# Lecture 12
# Deductive Reasoning in Program Synthesis

*Nadia Polikarpova*

# Last 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

# Verification with transition systems

$\{pre\}$

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

$\{post\}$

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

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

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

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

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

# From verification to inference

Verification

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

$$=$$

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

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

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

$$\underbrace{\exists I . \forall x . \; pre \wedge init \Rightarrow I' \;\wedge\; I \wedge c \wedge body \Rightarrow I' \;\wedge\; I \wedge \neg c \wedge final \Rightarrow post}_{\text{verification condition}}$$

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

# 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 → 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 ??;
{ ?? }
??
```

$\{post\}$

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

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

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

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

$pre \wedge S_i \Rightarrow I'$

$\wedge$

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

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

$\wedge$

$$\underbrace{I \wedge \neg G_0 \wedge S_f \Rightarrow post}_{\text{synthesis condition}}$$

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

# 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

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

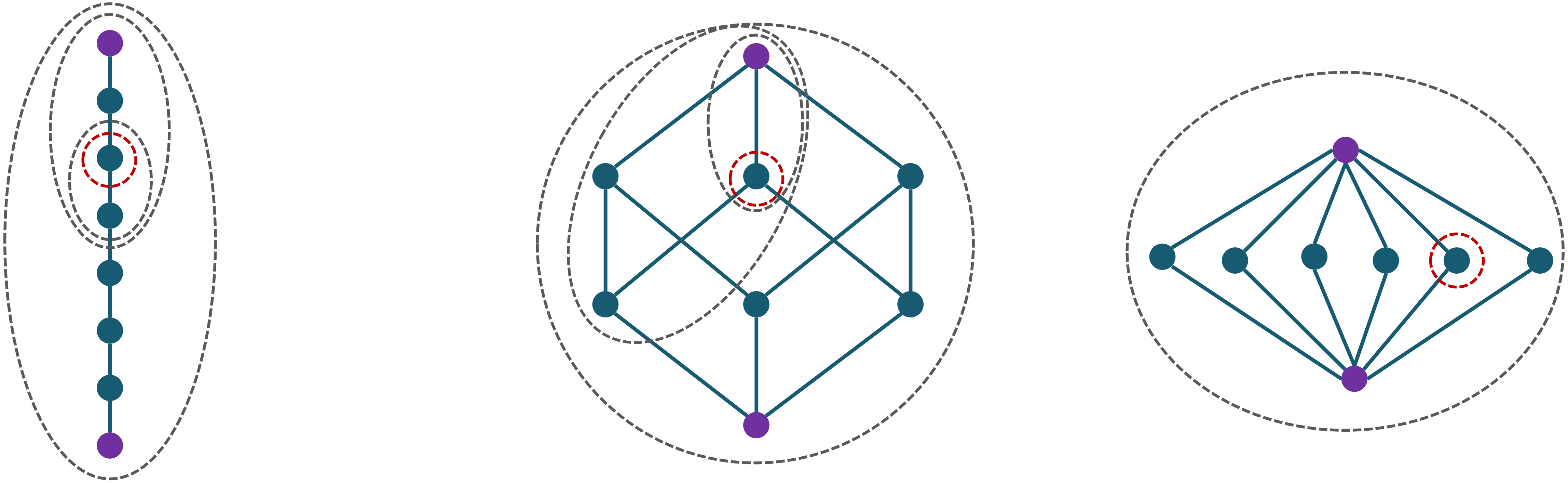
$$I \wedge G_i \wedge S_i \Rightarrow \omega$$

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

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?)

# 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

# Map of the unit

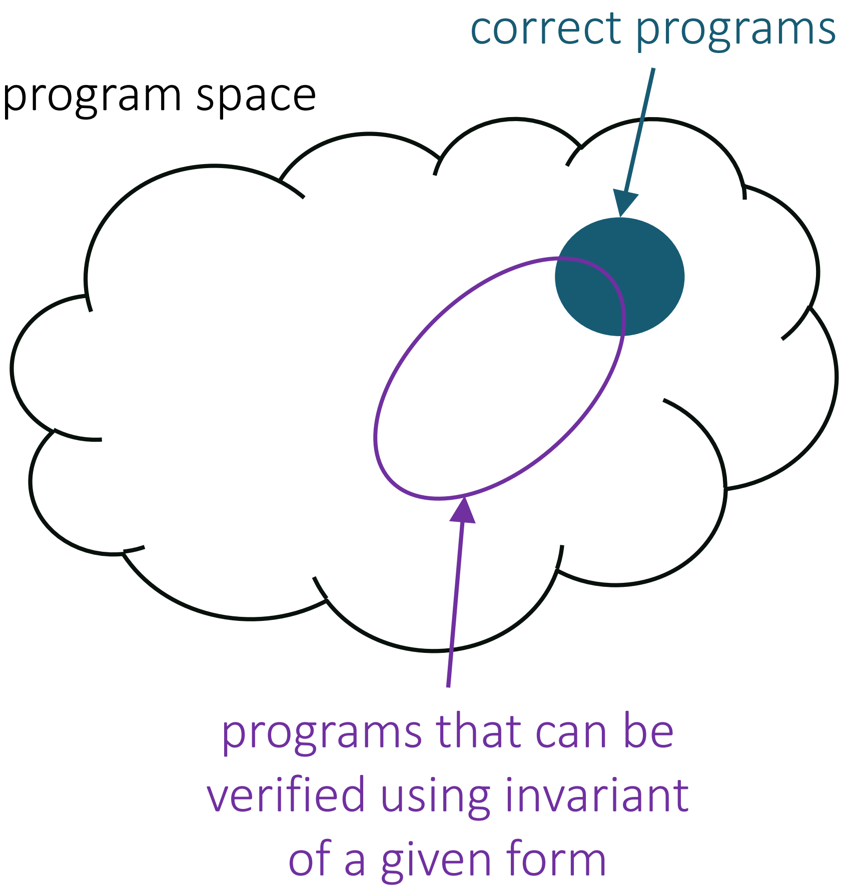
Constraint-based synthesis

- 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?

# The big picture



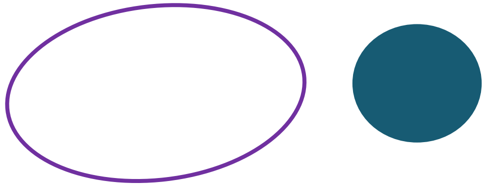

Program verification is conservative

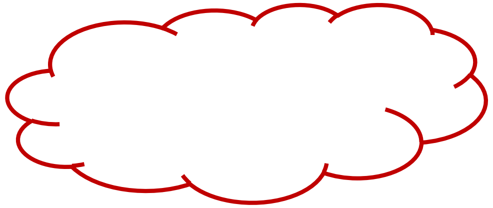
- Not all correct programs can be verified

For synthesis, this is a feature!

- Only need to explore verifiable programs

Caveats

- This can happen: 
  - but if you want a verified program, there's no way around it
- We also need to search for the invariant



# Deductive reasoning for synthesis

**Main idea:** Look for the proof to find the program

- The space of valid program derivations is smaller than the space of all programs
- The result is provably correct!

Applications:

- Constraint-based search: use loop invariants to encode the space of correct looping programs
- Enumerative search: prune unverifiable candidates early
- Deductive search: search in the space of provably correct transformations / decompositions

# Deductive Synthesis

# Deductive synthesis

The synthesis problem:

- Find $x$ such that $Q(a, x)$ whenever $P(a)$

Using semantic-preserving transformations, gradually rewrite the problem above into:

- Find $T$ such that T whenever $P(a)$
- where $T$ is a term that does not mention $x$

Toy example:

- “Find $x$ such that $x + x = 4a$“ → “Find $x$ such that $2x = 4a$“→
  “Find $2y$ such that $4y = 4a$“→ “Find $2y$ such that $y = a$“→
  “Find $2a$ such that T“

# Deductive synthesis: challenges

Define a set of transformation rules that is sound

- A solution to the transformed problem is a solution to the original problem

... and complete

- All programs we care about can be derived

In most cases, multiple rules apply to a problem

- Need a search strategy!

# Two approaches

## Transformation rules

A set of inference rules for decomposing a synthesis problem into simpler problems

- Axioms (terminal rules) for solving elementary problems
- Rules have side conditions to prove

Depth- or best-first search in the space of derivations

[Manna, Waldinger'79]
[Kneuss et al.'13]

## Theorem proving

today

Extract the program from a constructive proof of $\exists x. \forall a. P(a) \Rightarrow Q(a, x)$

- Instead of inventing custom rules, reuse an existing theorem prover
- ... but augment its rules with term extraction
- Reuse the prover's search strategy!

[Green'69] [Manna, Waldinger'80]
Coq and Sketch, in some sense...

# Synthesis as theorem proving: intuition

Axioms: $1.\, \mathrm{head}(x :: xs) = x$ $\quad$ $2.\, \mathrm{tail}(x :: xs) = xs$

Prove: $\exists l.\, \mathrm{head}(l) = 5 \wedge \mathrm{tail}(l) = []$

$\mathrm{head}(l) = 5 \wedge \mathrm{tail}(l) = []$

- Unify first conjunct with 1, substituting $l \rightarrow x :: xs, x \rightarrow 5$

$\mathrm{head}(5 :: xs) = 5 \wedge \mathrm{tail}(5 :: xs) = []$

- Unify second conjunct with 2, substituting $x \rightarrow 5, xs \rightarrow []$

$\mathrm{head}(5 :: []) = 5 \wedge \mathrm{tail}(5 :: []) = []$

$\top$

# Synthesis as theorem proving

[Manna, Waldinger'80]

Sequent:

| assertions | goals | output |
|---|---|---|
| $A_i(a, x)$ | | |
| | $G_i(a, x)$ | $t_i(a, x)$ |

Meaning: if $\bigwedge_i \forall x. A_i$ holds, then $\bigvee_i \exists x. G_i$ holds

- and the corresponding $t_i$ is an acceptable solution

# Synthesis as theorem proving

[Manna, Waldinger'80]

Synthesis problem: "Find $x$ such that $Q(a, x)$ whenever $P(a)$"

| assertions | goals | output |
| --- | --- | --- |
| $P(a)$ | | |
| | $Q(a, x)$ | $x$ |

Apply inference rules to add new assertions and goals

- eventually arrive to (where $t$ does not contain $x$)

| | | |
| --- | --- | --- |
| | $\top$ | $t$ |

# Inference rules

Splitting

- E.g. split assertion $A_1 \wedge A_2$ into two assertions $A_1$ and $A_2$

Transformation

- Apply a rewrite rule $s \rightarrow t$ to a subterm of assertion / goal
- Apply the unifying substitution of the rewrite to the output!

Resolution

- Let $A[P]$, $B[Q]$ two assertions, and let $P\theta = Q\theta$; add assertion $A[P\theta \rightarrow \top] \vee B[Q\theta \rightarrow \bot]$

Induction

- Introduce an induction hypothesis

# Example: quotient and remainder

Specification:

$$div(i,j), rem(i,j) \Leftarrow \text{find } (q,r) \text{ s.t.}$$

$$i = q * j + r \quad \wedge \quad 0 \le r < j$$

$$\text{where } 0 \le i \wedge 0 < j$$

# Example: base case

| | assertions | goals | outputs $div(i,j)$ | $rem(i,j)$ |
|---|---|---|---|---|
| | 1. $0 \le i \land 0 < j$ | 2. $i = q * j + r \;\land\; 0 \le r < j$ | $q$ | $r$ |
| and-split 1 | 3. $0 \le i$<br>4. $0 < j$ | | | |
| trans 2 $\quad 0 * v \to 0$<br>$[q \to 0, v \to j]$ | | 5. $i = 0 + r \;\land\; 0 \le r < j$ | $0$ | $r$ |
| trans 5 $\quad 0 + v \to v$<br>$[v \to r]$ | | 6. $i = r \;\land\; 0 \le r < j$ | $0$ | $r$ |
| resolve 6 & $\quad v = v$<br>$[v \to i, r \to i]$ | | 7. $0 \le i < j$ | $0$ | $i$ |
| resolve 7 & 3<br>$[]$ | | 8. $i < j$ | $0$ | $i$ |

# Example: step case

| | assertions | goals | outputs $div(i,j)$ | outputs $rem(i,j)$ |
|---|---|---|---|---|
| | 3. $0 \leq i$ 4. $0 < j$ | 2. $i = q * j + r \ \wedge \ 0 \leq r < j$ | $q$ | $r$ |
| trans 2 $(u+1) * v \rightarrow u * v + v$ $[q \rightarrow q' + 1, u \rightarrow q', v \rightarrow j]$ | | 9. $i = q' * j + j + r \ \wedge \ 0 \leq r < j$ | $q' + 1$ | $r$ |
| | | 10. $i - j = q' * j + r \ \wedge \ 0 \leq r < j$ | $q' + 1$ | $r$ |
| induction | 11. $(u,v) < (i,j) \Rightarrow$ $0 \leq u \wedge 0 < v \Rightarrow$ $u = div(u,v) * v + rem(u,v)$ $\wedge \ 0 \leq rem(u,v) < v$ | | | |
| resolve 10 and 11 $[u \rightarrow i - j, v \rightarrow j,$ $q' \rightarrow div(i-j,j), r \rightarrow rem(i-j,j)]$ | | 12. $(i-j,j) < (i,j) \wedge$ $0 \leq i - j \wedge 0 < j$ | $div(i-j,j) + 1$ | $rem(i-j,j)$ |
| … | | 13. $\neg(i < j)$ | … | |

# Example: put them together

| | assertions | goals | outputs $div(i,j)$ | $rem(i,j)$ |
|---|---|---|---|---|
| | | 8. $i < j$ | $0$ | $i$ |
| | | 13. $\neg(i < j)$ | $div(i-j,j)+1$ | $rem(i-j,j)$ |
| resolve 8 & 13<br>[] | | 13. T | if $i < j$<br>then $0$<br>else $div(i-j,j)+1$ | if $i < j$<br>then $i$<br>else $rem(i-j,j)$ |