Lecture 13
Deductive Synthesis

Nadia Polikarpova

Map of the module

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

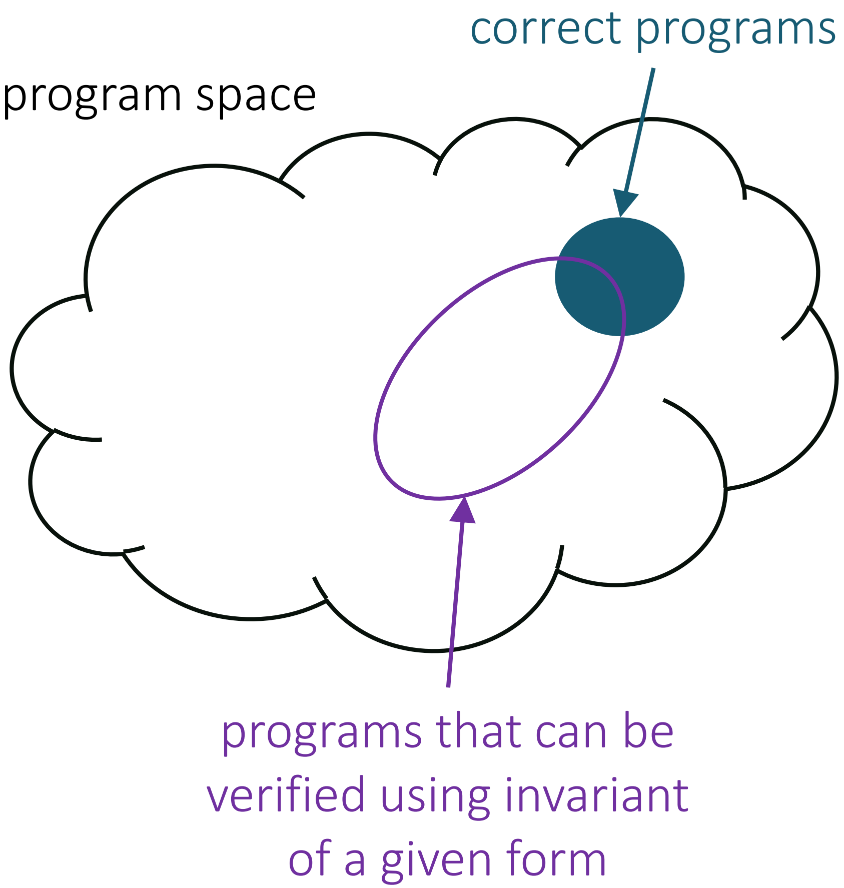
Deductive synthesis

- How to derive programs from specifications?

Enumerative synthesis with deduction

- How to use deductive reasoning to guide the search?

The big picture



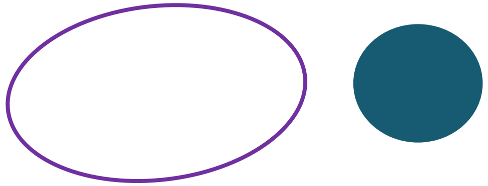
Program verification is conservative

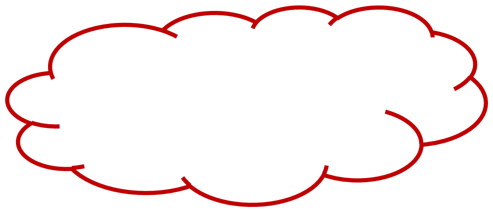
- Not all correct programs can be verified

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

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]

Two approaches

Theorem proving

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]

Synthesis as theorem proving: intuition

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

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

$\text{head}(l) = 5 \land \text{tail}(l) = []$

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

$\text{head}(5 :: xs) = 5 \land \text{tail}(5 :: xs) = []$

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

$\text{head}(5 :: []) = 5 \land \text{tail}(5 :: []) = []$

$\top$

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

- Split assertion $A_1 \land A_2$ into two assertions A_1 and A_2
- Split a goal $G_1 \lor G_2$ into two goals G_1 and G_2

Transformation

- Given a rewrite rule $s \to t$, and assertion / goal $F[s\theta]$, add assertion / goal $F[t\theta]$
- Apply the unifying substitution θ to the output!

Inference rules

Resolution

- Intuition: given assertion P and goal $G[Q]$, where $P\theta = Q\theta$, we can add goal $G[Q\theta \to \top]$
- The real rules is a bit more general

Induction

- If $\mathrm{A}[n]$ conjunctin of all assumptions, $G[n]$ is a goal, add assumption $\forall i < n. \mathrm{A}[i] \Rightarrow G[i]$

Example: quotient and remainder

Specification:

$$div(i,j), rem(i,j) \Lleftarrow \text{find } (q,r) \text{ s.t.}$$
$$i = q * j + r \;\wedge\; 0 \leq r < j$$
$$\text{where } 0 \leq i \wedge 0 < j$$

Transformation rules:

$$0 * v \rightarrow 0 \qquad 0 + v \rightarrow v$$
$$(u+1) * v \rightarrow u * v + v$$

Additional hypotheses:

$$v = v$$
$$u \leq v \vee v < u$$

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. $0 \le i$ 4. $0 < j$			
trans 2 $0 * v \to 0$ $[q \to 0, v \to j]$		5. $i = 0 + r \ \land \ 0 \le r < j$	0	r
trans 5 $0 + v \to v$ $[v \to r]$		6. $i = r \ \land \ 0 \le r < j$	0	r
resolve 6 & $v = v$ $[v \to i, r \to i]$		7. $0 \le i \land i < j$	0	i
resolve 7 & 3 []		8. $i < j$	0	i

Example: step case

	assertions	goals	outputs $div(i,j)$	 $rem(i,j)$
	3. $0 \leq i$ 4. $0 < j$	2. $i = q * j + r \;\; \land \;\; 0 \leq r < j$	q	r
trans 2 $(u+1) * v \to u * v + v$ $[q \to q' + 1, u \to q', v \to j]$		9. $i = q' * j + j + r \;\; \land \;\; 0 \leq r < j$	$q' + 1$	r
		10. $i - j = q' * j + r \;\; \land \;\; 0 \leq r < j$	$q' + 1$	r
induction	11. $(u,v) < (i,j) \Rightarrow$ $0 \leq u \land 0 < v \Rightarrow$ $u = div(u,v) * v + rem(u,v)$ $\land \;\; 0 \leq rem(u,v) < v$			
resolve 10 and 11 $[u \to i - j, v \to j,$ $q' \to div(i-j,j), r \to rem(i-j,j)]$		12. $(i-j,j) < (i,j) \land$ $0 \leq i - j \land 0 < j$	$div(i-j,j) + 1$	$rem(i-j,j)$
…		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 []		13. T	if $i < j$ then 0 else $div(i-j,j)+1$	if $i < j$ then i else $rem(i-j,j)$

Two approaches

Modern approach:
[Reynolds et al, CAV'15]

Theorem proving

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!

[Manna, Waldinger'80]

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

[Kneuss et al.'13]

The Leon synthesis framework

Deductive synthesis is good at figuring out high-level structure

Inductive synthesis is good at generating straight-line fragments

Idea: combine them!

- first decompose the synthesis problem using deductive rules
- then use inductive synthesizers as terminal rules

Synthesis problem

inputs outputs

$$[\![\bar{a}\ \langle \Pi \rhd \phi \rangle\ \bar{x}]\!]$$

path condition synthesis predicate

c.f. deductive synthesis problem

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

Synthesis judgment

Instead of transforming synthesis problems directly, Leon transforms synthesis judgments:

$$\underbrace{[\![\bar{a}\ \langle \Pi \rhd \phi \rangle\ \bar{x}]\!]}_{\text{synthesis problem}} \vdash \langle P, \bar{T} \rangle$$

program terms

precondition

Meaning

- relation refinement

$$\Pi \land P \vDash \phi[\bar{x} \mapsto \bar{T}]$$

- domain preservation

$$\Pi \land (\exists \bar{x}. \phi) \vDash P$$

Inference rules

one-point

$$\frac{x_0 \notin \mathrm{vars}(t) \qquad [\![\bar{a}\ \langle \Pi \rhd \phi[x_0 \mapsto t] \rangle\ \bar{x}]\!] \vdash \langle P, \bar{T} \rangle}{[\![\bar{a}\ \langle \Pi \rhd x_0 = t \land \phi \rangle\ x_0, \bar{x}]\!]}$$

case-split

$$\frac{[\![\bar{a}\ \langle \Pi \rhd \phi_1 \rangle\ \bar{x}]\!] \vdash \langle P_1, \bar{T}_1 \rangle \qquad [\![\bar{a}\ \langle \Pi \land \neg P_1 \rhd \phi_2 \rangle\ \bar{x}]\!] \vdash \langle P_2, \bar{T}_2 \rangle}{[\![\bar{a}\ \langle \Pi \rhd \phi_1 \lor \phi_2 \rangle\ \bar{x}]\!]}$$

list-rec

$$\frac{\begin{array}{c}\Pi[\mathrm{l} \mapsto h :: t] \Rightarrow \Pi[\mathrm{l} \mapsto t] \qquad [\![\bar{a}\ \langle \Pi[l \mapsto \emptyset] \rhd \phi[l \mapsto \emptyset] \rangle\ x]\!] \vdash \langle \top, T_1 \rangle \\ [\![h, t, r, \bar{a}\ \langle \Pi[\mathrm{l} \mapsto h :: t] \land \phi[l \mapsto t, x \mapsto r] \rhd \phi[l \mapsto h :: t] \rangle\ x]\!] \vdash \langle \top, T_2 \rangle\end{array}}{[\![l, \bar{a}\ \langle \Pi \rhd \phi \rangle\ x]\!]}$$

Complex terminal rules

Symbolic Term Exploration

- Essentially Sketch

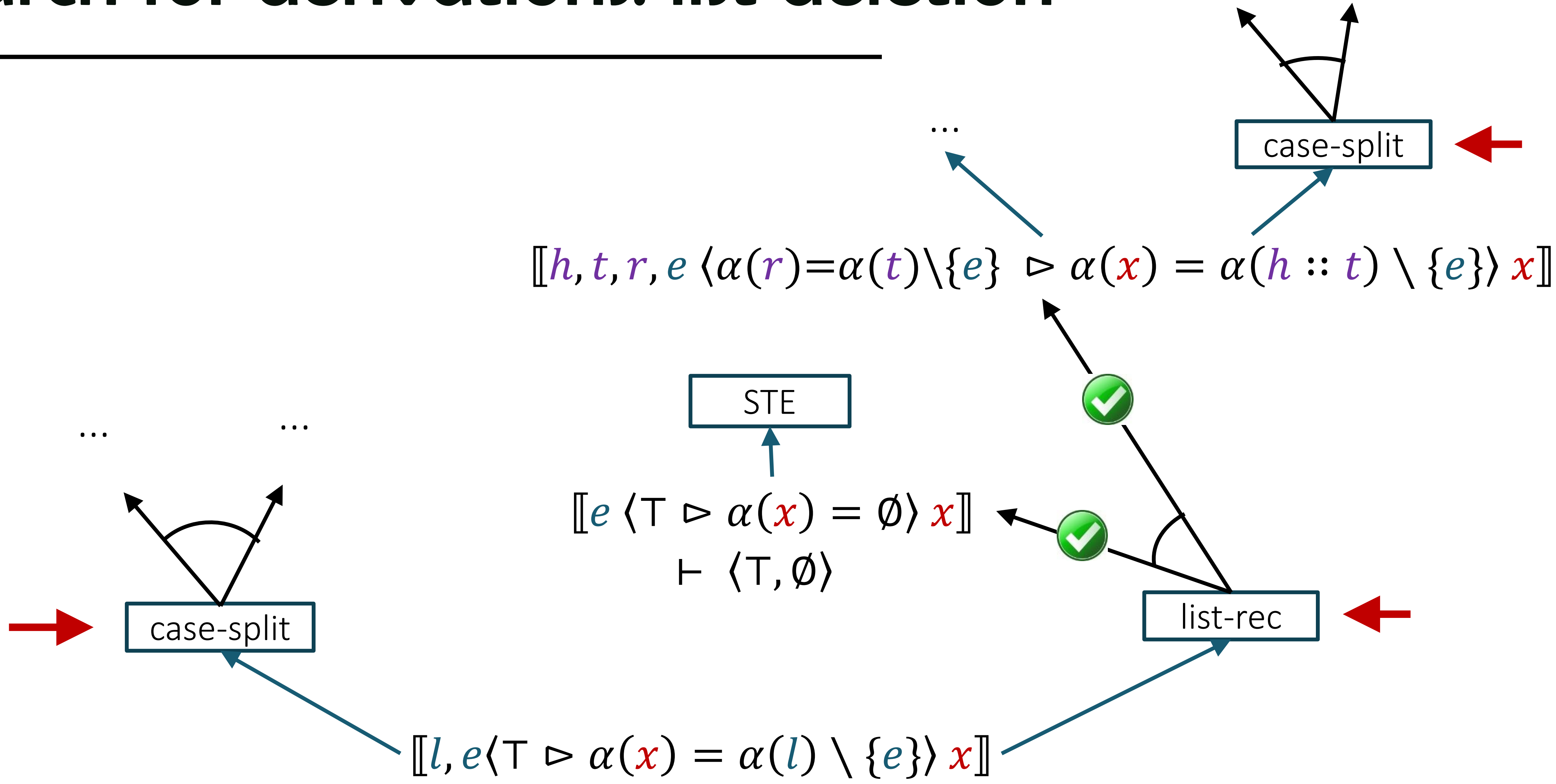
$$\frac{\mathrm{STE}(\Pi, \phi) = \bar{T}}{[\![\bar{a} \; \langle \Pi \rhd \phi \rangle \; \bar{x}]\!] \vdash \langle \top, \bar{T} \rangle}$$

Condition Abduction

- Essentially EUSolver

$$\frac{\mathrm{CA}(\Pi, \phi) = \bar{T}}{[\![\bar{a} \; \langle \Pi \rhd \phi \rangle \; \bar{x}]\!] \vdash \langle \top, \bar{T} \rangle}$$

Search for derivations: list deletion



Leon is…

A deductive synthesis framework

- with powerful terminal rules that use inductive synthesis
  - Symbolic Term Exploration
  - Condition Abduction
- cost-based search for derivations

A synthesis-aided language

- functional language + choose
- interaction model

Modern deductive synthesizers

Combine deductive synthesis with modern techniques

- automated reasoning
- inductive synthesis

Are still mostly interactive!

- search in the space of derivations is generally hard
- even a little user guidance goes a long way
- Examples: Fiat, Bellmania

Fiat

[Delaware et al. ‘15]

Input: a high-level spec, e.g. a database query

```
query NumOrders (author: string) : nat :=
  Count (For (o in Orders) (b in Books)
         Where (author = b!Author)
         Where (b!ISBN = o!ISBN)
         Return ())
```

Iteratively refined into efficient implementations via automated tactics

```
meth NumOrders (p: rep , author: string) : nat :=
  let (books, orders) := p in
  ret (books, orders,
       fold_left
       (\ count tup .
         count + bcount orders (Some tup!ISBN, []))
        (bnd books (Some author, None, [])) 0)
```

Bellmania

[Itzhaky et al. '16]

Deriving parallel divide-and-conquer implementations of dynamic programming algorithms

- start from a naive implementation
- at each step, the user picks a tactic to transform the program
- tool checks side-conditions
    - i.e. that the transformation produces an equivalent program
    - but has no idea where the whole thing is going