COSE212: Programming Languages

Lecture 15 — Program Synthesis

Hakjoo Oh
2016 Fall

Program Synthesis

Technology for automatically generating programs from specifications.

- Specifications: logic, examples, natural language, programs, etc
- Programs: general-purpose, domain-specific, etc

Applications:

- Programming assistance: programs write programs
- End-user programming: automate repetitive tasks
- Algorithm discovery: find a new solution or insight
- Patch generation: automatically fix software bugs
- Program optimization: find a more efficient implementation

Example 1

Q. Find a regular expression for the following language:

$$L = \{w \in \{0,1\}^* \mid w \text{ has exactly one pair of consecutive 0s}\}$$

- Positive examples: 00, 1001, 010010, 1011001110, ...
- Negative examples: 01, 11, 000, 00100, ...

Regular Expression Synthesizer[1]

Positive Examples

00,
1001,
010010,
1011001110

Negative Examples

01,
11,
000,
00100

$\Longrightarrow$ RE Synthesizer $\Longrightarrow$ $(0?1)^*00(10?)^*$

[1] `http://prl.korea.ac.kr/AlphaRegex`

Example 2

Q. Complete the following program that reverses integers:

```
int reverse (int n) {
  int r = 0;
  while (n > 0) {
    ?;
  }
  return r;
}
```

For example,

- $\mathbf{1 \Rightarrow 1}$
- $\mathbf{12 \Rightarrow 21}$
- $\mathbf{123 \Rightarrow 321}$

Given a partial program (sketch) and input-output examples, a program synthesizer completes the program.

Today: Automatic Synthesis of Regular Expressions

Regular expressions:

- Syntax:

$$e \rightarrow a \in \Sigma \mid \epsilon \mid \emptyset \mid e_1 + e_2 \mid e_1 \cdot e_2 \mid e^*$$

- Semantics:

$$\begin{aligned}
[\![a]\!] &= \{a\}\ (a \in \Sigma) \\
[\![\epsilon]\!] &= \{\epsilon\} \\
[\![\emptyset]\!] &= \emptyset \\
[\![e_1 + e_2]\!] &= [\![e_1]\!] \cup [\![e_2]\!] \\
[\![e_1 \cdot e_2]\!] &= [\![e_1]\!][\![e_2]\!] \\
[\![e^*]\!] &= [\![e]\!]^*
\end{aligned}$$

Synthesis Problem

Given $\mathcal{P} \subseteq \Sigma^*$ of *positive examples* and $\mathcal{N} \subseteq \Sigma^*$ of *negative examples*, find a regular expression e that accepts all the positive examples while rejecting all the negative examples:

$$\forall p \in \mathcal{P}.p \in [\![e]\!] \wedge \forall n \in \mathcal{N}.n \notin [\![e]\!].$$

Synthesis Algorithm

Reduction to a search problem:

$$(S, \rightarrow, I, F)$$

- **States**: Partial regular expressions possibly with holes ($\square$):

$$s \rightarrow a \in \Sigma \mid \epsilon \mid \emptyset \mid s_1 + s_2 \mid s_1 \cdot s_2 \mid s^* \mid \square$$

- **Initial State**: $I = \square$.
- **Transition Relation**: $(\rightarrow) \subseteq S \times S$ determines the next states. The set of all states that follow s:

$$\textbf{next}(s) = \{s' \mid s \rightarrow s'\}.$$

- **Solution States**: $F = \{s \mid \textbf{solution}(s)\}$

$$\textbf{solution}(s) \iff s \not\rightarrow \ \wedge \forall p \in \mathcal{P}.p \in [\![s]\!] \wedge \forall n \in \mathcal{N}.n \notin [\![s]\!].$$

Synthesis Algorithm

Transition relation:

$$\frac{s_1 \to s_1'}{s_1 + s_2 \to s_1' + s_2} \qquad \frac{s_2 \to s_2'}{s_1 + s_2 \to s_1 + s_2'}$$

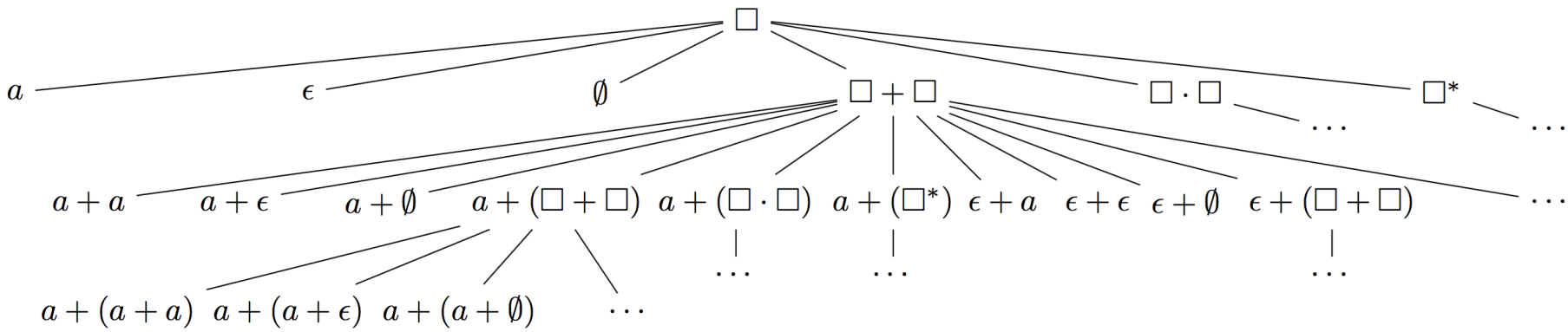
$$\frac{s_1 \to s_1'}{s_1 \cdot s_2 \to s_1' \cdot s_2} \qquad \frac{s_2 \to s_2'}{s_1 \cdot s_2 \to s_1 \cdot s_2'}$$

$$\frac{s \to s'}{s^* \to s'^*}$$

$$\frac{}{\square \to a}\ a \in \Sigma \qquad \frac{}{\square \to \epsilon} \qquad \frac{}{\square \to \emptyset}$$

$$\frac{}{\square \to \square + \square} \qquad \frac{}{\square \to \square \cdot \square} \qquad \frac{}{\square \to \square^*}$$

Synthesis Algorithm



Synthesis Algorithm

Naive search algorithm:

$W := \{\square\}$
repeat
  Pick a state s from W
  if **solution**(s) then return s
  else
    $W := W \cup$ **next**(s)
until $W \neq \emptyset$

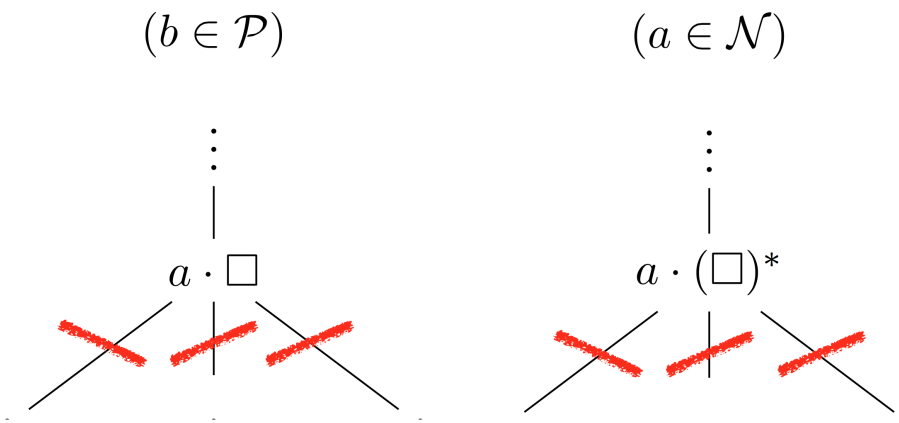
Challenge: extremely large search space!

$$N(d) = 7^{2^d} - 1$$

Pruning the Search Space

Techniques for safely pruning the search space:

- pruning semantically-equivalent states,
- pruning states that are guaranteed not to be solutions, and
- pruning redundant states.



More details can be found at:

- Mina Lee, Sunbeom So, and Hakjoo Oh.
  Synthesizing Regular Expressions from Examples for Introductory Automata Assignments. GPCE 2016.

Performance

Level	No	Description	Examples		Full	No Apr	No Rd	Output
			Pos	Neg	sec	sec	sec	
Easy	1	Start with 0.	3	3	0.0	0.0	0.0	0(0+1)*
	2	End with 01.	3	6	0.0	0.1	0.0	(0+1)*01
	3	Contain the substring 0101.	3	7	1.4	10.6	1.9	(0+1)*0101(0+1)*
	4	Begin with 1 and end with 0.	3	5	0.0	0.0	0.0	1(0+1)*0
	5	Length is at least 3 and the 3rd symbol is 0.	3	4	0.0	0.0	0.0	(0+1)(0+1)0(0+1)*
	6	Length is a multiple of 3.	2	3	0.1	0.1	0.1	((0+1)(0+1)(0+1))*
Normal	7	The number of 0s is divisible by 3.	8	7	9.5	238.4	36.9	(1+01*01*0)*
	8	Even number of 0s.	7	7	0.1	1.0	0.2	(1+01*0)*
	9	The 5th symbol from the right end is 1.	3	3	9.0	56.3	14.3	(0+1)*1(0+1)(0+1)(0+1)(0+1)
	10	0 and 1 alternate.	9	8	1.7	19.7	3.5	1?(01)*0?
	11	Each 0 is followed by at least one 1.	7	6	0.0	0.0	0.0	(0?1)*
	12	0^n1^m where $n \geq 3$ and m is even.	6	5	0.2	1.0	0.7	000*0(11)*
	13	Have at most two 0s.	8	7	1.4	9.5	2.3	1*0?1*0?1*
	14	Start with 0 and have odd length or start with 1 and have even length.	5	5	16.5	771.9	14.7	(0+1(0+1))((0+1)(0+1))*
	15	Any strings except 0 and 1.	3	2	0.0	0.0	0.0	(0+1)(0+1)(0+1)*
	16	Do not end with 01.	8	3	17.8	302.1	15.2	0+1+(0+1)*(0+1(0+1))
Hard	17	Contain at least one 0 and at most one 1.	6	9	2.7	33.5	7.7	0*(01?+100*)
	18	At least two occurrences of 1 between any two occurrences of 0.	7	7	0.2	0.4	0.2	0?(1(10)?)*
	19	Do not contain 100 as a substring.	8	4	0.1	0.1	0.1	0*(1(0+1)?)*
	20	Every odd position is 1.	6	9	3.6	10.5	5.0	(1(0+1))*1*
	21	Have exactly one pair of consecutive 0s.	4	4	1.0	16.1	2.9	(0?1)*00(10?)*
	22	Do not end with 10 and have a length of at least two	9	4	57.2	4248.0	75.3	(0+1)(0+1)*1+(0+1)*0(0+1)
	23	Even number of 0s and each 0 is followed by at least one 1.	6	9	1.8	30.2	4.3	(1+01*101)*
	24	Every pair of adjacent 0s appears before any pair of adjacent 1s.	9	9	14.4	506.1	73.7	0?((01)?10?)*
	25	At most one pair of consecutive 1s.	5	5	28.8	863.6	211.2	(1?0)*1?1?(01?)*
Average					6.7	284.8	18.8	

Summary

- Program synthesis is an active research area, which enables computers to write programs.
- The task of synthesizing a program is reduced to a search problem over the space of programs (i.e., programming language).