Lecture 12
From verification to synthesis (II)

Nadia Polikarpova

Verification → inference → synthesis

Verification	Inference	Synthesis
Program logic ↓	Program logic ↓	Program logic ↓
verification condition $\forall x\,.\,Q(x)$	$\exists I.\,\forall x\,.\,Q(I,x)$	$\exists I\ P.\,\forall x\,.\,Q(I,P,x)$
$=$ $\exists x\,.\,\neg Q(x)$	unknown formulas for invariants	unknown formulas for invariants and commands
SMT ↓		
UNSAT ✓ ✗ SAT		on the bright side: not much harder than inference!

Verification with transition systems

$\{pre\}$

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

$\{post\}$

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

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

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

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

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

Invariant inference

$\{pre\}$

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

$\{post\}$

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

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

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

$\exists I . \forall x .$

$pre \land init \Rightarrow I'$

$\land$

$I \land c \land body \Rightarrow I'$

$\land$

$$\underbrace{I \land \neg c \land final \Rightarrow post}_{\text{verification condition}}$$

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

Horn constraints

Constraints of this form are called Horn constraints (clauses)

$$\phi \Rightarrow I'$$

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

$$I \Rightarrow \omega$$

Can be solved using lattice search

From verification to inference

Verification

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

$$=$$

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

SMT

UNSAT / SAT

Inference

$$\exists \color{red}{I}\color{black}{.\,\forall x \,.\, Q(}\color{red}{I}\color{black}{, x)}$$

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 ??;
{ ?? }
??
```

$\{post\}$

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

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

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

$$\underbrace{\begin{gathered}\exists S\ G\ I . \forall x\ . \\ pre \wedge S_i \Rightarrow I' \\ \wedge \\ I \wedge G_0 \wedge G_1 \wedge S_1 \Rightarrow I' \\ I \wedge G_0 \wedge G_2 \wedge S_2 \Rightarrow I' \\ \wedge \\ I \wedge \neg G_0 \wedge S_f \Rightarrow post\end{gathered}}_{\text{synthesis condition}}$$

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

Synthesis constraints

Similar to Horn constraints but not quite

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

$$I \land G_i \land S_i \Rightarrow \omega$$

$$\top \Rightarrow G_i \lor G_j$$

Domain for I, G_i: like in inference

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

- conjunction of equalities, one per variables

Solving synthesis constraints

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

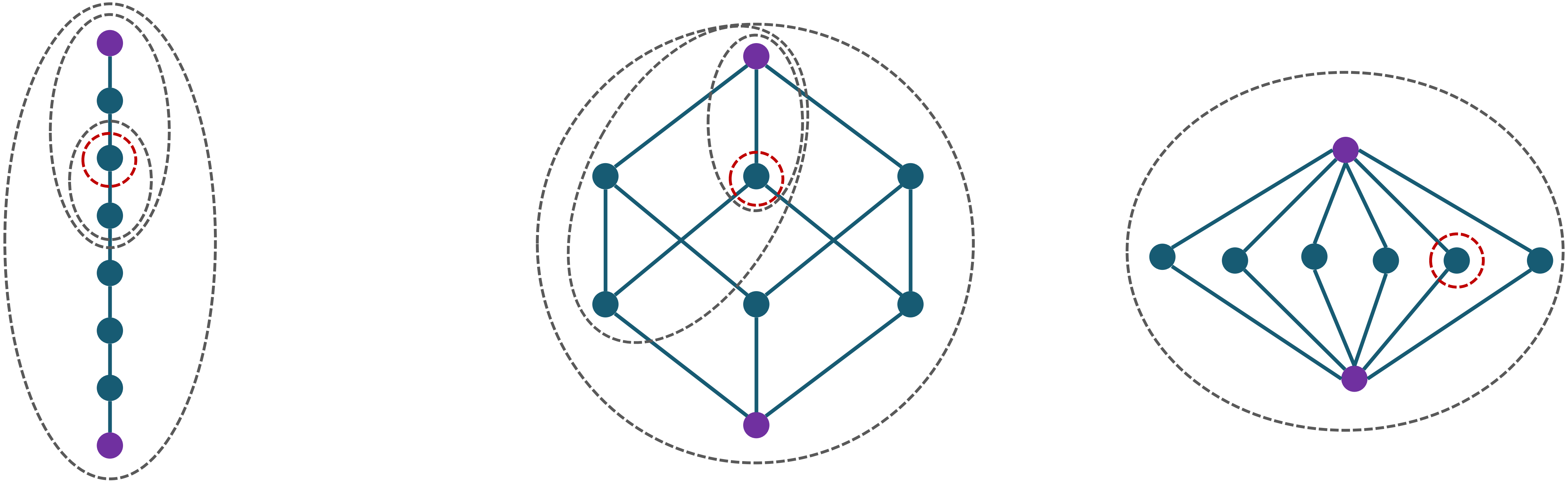
$$I \wedge G_i \wedge S_i \Rightarrow \omega$$

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

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?)

Lattice search



Verification → inference → 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

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

Fix the domain
lattice → lattice search
otherwise →
combinatorial search

VS3: contributions

Proof-theoretic synthesis

- = we can make constraint-based synthesis unbounded by synthesizing loop invariants alongside programs
- this idea was later used e.g. in Natural Synthesis [Qi, Solar-Lezama, OOPSLA'17]

Reusing existing invariant inference tools

- encoding as a guarded transition system (makes synthesis constraints look like Horn clauses)
- using lattice search for synthesis
- as Horn solver get better, this approach should too

VS3: limitations

Requires a lot of user input

- Pre/postcondition, domain constraints, resource constraints
- How does this compare to other synthesizers we have seen?

Incomplete

- inherits incompleteness from the verifier
- even if a program exists in the domain, it might not be verifiable

Slow / unpredictable performance

- unavoidable with general verification

VS3: questions

Behavioral constraints? structural constraints? search strategy?

- pre-/postconditions in logic
- domain and resource constraints
- constraint-based (Horn clause solving)

Trivial solution to Bresenham’s algorithm

- Set all unknowns to False

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