant) pair that I can
prove correct

Verification \rightarrow inference \rightarrow synthesis



How verification works

Verification



$\forall x . Q(x)$

Step 1: eliminate loops

$\{T\}$

```
sum := 0;
max := 0;
i := 0;
while (i < a.Length)
  invariant i <= a.Length;
  invariant sum <= i * max;
{
  assert i < a.Length;
  if (max < a[i]) { max := a[i]; }
  sum := sum + a[i];
  i := i + 1;
}
```

$\{s \leq a.L * m\}$

$\{T\}$

```
sum := 0;
max := 0;
i := 0;
```

$\{i \leq a.L \wedge s \leq i * m\}$

$\{i \leq a.L \wedge s \leq i * m \wedge i < a.L\}$

```
assert i < a.Length;
if (max < a[i]) { max := a[i]; }
sum := sum + a[i];
i := i + 1;
```

$\{i \leq a.L \wedge s \leq i * m\}$

$\{i \leq a.L \wedge s \leq i * m \wedge \neg(i < a.L)\}$

skip

$\{s \leq a.L * m\}$

Step 2: generate VC

$\{T\}$

\Rightarrow

$\{0 \leq a.L \wedge 0 \leq 0 * 0\}$

sum := 0; max := 0; i := 0

$\{i \leq a.L \wedge s \leq i * m\}$

$T \Rightarrow$
 $0 \leq a.L \wedge 0 \leq 0 * 0$

$\{i \leq a.L \wedge s \leq i * m \wedge \neg(i < a.L)\}$

skip

$\{s \leq a.L * m\}$

$i \leq a.L \wedge$
 $s \leq i * m \wedge \neg(i < a.L) \Rightarrow$
 $s \leq a.L * m$

Step 2: generate VC

$$\{i \leq a.L \wedge s \leq i * m \wedge i < a.L\}$$
$$\Rightarrow$$
$$i + 1 \leq a.L \wedge (m < a[i] \Rightarrow s + a[i] \leq (i + 1) * a[i]) \\ \wedge (\neg(m < a[i]) \Rightarrow s + a[i] \leq (i + 1) * m) \wedge i < a.l$$

```
assert i < a.Length;
```

```
if (max < a[i]) { max := a[i]; }
```

$$i + 1 \leq a.L \wedge s + a[i] \leq (i + 1) * m$$

```
sum := sum + a[i];
```

```
i := i + 1;
```

$$\{i \leq a.L \wedge s \leq i * m\}$$

$$i \leq a.L \wedge s \leq i * m \wedge i < a.L \Rightarrow$$
$$i + 1 \leq a.L \wedge$$
$$(m < a[i] \Rightarrow s + a[i] \leq (i + 1) * a[i]) \wedge \\ (\neg(m < a[i]) \Rightarrow s + a[i] \leq (i + 1) * m) \wedge \\ i < a.l$$

Step 3: Ask the solver

$\forall a \ i \ s \ m .$

$\top \Rightarrow$

$0 \leq a.L \wedge 0 \leq 0 * 0$

\wedge

$i \leq a.L \wedge$

$s \leq i * m \wedge \neg(i < a.L) \Rightarrow$

$s \leq a.L * m$

\wedge

$i \leq a.L \wedge s \leq i * m \wedge i < a.L \Rightarrow$

$i + 1 \leq a.L \wedge$

$(m < a[i] \Rightarrow s + a[i] \leq (i + 1) * a[i]) \wedge$

$(\neg(m < a[i]) \Rightarrow s + a[i] \leq (i + 1) * m) \wedge$

$i < a.l$

SMT



UNSAT



Programs as transition systems

{T}

```
sum := 0;  
max := 0;  
i := 0;
```

$\{i \leq a.L \wedge s \leq i * m\}$

{T}

```
sum' := 0;  
max' := 0;  
i' := 0;
```

$\{i' \leq a.L \wedge s' \leq i' * m'\}$

$s' = 0$
 $\wedge m' = 0$
 $\wedge i' = 0$

$\{i \leq a.L \wedge s \leq i * m \wedge i < a.L\}$

```
assert i < a.Length;  
if (max < a[i]) { max := a[i]; }  
sum := sum + a[i];  
i := i + 1;
```

$\{i \leq a.L \wedge s \leq i * m\}$

$\{i \leq a.L \wedge s \leq i * m \wedge i < a.L\}$

```
assert 0 <= i < a.Length;  
if (max < a[i]) { max' := a[i]; }  
sum' := sum + a[i];  
i' := i + 1;
```

$\{i' \leq a.L \wedge s' \leq i' * m'\}$

$s' = s + a[i]$
 $\wedge m' = m < a[i] ?$
 $a[i] : m$
 $\wedge i' = i + 1$

$\{i \leq a.L \wedge s \leq i * m \wedge \neg(i < a.L)\}$

skip

$\{s \leq a.L * m\}$

$\{i \leq a.L \wedge s \leq i * m \wedge \neg(i < a.L)\}$

sum' := sum; max' := max

$\{s' \leq a.L * m'\}$

$s' = s$
 $\wedge m' = m$
 $\wedge i' = i$

VC for transition systems

$\{T\}$

$$s' = 0 \wedge m' = 0 \wedge i' = 0$$

$\{i' \leq a.L \wedge s' \leq i' * m'\}$

$$T \wedge s' = 0 \wedge m' = 0 \wedge i' = 0 \Rightarrow i' \leq a.L \wedge s' \leq i' * m'$$

$\{i \leq a.L \wedge s \leq i * m \wedge i < a.L\}$

$$s' = a + a[i] \wedge i' = i + 1 \\ \wedge m' = \max(m, a[i])$$

$\{i' \leq a.L \wedge s' \leq i' * m\}$

$$i \leq a.L \wedge s \leq i * m \wedge i < a.L$$

$$\wedge s' = a + a[i] \\ \wedge m' = \max(m, a[i]) \wedge i' = i + 1 \Rightarrow i' \leq a.L \wedge s' \leq i' * m'$$

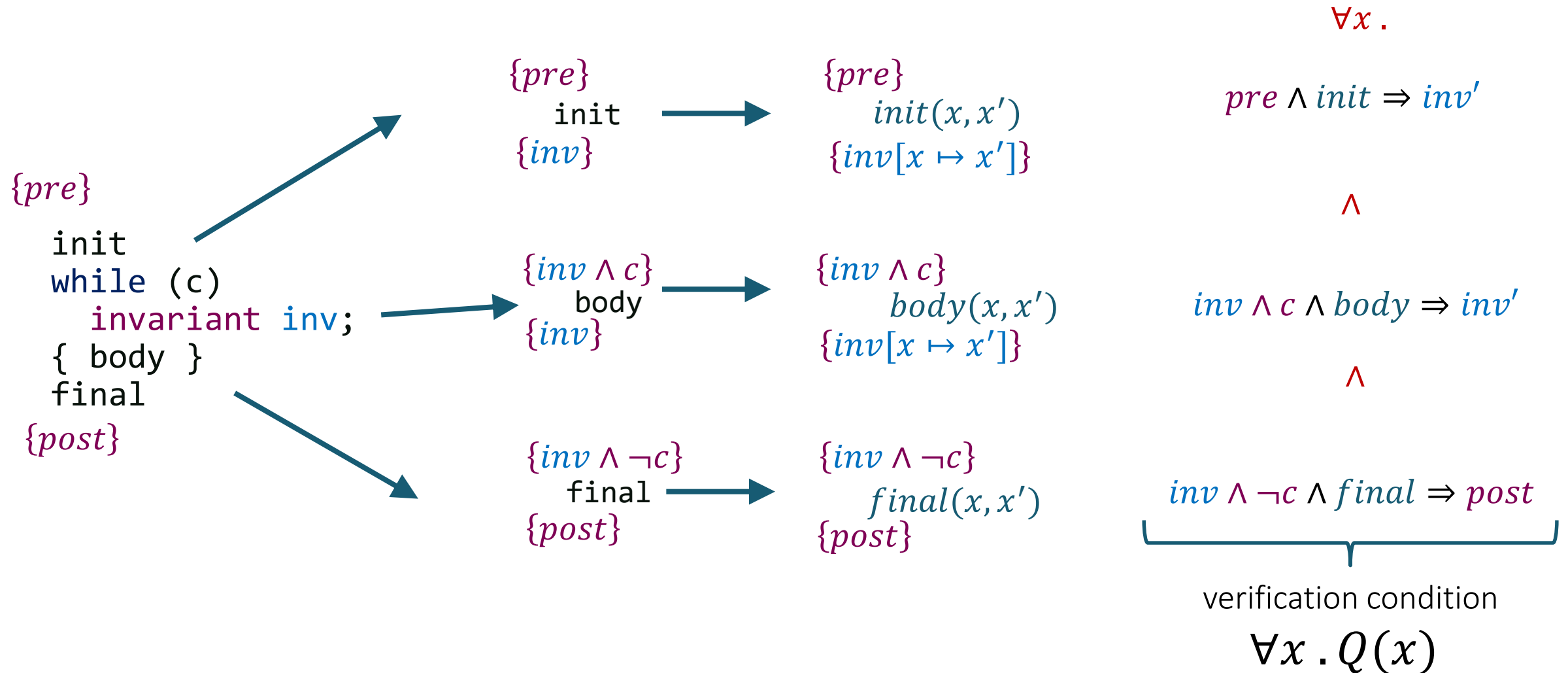
$\{i \leq a.L \wedge s \leq i * m \wedge \neg(i < a.L)\}$

$$s' = s \wedge m' = m \wedge i' = i$$

$\{s' \leq a.L * m'\}$

$$i \leq a.L \wedge s \leq i * m \wedge \neg(i < a.L) \\ \wedge s' = s \wedge m' = m \wedge i' = i \Rightarrow s' \leq a.L * m'$$

Verification with transition systems



From verification to inference

Verification



$$\forall x . Q(x)$$

$$\equiv \exists x . \neg Q(x)$$

SMT



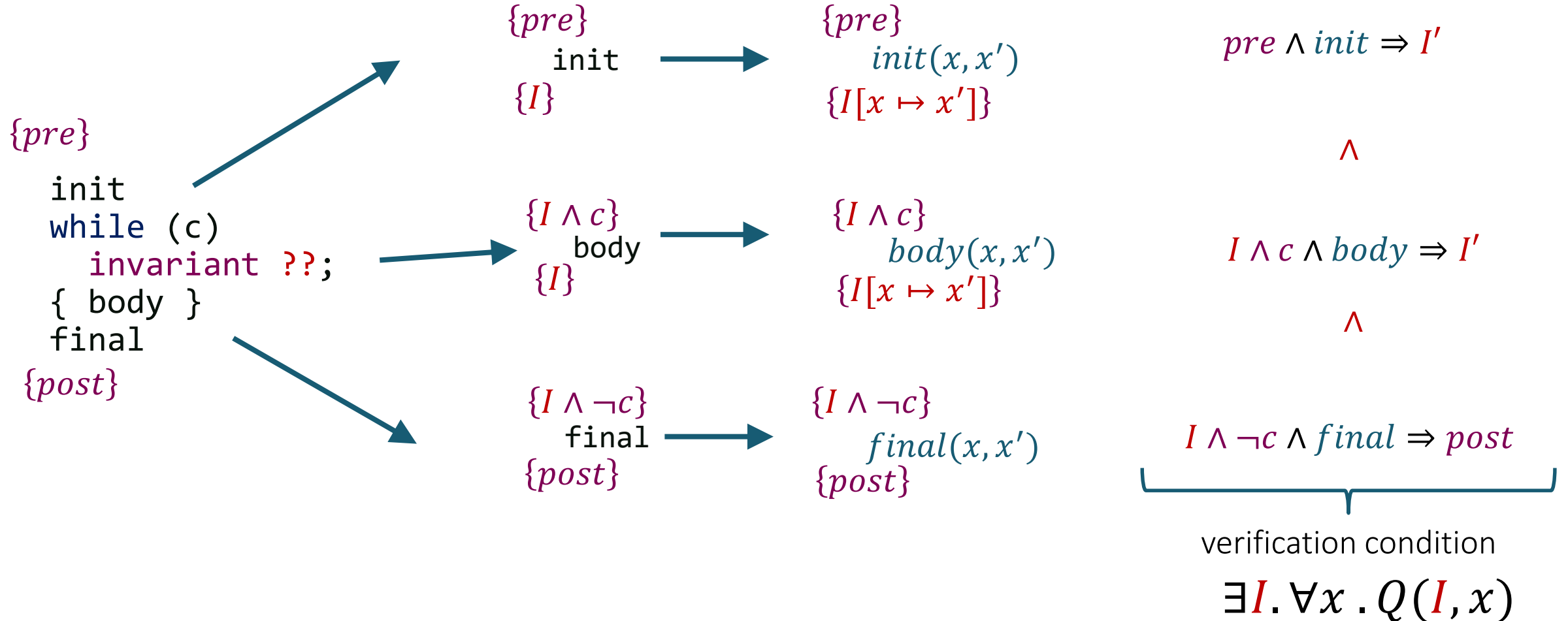
UNSAT / SAT

Inference



$$\exists I . \forall x . Q(I, x)$$

Invariant inference



Horn constraints

Constraints of this form are called Horn constraints (clauses)

$$\phi \Rightarrow I'$$

$$I \wedge \psi \Rightarrow I'$$

$$I \Rightarrow \omega$$

How do we find I ?

- Fix a *domain* (search space)
- Search for an element of the *domain* that makes all Horn clauses true

Horn constraints for SumMax

$$s' = 0 \wedge m' = 0 \wedge i' = 0 \Rightarrow I'$$

$$I \wedge i < a.L \wedge s' = a + a[i] \wedge \\ m' = \max(m, a[i]) \wedge i' = i + 1 \Rightarrow I'$$

$$I \Rightarrow s \leq a.L * m \vee i < a.L$$

(Solution: $I \equiv i \leq a.L \wedge s \leq i * m$)

Can we solve this with...

- Enumerative search?

$$\bullet I ::= T \leq T \mid I \ \&\& \ I$$

$$\bullet T ::= x \mid T.L \mid T + T \mid T * T$$

- Sketch?

$$\exists I. \forall x. Q(I, x)$$



$$\exists c. \forall x. Q(I[c], x)$$

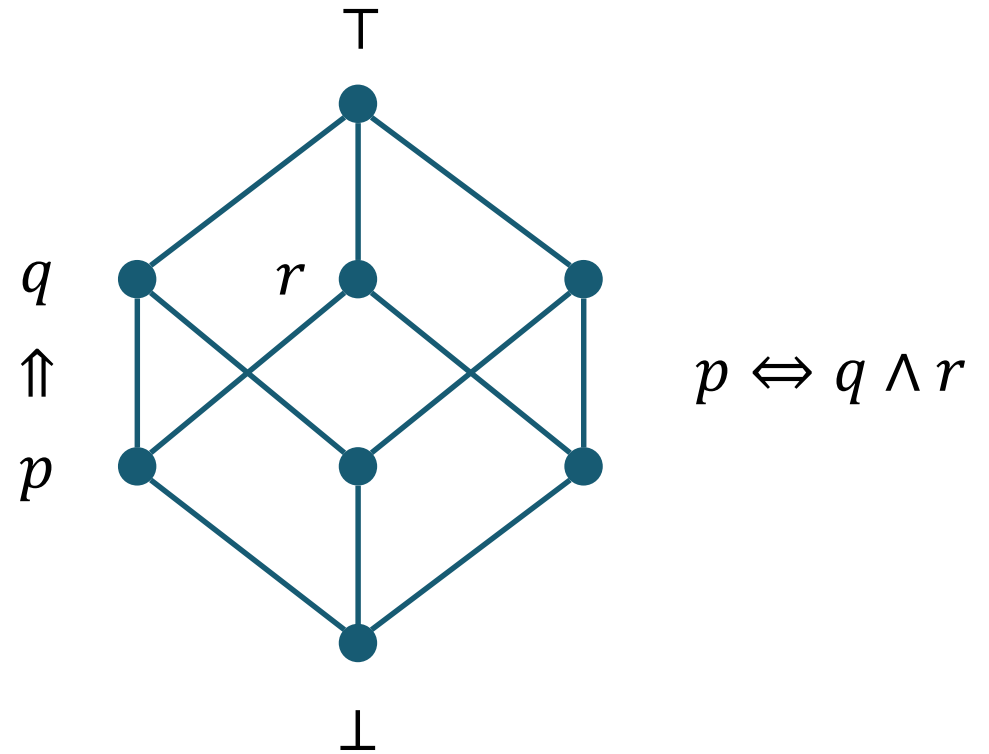
Lattice search

Idea: if domain is a lattice, can search more efficiently

$$\phi \Rightarrow I'$$

$$I \wedge \psi \Rightarrow I'$$

$$I \Rightarrow \omega$$



Least fixpoint (forward search)

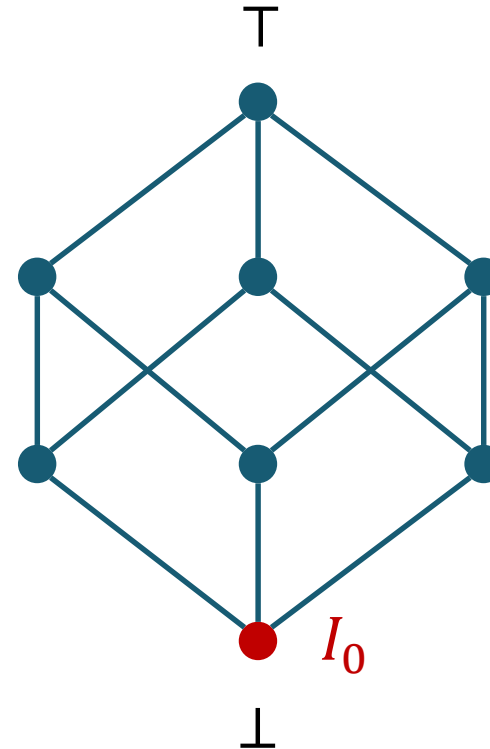


$$\phi \Rightarrow I'$$

$$I \wedge \psi \Rightarrow I'$$

$$I \Rightarrow \omega$$

In every iteration: $I_k \Rightarrow I$



Least fixpoint (forward search)



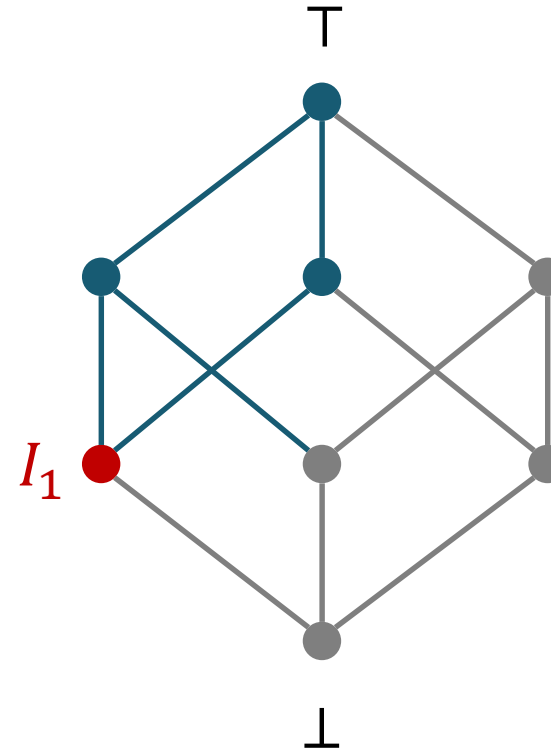
$$\phi \Rightarrow I'$$



$$I \wedge \psi \Rightarrow I'$$

$$I \Rightarrow \omega$$

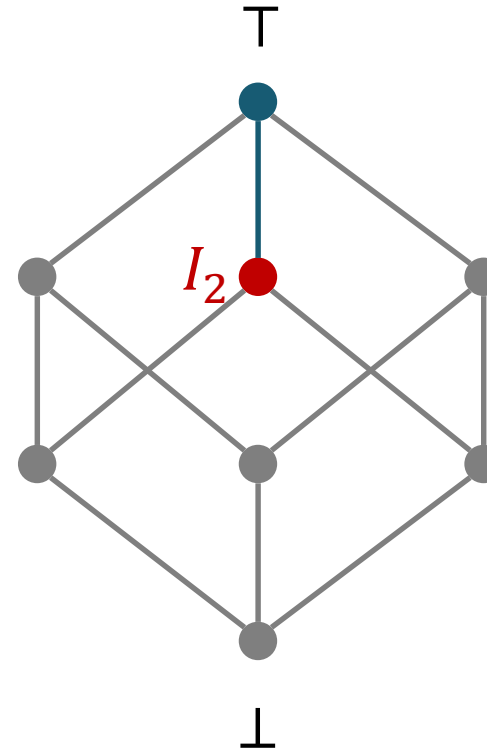
In every iteration: $I_k \Rightarrow I$



Least fixpoint (forward search)

- ✓ $\phi \Rightarrow I'$
- ✓ $I \wedge \psi \Rightarrow I'$
- ✗ ✓ $I \Rightarrow \omega$

- Finds the *strongest* solution
- Did not have to look at all candidates
- Relies on efficient weakening operation



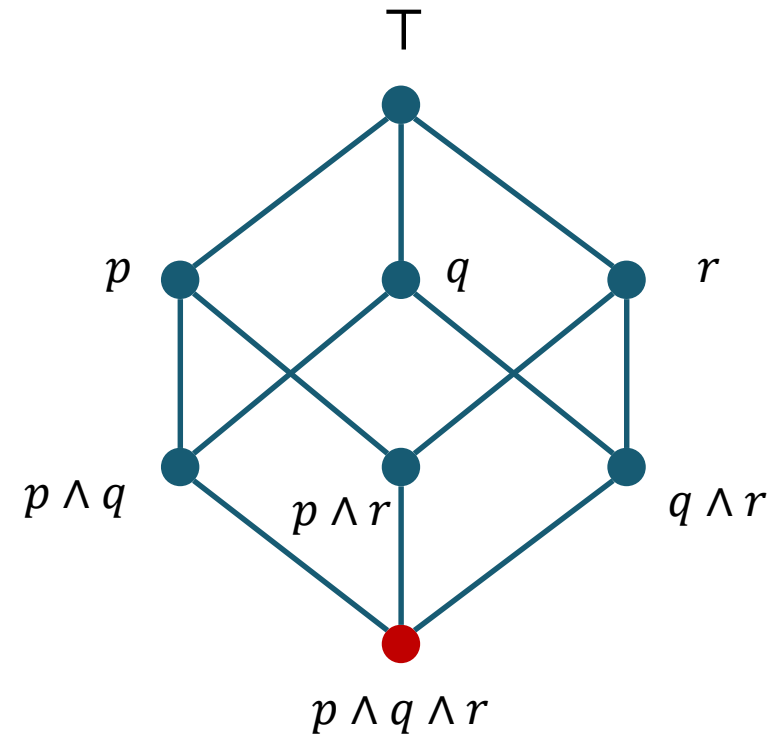
Example

Domain = all conjunctions of predicates from
 $\{p, q, r\} \equiv \{i \leq a.L, i \geq a.L, i \neq a.L\}$

$$i' = 0 \wedge a.L > 0 \Rightarrow I'$$

$$I \wedge i < a.L \wedge i' = i + 1 \Rightarrow I'$$

$$I \Rightarrow i = a.L \vee i < a.L$$



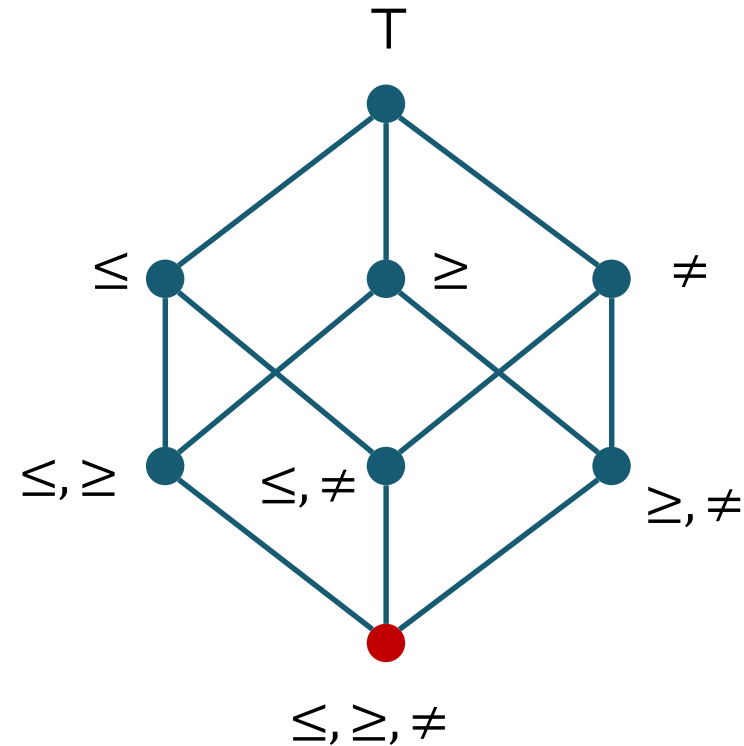
Example

Domain = all conjunctions of predicates from
 $\{p, q, r\} \equiv \{i \leq a.L, i \geq a.L, i \neq a.L\}$

$$i' = 0 \wedge a.L > 0 \Rightarrow I'$$

$$I \wedge i < a.L \wedge i' = i + 1 \Rightarrow I'$$

$$I \Rightarrow i = a.L \vee i < a.L$$



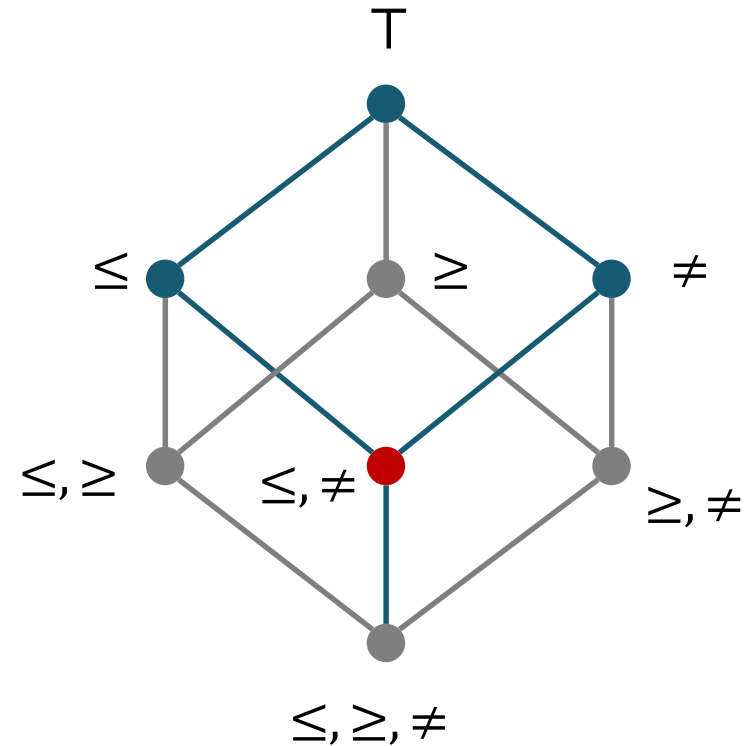
Example

Domain = all conjunctions of predicates from
 $\{p, q, r\} \equiv \{i \leq a.L, i \geq a.L, i \neq a.L\}$

$$i' = 0 \wedge a.L > 0 \Rightarrow I'$$

$$I \wedge i < a.L \wedge i' = i + 1 \Rightarrow I'$$

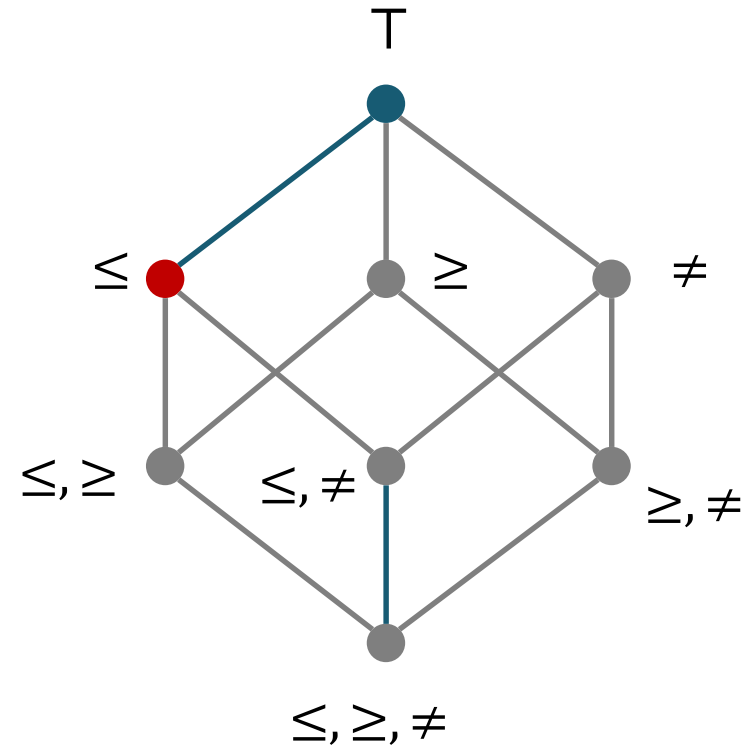
$$I \Rightarrow i = a.L \vee i < a.L$$



Example

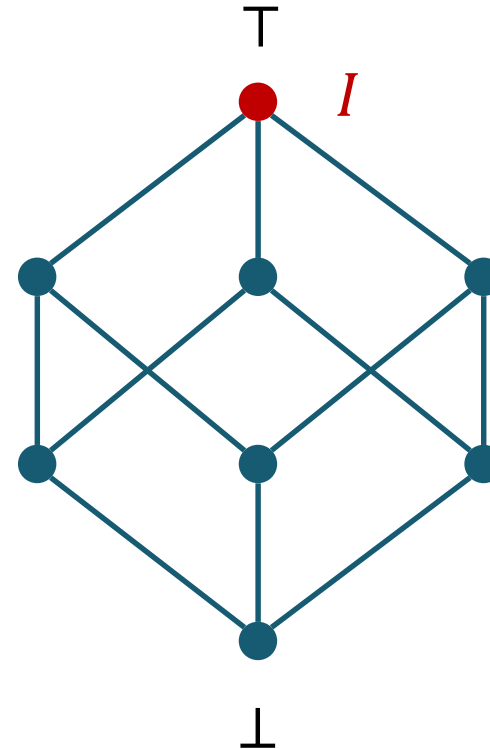
Domain = all conjunctions of predicates from
 $\{p, q, r\} \equiv \{i \leq a.L, i \geq a.L, i \neq a.L\}$

- ✓ $i' = 0 \wedge a.L > 0 \Rightarrow I'$
- ✓ $I \wedge i < a.L \wedge i' = i + 1 \Rightarrow I'$
- ✓ $I \Rightarrow i = a.L \vee i < a.L$



Greatest fixpoint (backward search)

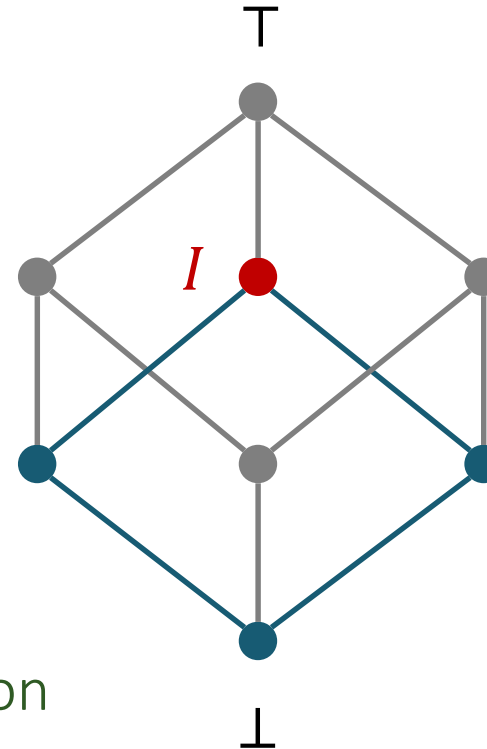
$$\begin{aligned}\phi &\Rightarrow I' \\ I \wedge \psi &\Rightarrow I' \\ \textcircled{\times} \quad I &\Rightarrow \omega\end{aligned}$$



Greatest fixpoint (backward search)

- ✓ $\phi \Rightarrow I'$
- ✓ $I \wedge \psi \Rightarrow I'$
- ✓ $I \Rightarrow \omega$

- Finds the *weakest* solution
- Relies on efficient *strengthening* operation
 - hard to implement



From verification to inference

Verification



$$\forall x . Q(x)$$

$$\equiv \exists x . \neg Q(x)$$

SMT



UNSAT / SAT

Inference



$$\exists I . \forall x . Q(I, x)$$

Fix the domain
lattice \rightarrow lattice search
otherwise \rightarrow
combinatorial search

