Lecture 11
From verification to synthesis

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

AutoProof

VCC

Verifast

…

correct!

can’t proof correctness

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 <= i <= a.Length
  && sum <= 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 → inference → synthesis

Verification

Program logic

verification condition $\forall x \,.\, Q(x)$

$=$

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

SMT

UNSAT SAT

Inference

Program logic

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

unknown formulas for invariants

Synthesis

Program logic

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

unknown formulas for invariants and commands

on the bright side: not much harder than inference!

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;
```

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

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

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

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

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

```
skip
```

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

Step 2: generate VC

$\{\top\}$
$\Rightarrow$
$\{0 \leq a.L \land 0 \leq 0 * 0\}$

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

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

$$\top \Rightarrow 0 \leq a.L \land 0 \leq 0 * 0$$

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

```
skip
```

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

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

Step 2: generate VC

$\{i \le a.L \land s \le i * m \land \ i < a.L\ \}$

$\Rightarrow$

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

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

$i + 1 \le a.L \ \land \ s + a[i] \le (i+1) * m$

```
sum := sum + a[i];
i := i + 1;
```

$\{i \le a.L \land s \le i * m\}$

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

Step 3: Ask the solver

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

$$\neg\ \begin{array}{c}
\top \Rightarrow \\
0 \leq a.L \land 0 \leq 0 * 0 \\
\land \\
i \leq a.L \land \\
s \leq i * m \land \neg(i < a.L) \Rightarrow \\
s \leq a.L\ * m \\
\land \\
i \leq a.L \land s \leq i * m \land\ i < a.L \Rightarrow \\
i + 1 \leq a.L \land \\
(m < a[i] \Rightarrow s + a[i] \leq (i + 1) * a[i]) \land \\
(\neg(m < a[i]) \Rightarrow s + a[i] \leq (i + 1) * m) \land \\
i < a.l
\end{array}$$

$\xrightarrow{\text{SMT}}$ UNSAT

Programs as transition systems

$\{\top\}$
```
sum := 0;
max := 0;
i := 0;
```
$\{i \leq a.L \land s \leq i * m\}$

$\{\top\}$
```
sum' := 0;
max' := 0;
i' := 0;
```
$\{i' \leq a.L \land s' \leq i' * m'\}$

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

$\{i \leq a.L \land s \leq i * m \land i < a.L\}$
```
assert i < a.Length;
if (max < a[i]) { max := a[i]; }
sum := sum + a[i];
i := i + 1;
```
$\{i \leq a.L \land s \leq i * m\}$

$\{i \leq a.L \land s \leq i * m \land i < a.L\}$
```
assert 0 <= i < a.Length;
if (max < a[i]) { max' := a[i]; }
sum' := sum + a[i];
i' := i + 1;
```
$\{i' \leq a.L \land s' \leq i' * m\}$

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

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

$\{i \leq a.L \land s \leq i * m \land \neg(i < a.L)\}$
```
sum' := sum; max' := max
```
$\{s' \leq a.L \ * m'\}$

$$s' = s \\ \land m' = m \\ \land i' = i$$

VC for transition systems

$\{\top\}$

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

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

$\top \wedge \ s' = 0 \wedge m' = 0 \ \wedge \ i' = 0 \Rightarrow$
$i' \leq a.L \wedge s' \leq i' * m'$

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

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

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

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

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

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

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

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

Verification with transition systems

$\{pre\}$

```
init
while (c)
  invariant inv;
{ body }
final
```

$\{post\}$

$\{pre\}$ init $\{inv\}$ $\rightarrow$ $\{pre\}\ init(x, x')\ \{inv[x \mapsto x']\}$

$\{inv \land c\}$ body $\{inv\}$ $\rightarrow$ $\{inv \land c\}\ body(x, x')\ \{inv[x \mapsto x']\}$

$\{inv \land \neg c\}$ final $\{post\}$ $\rightarrow$ $\{inv \land \neg c\}\ final(x, x')\ \{post\}$

$$\underbrace{\forall x\,.\ \begin{array}{c} pre \land init \Rightarrow inv' \\ \land \\ inv \land c \land body \Rightarrow inv' \\ \land \\ inv \land \neg c \land final \Rightarrow post \end{array}}_{\text{verification condition}}$$

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

From verification to inference

Verification

$$\forall x\,.\,Q(x)$$
$$=$$
$$\exists x\,.\,\neg Q(x)$$

SMT

UNSAT / SAT

Inference

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

Invariant inference

$\{pre\}$

```
init
while (c)
  invariant ??;
{ body }
final
```

$\{post\}$

$\{pre\}$
`init`
$\{I\}$

$\{pre\}$
$init(x, x')$
$\{I[x \mapsto x']\}$

$\{I \wedge c\}$
`body`
$\{I\}$

$\{I \wedge c\}$
$body(x, x')$
$\{I[x \mapsto x']\}$

$\{I \wedge \neg c\}$
`final`
$\{post\}$

$\{I \wedge \neg c\}$
$final(x, x')$
$\{post\}$

$$\exists I . \forall x .$$

$$pre \wedge init \Rightarrow I'$$

$$\wedge$$

$$I \wedge c \wedge body \Rightarrow I'$$

$$\wedge$$

$$I \wedge \neg c \wedge final \Rightarrow post$$

verification condition

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

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...

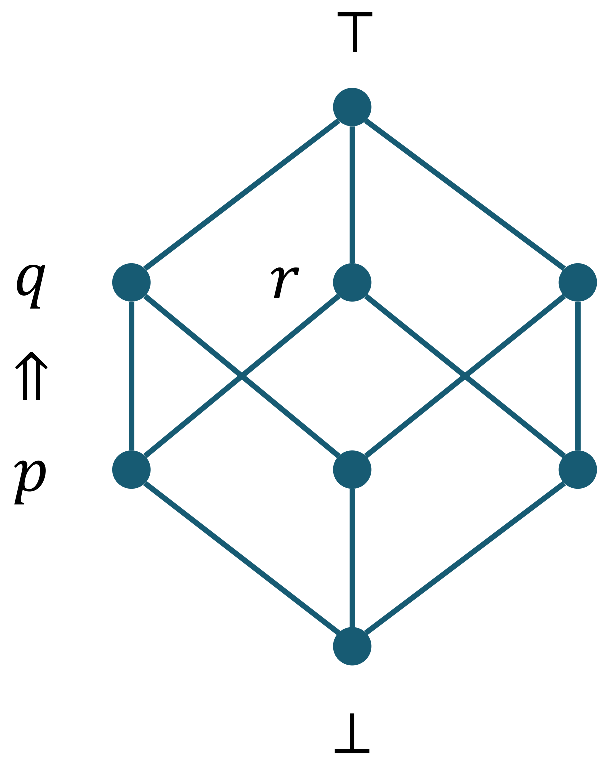
- Enumerative search?
  - `I ::= T <= T | I && I`
  - `T ::= x | T.L | T + T | T * T`
- Sketch?

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

Lattice search

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

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



$p \Leftrightarrow q \wedge r$

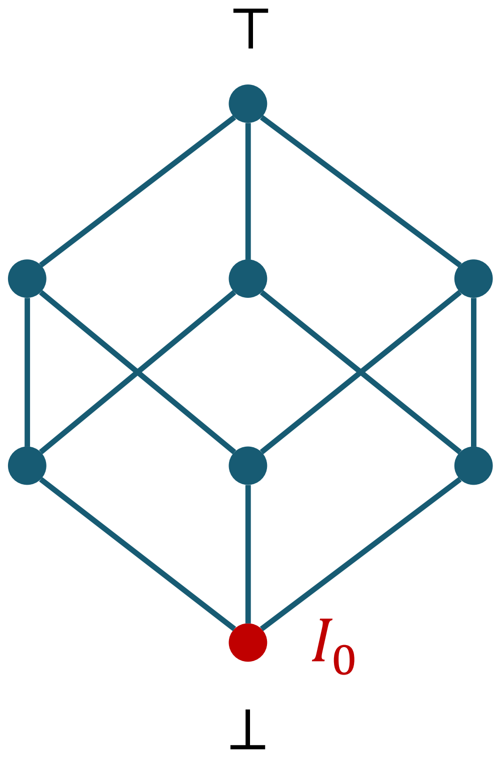
Least fixpoint (forward search)

$\boxed{\phi \Rightarrow I'}$

$I \wedge \psi \Rightarrow I'$

$I \Rightarrow \omega$

In every iteration: $I_k \Rightarrow I$



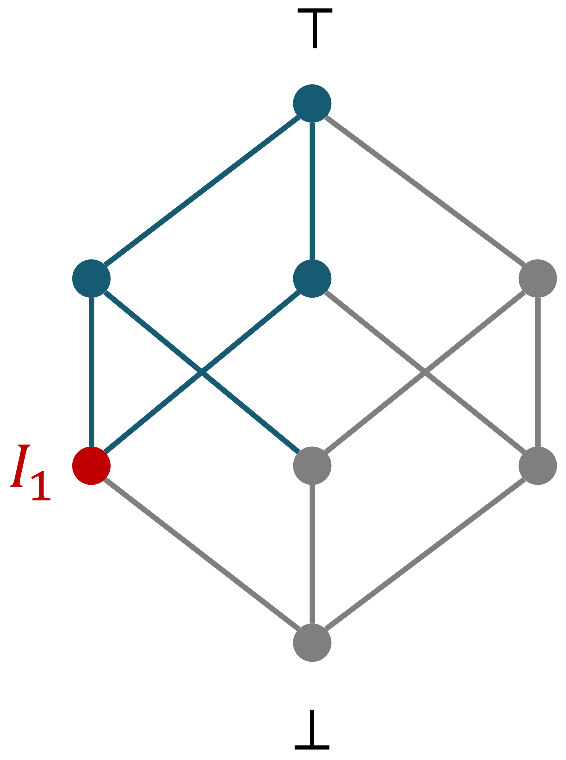
Least fixpoint (forward search)

✔ $\phi \Rightarrow I'$

✘ $\boxed{I \wedge \psi \Rightarrow I'}$

$I \Rightarrow \omega$

In every iteration: $I_k \Rightarrow I$

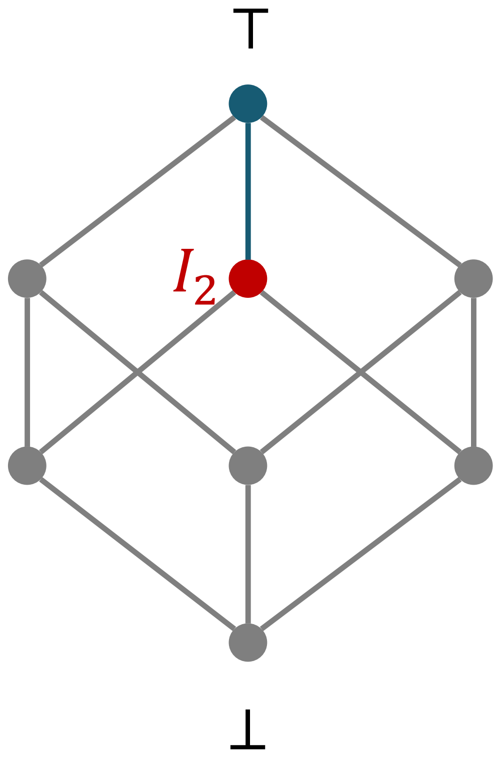


Least fixpoint (forward search)

$\phi \Rightarrow I'$

$I \wedge \psi \Rightarrow I'$

$I \Rightarrow \omega$



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

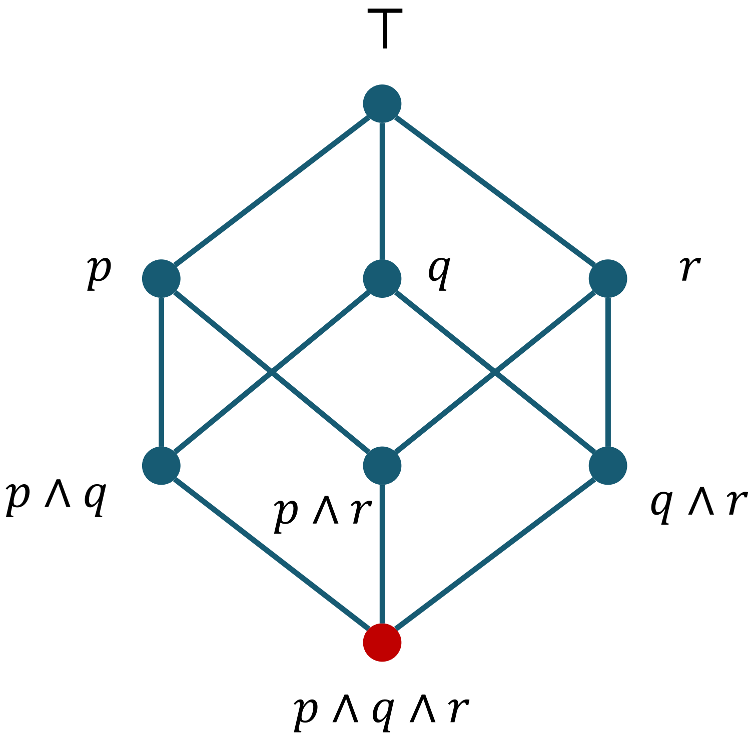
Example

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

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

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

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



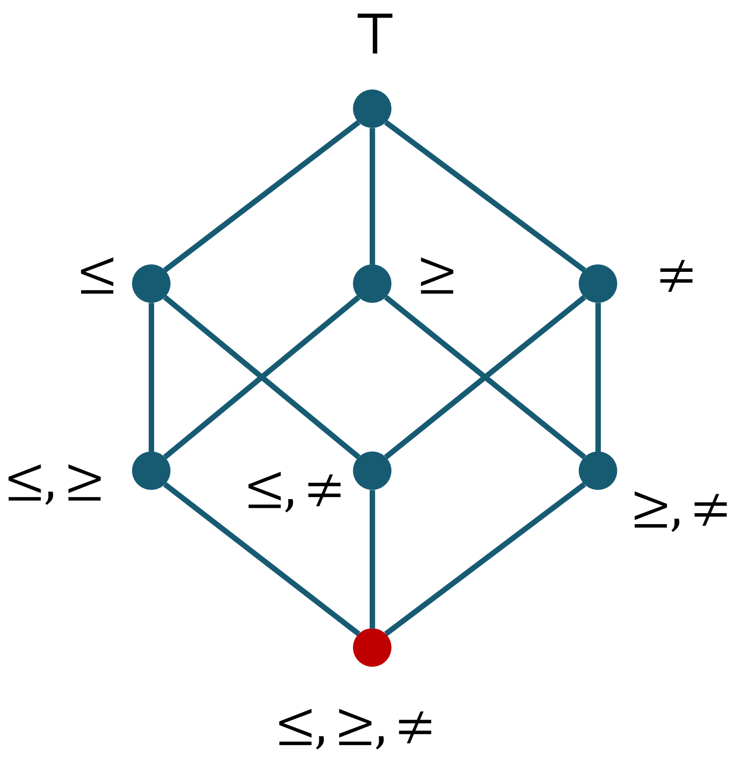
Example

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

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

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

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



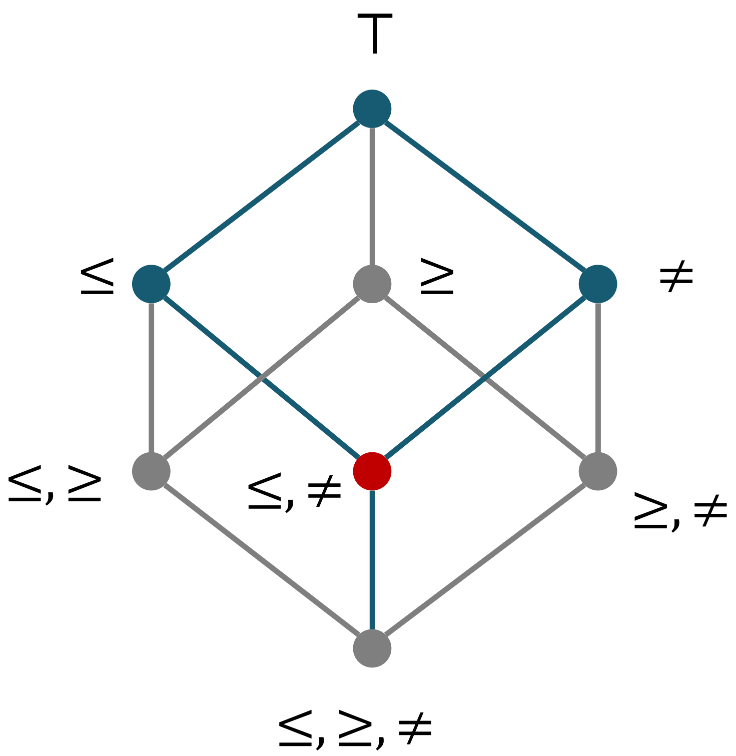
Example

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

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

$$\boxed{I \land i < a.L \land i' = i + 1 \Rightarrow I'}$$

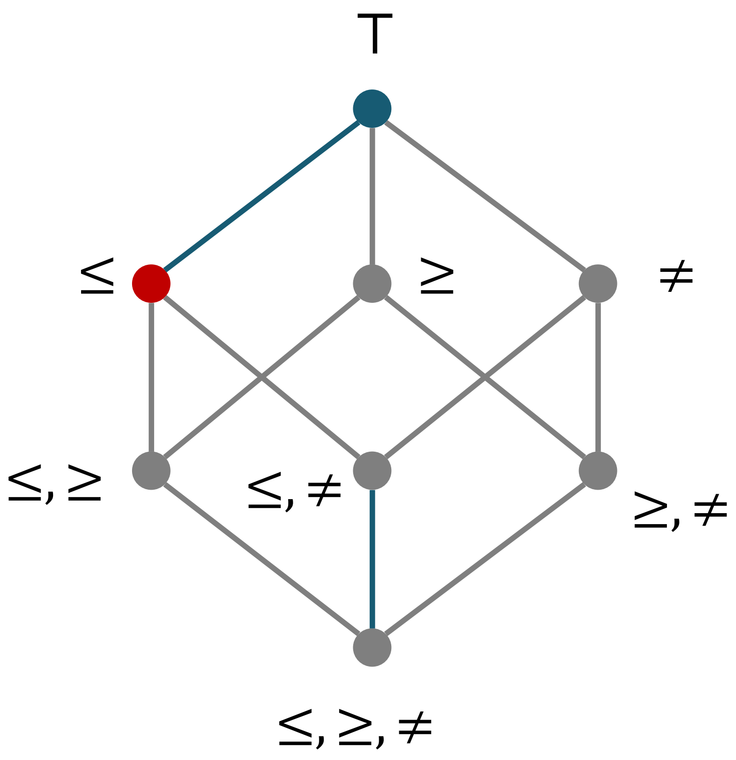
$$I \Rightarrow i = a.L \lor i < a.L$$



Example

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

- ✅ $i' = 0 \land a.L > 0 \Rightarrow I'$
- ✅ $I \land i < a.L \land i' = i + 1 \Rightarrow I'$
- ✅ $I \Rightarrow i = a.L \lor i < a.L$

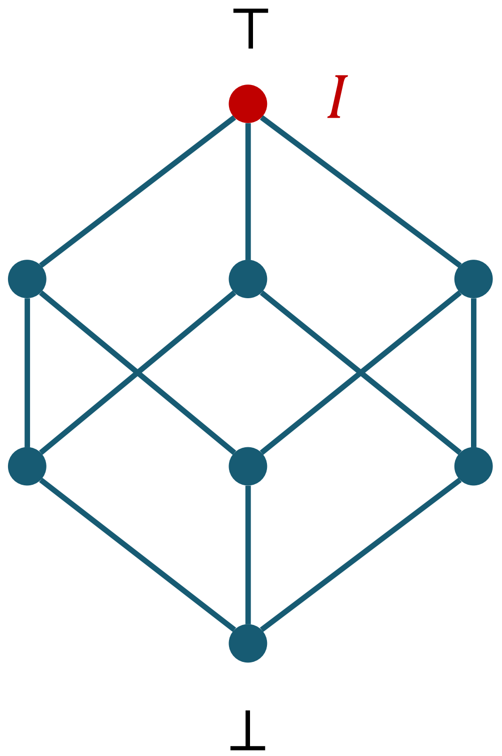


Greatest fixpoint (backward search)

$$\phi \Rightarrow I'$$

$$I \land \psi \Rightarrow I'$$

$$\boxed{I \Rightarrow \omega}$$

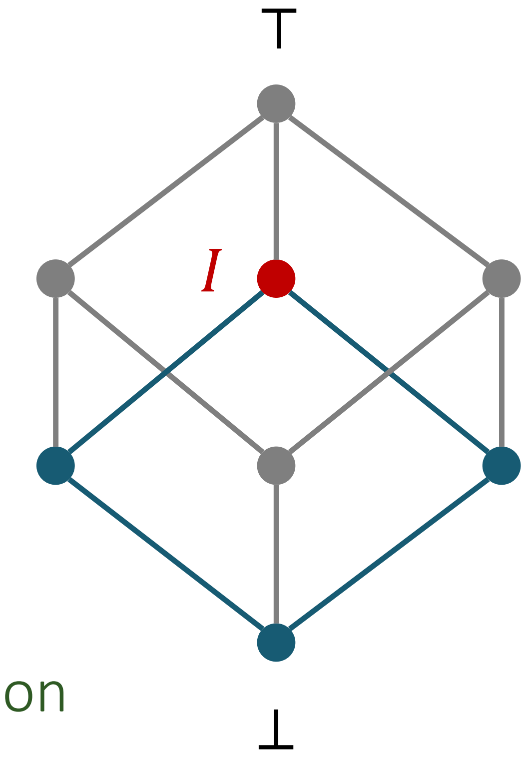


Greatest fixpoint (backward search)

$\phi \Rightarrow I'$

$I \wedge \psi \Rightarrow I'$

$I \Rightarrow \omega$



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

From verification to inference

Verification

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

$$=$$

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

SMT

UNSAT / SAT

Inference

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

Fix the domain

lattice → lattice search

otherwise → combinatorial search

From inference to synthesis

Verification

$$\forall x\,.\,Q(x)$$
$$=$$
$$\exists x\,.\,\neg Q(x)$$

SMT

UNSAT / SAT

Inference

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

Fix the domain lattice → lattice search
otherwise → combinatorial search

Synthesis

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

Program synthesis

$\{pre\}$
```
  ??
  while (??)
    invariant ??;
  { ?? }
  ??
```
$\{post\}$

$\{pre\}$
$S_i(x, x')$
$\{I[x \mapsto x']\}$

$\{I \wedge G_0\}$
$G_1 \rightarrow S_1(x, x')$
$G_2 \rightarrow S_2(x, x')$
$\{I[x \mapsto x']\}$

$\{I \wedge \neg c\}$
$S_f(x, x')$
$\{post\}$

$\exists S\; G\; I . \forall x\, .$

$pre \wedge S_i \Rightarrow I'$

$\wedge$

$I \wedge G_0 \wedge G_1 \wedge S_1 \Rightarrow I'$

$I \wedge G_0 \wedge G_2 \wedge S_2 \Rightarrow I'$

$\wedge$

$I \wedge \neg G_0 \wedge S_f \Rightarrow post$

synthesis condition

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

Synthesis constraints

Similar to Horn constraints but not quite

$$I \wedge G_i \wedge S_i \wedge \psi \Rightarrow I'$$

$$I \wedge G_i \wedge S_i \Rightarrow \omega$$

$$\top \Rightarrow G_i \vee G_j$$

Domain for I, G_i: like in inference

Domain for $S_i = \{x' = e_x \wedge y' = e_y \wedge \cdots \mid e_x, e_y, \ldots \in Expr\}$

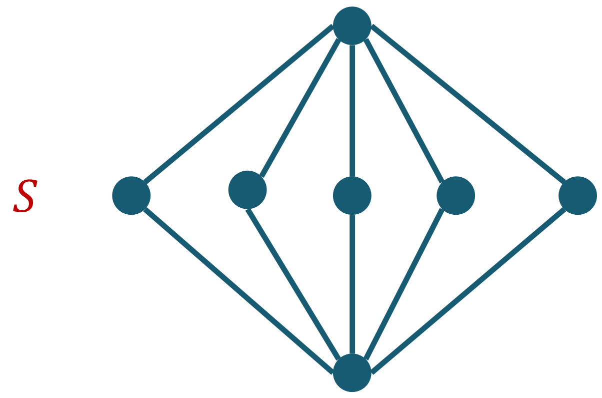
- conjunction of equalities, one per variables

Solving synthesis constraints

$$I \wedge G_i \wedge S_i \wedge \psi \Rightarrow I'$$

$$I \wedge G_i \wedge S_i \Rightarrow \omega$$

$$\top \Rightarrow G_i \vee G_j$$



Can we solve this with...

- Enumerative search?
  - Sure (slow)
- Sketch?
  - Yep!
  - Look we made an unbounded synthesizer out of Sketch!
- Lattice search?
  - That's what VS3 does
  - Great for I, G, not so great for S (why?)

This week

We can reason about unbounded loops using *loop invariants*

- Hoare logic soundly translates a program with a loop (and invariant) into three straight-line programs

We can synthesize a program with a loop by synthesizing *those three straight-line programs* (and the invariant)!

- Can use existing synthesis techniques

Powerful idea: to synthesize a provably correct program, look for the program *and its proof* together