CS 4110
Programming Languages & Logics

Lecture 36
Program Synthesis

The Dream

What if I told you that I could synthesize a program from a prose description, or input–output examples?

The Dream

What if I told you that I could synthesize a program from a prose description, or input–output examples?

- A few years ago, this would have sounded like science fiction.

The Dream

What if I told you that I could synthesize a program from a prose description, or input–output examples?

- A few years ago, this would have sounded like science fiction.
- But with LLMs (ChatGPT, Cursor, etc.) it has become totally common!

The Dream

What if I told you that I could synthesize a program from a prose description, or input–output examples?

- A few years ago, this would have sounded like science fiction.
- But with LLMs (ChatGPT, Cursor, etc.) it has become totally common!
- How do we know a synthesized program is correct?

Stanford Artificial Intelligence Laboratory
Memo AIM-302

November 1977

Computer Science Department
Report No. STAN-CS-77-630

12

SYNTHESIS: DREAMS => PROGRAMS

by

Zohar Manna
Artificial Intelligence Lab
Stanford University
Stanford, CA.

Richard Waldinger
Artificial Intelligence Center
SRI International
Menlo Park, CA.

See back page for 1473

Research sponsored by

National Science Foundation
Office of Naval Research
Advanced Research Projects Agency

COMPUTER SCIENCE DEPARTMENT
Stanford University

Program Synthesis

Goal: Generate a program satisfying constraints

Program Synthesis

Goal: Generate a program satisfying constraints

Constraints

- Logical specifications
- Partial programs (“sketches”)
- Input–output examples

Program Synthesis

Goal: Generate a program satisfying constraints

Constraints

- Logical specifications
- Partial programs (“sketches”)
- Input–output examples

Techniques

- Deductive synthesis
- Inductive synthesis

Program Synthesis

Goal: Generate a program satisfying constraints

Constraints

- Logical specifications
- Partial programs (“sketches”)
- Input–output examples

Techniques

- Deductive synthesis
- Inductive synthesis
  - Enumeration (top-down/bottom-up)
  - CEGIS
  - Sketching

Inductive Synthesis

Goal: Generate a program satisfying constraints

Inductive Synthesis

Goal: Generate a program satisfying constraints

Key Challenge: search space is *huge*.

Inductive Synthesis

Goal: Generate a program satisfying constraints

Key Challenge: search space is *huge*.

Common Approaches

- Limit space of programs
- Harness symbolic representations
- Use SAT/SMT solvers
- Rely on domain heuristics

Example: Synthesizing Regular Expressions

Definition (Problem Statement)

Given positive (P) and negative (N) examples, find R s.t. $P \subseteq [\![R]\!]$ and $N \cap [\![R]\!] = \emptyset$.

Example: Synthesizing Regular Expressions

Definition (Problem Statement)

Given positive (P) and negative (N) examples, find R s.t. $P \subseteq [\![R]\!]$ and $N \cap [\![R]\!] = \emptyset$.

Examples

```
01,+
X01,+
XX01,+
X0,-
X11,-
```

Example: Synthesizing Regular Expressions

Definition (Problem Statement)

Given positive (P) and negative (N) examples, find R s.t. $P \subseteq [\![R]\!]$ and $N \cap [\![R]\!] = \emptyset$.

Examples

```
01,+
X01,+
XX01,+
X0,-
X11,-
```

$\Longrightarrow$

Solution

```
(0 + 1)* . 0 . 1
```

Defining the Search Space

Definition (Partial Regular Expressions)

R	$::=$	$\square$	*Hole*
	$\mid$	$\emptyset$	*Empty Set*
	$\mid$	ϵ	*Empty String*
	$\mid$	c	*Character*
	$\mid$	$R_1 + R_2$	*Union*
	$\mid$	$R_1 \cdot R_2$	*Concatenation*
	$\mid$	R^*	*Kleene Star*

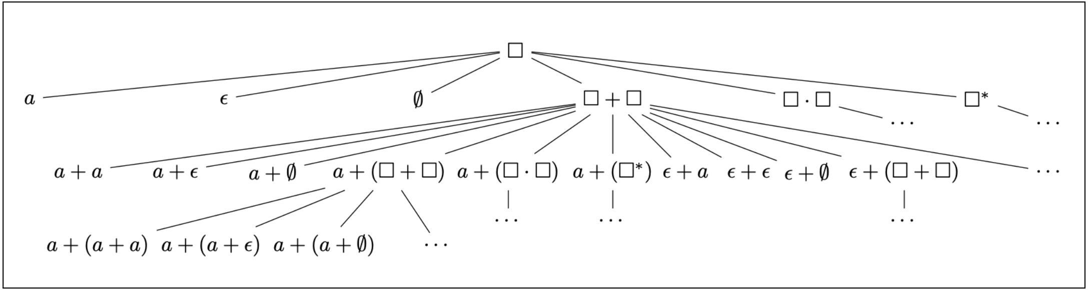
Defining the Search Space

Definition (Expansion Rules)

$$\begin{aligned}
\square &\rightarrow \emptyset \\
\square &\rightarrow \epsilon \\
\square &\rightarrow c \\
\square &\rightarrow \square + \square \\
\square &\rightarrow \square \cdot \square \\
\square &\rightarrow \square *
\end{aligned}$$

Plus the “obvious” congruence rules for $+$, $\cdot$, and *.

Visualizing the search space



https://cs.stanford.edu/~minalee/pdf/gpce2016-alpharegex.pdf

Synthesis Algorithm

Definition (Naive Synthesis)

```
synthesize(P, N):
  W := { □ }
  repeat
    pick R from W
    if solution(R, P, N) then
      return R
    else
      W := W ∪ { R' | R → R' }
  until W = {}
```

Heuristic Optimizations

Prioritize small states

- Try smaller programs before bigger ones.

Heuristic Optimizations

Prioritize small states

- Try smaller programs before bigger ones.

Recognize equivalent states

- Normalize regexes using rewrite rules

Heuristic Optimizations

Prioritize small states

- Try smaller programs before bigger ones.

Recognize equivalent states

- Normalize regexes using rewrite rules

Prune dead states

- Fill holes and over/under approximate

Heuristic Optimizations

Prioritize small states

- Try smaller programs before bigger ones.

Recognize equivalent states

- Normalize regexes using rewrite rules

Prune dead states

- Fill holes and over/under approximate

Avoid redundancy

- Skip states irrelevant for covering positive examples.

Cost function

Prioritize small states: Try smaller programs before bigger ones.

Definition (Cost)

$$
\begin{aligned}
\mathcal{C}(\square) &\triangleq c_1 \\
\mathcal{C}(\emptyset) &\triangleq c_1 \\
\mathcal{C}(\epsilon) &\triangleq c_1 \\
\mathcal{C}(c) &\triangleq c_1 \\
\mathcal{C}(R_1 + R_2) &\triangleq \mathcal{C}(R_1) + \mathcal{C}(R_2) + c_2 \\
\mathcal{C}(R_1 \cdot R_2) &\triangleq \mathcal{C}(R_1) + \mathcal{C}(R_2) + c_3 \\
\mathcal{C}(R*) &\triangleq \mathcal{C}(R) + c_4
\end{aligned}
$$

Cost function

Prioritize small states: Try smaller programs before bigger ones.

Definition (Cost)

$$
\begin{aligned}
\mathcal{C}(\square) &\triangleq c_1 \\
\mathcal{C}(\emptyset) &\triangleq c_1 \\
\mathcal{C}(\epsilon) &\triangleq c_1 \\
\mathcal{C}(c) &\triangleq c_1 \\
\mathcal{C}(R_1 + R_2) &\triangleq \mathcal{C}(R_1) + \mathcal{C}(R_2) + c_2 \\
\mathcal{C}(R_1 \cdot R_2) &\triangleq \mathcal{C}(R_1) + \mathcal{C}(R_2) + c_3 \\
\mathcal{C}(R*) &\triangleq \mathcal{C}(R) + c_4
\end{aligned}
$$

Typically, pick $c_2 > c_3$ so that concatenation is preferred over union.

Equivalence

Recognize equivalent states: Normalize regexes using rewrite rules

Definition (Regex Equivalences)

$$\emptyset \cdot R \equiv \emptyset \equiv R \cdot \emptyset$$
$$\epsilon \cdot R \equiv R \equiv R \cdot \epsilon$$
$$(R_1 \cdot R_2) \cdot R_3 \equiv R_1 \cdot (R_2 \cdot R_3)$$
$$(R_1 + R_2) + R_3 \equiv R_1 + (R_2 + R_3)$$
$$R_1 + R_2 \equiv R_2 + R_1$$
$$R_1 \cdot R_2 + R_1 \cdot R_3 \equiv R_1 \cdot (R_2 + R_3)$$
$$R_1 \cdot R_2 + R_3 \cdot R_2 \equiv (R_1 + R_3) \cdot R_2$$
$$R \cdot R* \equiv R*$$
$$\emptyset* \equiv \epsilon \equiv \epsilon*$$

Pruning

Prune dead states: Fill holes and over/under approximate

Definition (Over/Under Approximations)

$$\widehat{\square} \triangleq (c_1 + \ldots c_n)*$$
$$\widetilde{\square} \triangleq \emptyset$$

(And homomorphically for other constructs...)

Pruning

Prune dead states: Fill holes and over/under approximate

Definition (Over/Under Approximations)

$$\widehat{\square} \triangleq (c_1 + \ldots c_n)*$$

$$\widetilde{\square} \triangleq \emptyset$$

(And homomorphically for other constructs...)

Definition (Dead State)

$$dead(R) \triangleq \exists p \in P.\ p \notin [\![\widehat{R}]\!] \vee \exists n \in N.\ n \in [\![\widetilde{R}]\!]$$

Redundancy

Avoid redundancy: Skip states irrelevant for covering positive examples.

Example

Suppose $0 \notin P$. Then $0 + \square$ is redundant as $\square$ suffices.

Redundancy

Avoid redundancy: Skip states irrelevant for covering positive examples.

Example

Suppose $0 \notin P$. Then $0 + \square$ is redundant as $\square$ suffices.

Example

Suppose $P \triangleq \{0, 01, 011, 0111\}$. Then $0 * \cdot \square$ is redundant as $\square$ suffices.

Redundancy

Avoid redundancy: Skip states irrelevant for covering positive examples.

Example

Suppose $0 \notin P$. Then $0 + \square$ is redundant as $\square$ suffices.

Example

Suppose $P \triangleq \{0, 01, 011, 0111\}$. Then $0 * \cdot \square$ is redundant as $\square$ suffices.

The details for recognizing redundant states are slightly involved; see the original paper on AlphaRegex (GPCE '16) for full details.

Demo

Properties

Definition (Soundness)

If $R \in \texttt{synthesize(N,P)}$ then $P \subseteq [\![R]\!]$ and $N \cap [\![R]\!] = \emptyset$.

Definition (Completeness)

If $\texttt{synthesize}(N, P)$ fails then $\nexists R$ such that $P \subseteq [\![R]\!]$ and $N \cap [\![R]\!] = \emptyset$.

Note that synthesis can never fail, so completeness holds trivially...

Counterexample-Guided Inductive Synthesis

Definition (CEGIS)

```
cegis(spec):
  cexs := ∅
  while true:
    candidate := generate(spec, cexs)
    if verify(spec, candidate):
      return candidate
    else:
      obtain counterexample e
      cexs := cexs ∪ { e }
```

More Approaches

Bottom-up Enumeration

- Generate candidate programs from small to large
- Similar to dynamic programming

More Approaches

Bottom-up Enumeration

- Generate candidate programs from small to large
- Similar to dynamic programming

Symbolic Representations

- Version Space Algebras
- E-graphs

More Approaches

Bottom-up Enumeration

- Generate candidate programs from small to large
- Similar to dynamic programming

Symbolic Representations

- Version Space Algebras
- E-graphs

SAT/SMT-Based Synthesis

- Encode constraints as logical formulas
- Good for finding tricky constants

More Approaches

Bottom-up Enumeration

- Generate candidate programs from small to large
- Similar to dynamic programming

Symbolic Representations

- Version Space Algebras
- E-graphs

SAT/SMT-Based Synthesis

- Encode constraints as logical formulas
- Good for finding tricky constants

Stochastic Search

- Based on probabilistic algorithms
- Markov Chain Monte Carlo

Even More Approaches

Type-Guided Synthesis

- Prune ill-typed programs
- Use refinement types to capture semantic constraints

Even More Approaches

Type-Guided Synthesis

- Prune ill-typed programs
- Use refinement types to capture semantic constraints

Component Discovery

- Meta-learning to generate a DSL
- Uses “wake-sleep” algorithm

Even More Approaches

Type-Guided Synthesis

- Prune ill-typed programs
- Use refinement types to capture semantic constraints

Component Discovery

- Meta-learning to generate a DSL
- Uses “wake-sleep” algorithm

LLM-Based Synthesis

- Rejection sampling
- Constrained decoding