CS406: Compilers

Spring 2021

Week 4: Parsers

Parsing – so far..

- Parsing involves:
  - identifying if a program has *syntax errors*
  - Identifying the *structure* of a valid program
- CFGs are formal notations for specifying the rules of the programming language
  - Has *symbols* (start, terminal(s), non-terminal(s)), and *productions/rules*
  - *Derivations* are a sequence of expansions of a string of symbols

  Left-most derivation and *Right-most derivation* are popular methods defining the order in the sequence

Parsing – so far..

- Parse trees are tree structures having terminals as leaves and non-terminals as nodes
  - The sequence involved in derivations define them
  - For a given string having *terminal symbols only,* there exists *only one parse tree* in an *unambiguous grammar*
    - A grammar is ambiguous if there exists some string for which different derivations result in more than one tree structure
- Ambiguity fixing in grammars
  - Manual rewriting of grammar
  - Hints to parser generators
- Error handling in parsers
  - Panic mode, error productions, and error recovery.

3

Top-down Parsing

- Idea: we know sentence has to start with initial symbol
- Build up partial derivations by *predicting* what rules are used to expand non-terminals
  - Often called *predictive parsers*
- If partial derivation has terminal characters, *match* them from the input stream

4

Top-down Parsing

- Also called recursive-descent parsing
- Equivalent to finding the left-derivation for an input string
  - Recall: expand the leftmost non-terminal in a parse tree
  - Expand the parse tree in pre-order i.e., identify parent nodes before children

Top-down Parsing

↑: next symbol to be read

```
1: S -> cAd
2: A -> ab
3:    | a
```

Step	Input string	Parse tree
1	cad ↑ (at c)	S

String: **cad**

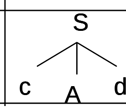
Start with S

6

Top-down Parsing

↑: next symbol to be read

```
1: S -> cAd
2: A -> ab
3:    | a
```

Step	Input string	Parse tree
1	cad	S
2	cad	

String: **cad**

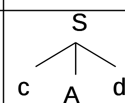
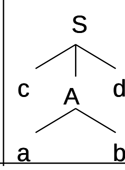
Predict rule 1

Top-down Parsing

↑: next symbol to be read

```
1: S -> cAd
2: A -> ab
3:      | a
```

String: **cad**

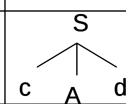
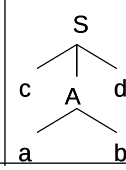
Step	Input string	Parse tree
1	cad ↑	S
2	cad ↑	
3	cad ↑	

Predict rule 2

Top-down Parsing

↑: next symbol to be read

```
1: S -> cAd
2: A -> ab
3:      | a
```

Step	Input string	Parse tree
1	cad	S
2	cad	
3	cad	

String: **cad**

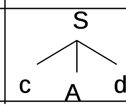
No more non terminals!
String doesn't match.
Backtrack.

9

Top-down Parsing

↑: next symbol to be read

```
1: S -> cAd
2: A -> ab
3:      | a
```

Step	Input string	Parse tree
1	cad ↑	S
2	cad ↑	

String: **cad**

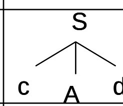
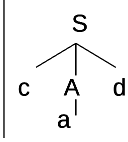
10

Top-down Parsing

↑: next symbol to be read

```
1: S -> cAd
2: A -> ab
3:      | a
```

String: **cad**

Step	Input string	Parse tree
1	cad	S
2	cad	
4	cad	

Predict rule 3

11

Top-down Parsing – Table-driven Approach

```
1: S -> F
2: S -> (S + F)
3: F -> a
```

string: `(a+a)`

string': `(a+a)$`

	(	)	a	+	$
S	2	-	1	-	-
F	-	-	3	-	-

Assume that the table is given.

- Table-driven (Parse Table) approach doesn't require backtracking

But how do we construct such a table?

Important Concepts: First Sets and Follow Sets

13

Concepts for analyzing the grammar

First and follow sets

- First(α): the set of terminals (and/or λ) that begin all strings that can be derived from α
  - First(A) = {x, y, λ}
  - First(xaA) = {x}
  - First (AB) = {x, y, b}
- Follow(A): the set of terminals (and/or $, but no λs) that can appear immediately after A in some partial derivation
  - Follow(A) = {b}

S → A B $

A → x a A

A → y a A

A → λ

B → b

First and follow sets

- $\text{First}(\alpha) = \{a \in V_t \mid \alpha \Rightarrow^* a\beta\} \cup \{\lambda \mid \text{if } \alpha \Rightarrow^* \lambda\}$
- $\text{Follow}(A) = \{a \in V_t \mid S \Rightarrow^+ \ldots Aa \ldots\} \cup \{\$ \mid \text{if } S \Rightarrow^+ \ldots A\ \$\}$

S:	start symbol
a:	a terminal symbol
A:	a non-terminal symbol
α,β:	a string composed of terminals and non-terminals (typically, α is the RHS of a production

$\Rightarrow$:	derived in 1 step
$\Rightarrow^*$:	derived in 0 or more steps
$\Rightarrow^+$:	derived in 1 or more steps

15

Computing first sets

- Terminal: First(a) = {a}
- Non-terminal: First(A)
  - Look at all productions for A

    $A \rightarrow X_1X_2 \ldots X_k$
  - First(A) $\supseteq$ (First(X_1) - λ)
  - If $\lambda \in$ First(X_1), First(A) $\supseteq$ (First(X_2) - λ)
  - If λ is in First(X_i) for all i, then $\lambda \in$ First(A)
- Computing First(α): similar procedure to computing First(A)

A simple example

$S \rightarrow A\ B\ c\ \$$

$A \rightarrow x\ a\ A$

$A \rightarrow y\ a\ A$

$A \rightarrow c$

$B \rightarrow b$

$B \rightarrow \lambda$

- A sentence in the grammar:

  x a c c $

A simple example

S → A B c $

special "end of input" symbol

A → x a A

A → y a A

A → c

B → b

B → λ

- A sentence in the grammar:

  x a c c $

18

A simple example

S → A B c $

A → x a A

A → y a A

A → c

B → b

B → λ

- A sentence in the grammar:

  x a c c $

Current derivation: S

A simple example

S → A B c $

A → x a A

A → y a A

A → c

B → b

B → λ

- A sentence in the grammar:

  x a c c $

Current derivation: A B c $

Predict rule

20

A simple example

$S \rightarrow A\ B\ c\ \$$

Choose based on *first set* of rules

$A \rightarrow x\ a\ A$

$A \rightarrow y\ a\ A$

$A \rightarrow c$

$B \rightarrow b$

$B \rightarrow \lambda$

- A sentence in the grammar:

  x a c c $

Current derivation: x a A B c $

Predict rule *based on next token*

A simple example

$S \rightarrow A\ B\ c\ \$$

$A \rightarrow x\ a\ A$

$A \rightarrow y\ a\ A$

$A \rightarrow c$

$B \rightarrow b$

$B \rightarrow \lambda$

- A sentence in the grammar:

  x a c c $

Current derivation: x a A B c $

Match token

A simple example

$S \rightarrow A\ B\ c\ \$$

$A \rightarrow x\ a\ A$

$A \rightarrow y\ a\ A$

$A \rightarrow c$

$B \rightarrow b$

$B \rightarrow \lambda$

- A sentence in the grammar:

  x a c c $

Current derivation: x a A B c $

Match token

A simple example

S → A B c $

Choose based on *first set* of rules

A → x a A

A → y a A

A → c

B → b

B → λ

- A sentence in the grammar:

  x a c c $

Current derivation: x a c B c $

Predict rule *based on next token*

A simple example

S → A B c $

A → x a A

A → y a A

A → c

B → b

B → λ

- A sentence in the grammar:

  x a c c $

Current derivation: x a c B c $

Match token

25

A simple example

S → A B c $

A → x a A

Choose based on *follow set*

A → y a A

A → c

B → b

B → λ

- A sentence in the grammar:

  x a c c $

Current derivation: x a c λ c $

Predict rule *based on next token*

26

A simple example

S → A B c $

A → x a A

A → y a A

A → c

B → b

B → λ

- A sentence in the grammar:

  x a c c $

Current derivation: x a c c $

Match token

A simple example

$S \rightarrow A\ B\ c\ \$$

$A \rightarrow x\ a\ A$

$A \rightarrow y\ a\ A$

$A \rightarrow c$

$B \rightarrow b$

$B \rightarrow \lambda$

- A sentence in the grammar:

  x a c c $

Current derivation: x a c c $

Match token

28

Top-down Parsing

```
1) S -> F
2) S -> (S + F)
3) F -> a
```

string: `(a+a)`

string': `(a+a)$`

	(	)	a	+	$
S	2	-	1	-	-
F	-	-	3	-	-

Assume that the table is given.

- Table-driven (Parse Table) approach doesn't require backtracking

But how do we construct such a table?

First and follow sets

- First(α): the set of terminals (and/or λ) that begin all strings that can be derived from α
  - First(A) = {x, y, λ}
  - First(xaA) = {x}
  - First (AB) = {x, y, b}
- Follow(A): the set of terminals (and/or \$, but no λs) that can appear immediately after A in some partial derivation
  - Follow(A) = {b}

$S \rightarrow A\ B\ \$$

$A \rightarrow x\ a\ A$

$A \rightarrow y\ a\ A$

$A \rightarrow \lambda$

$B \rightarrow b$

30

First and follow sets

- $\text{First}(\alpha) = \{a \in V_t \mid \alpha \Rightarrow^* a\beta\} \cup \{\lambda \mid \text{if } \alpha \Rightarrow^* \lambda\}$
- $\text{Follow}(A) = \{a \in V_t \mid S \Rightarrow^+ \ldots Aa \ldots\} \cup \{\$ \mid \text{if } S \Rightarrow^+ \ldots A\ \$\}$

S: start symbol
a: a terminal symbol
A: a non-terminal symbol
α,β: a string composed of terminals and non-terminals (typically, α is the RHS of a production

$\Rightarrow$: derived in 1 step
$\Rightarrow^*$: derived in 0 or more steps
$\Rightarrow^+$: derived in 1 or more steps

Towards parser generators

- Key problem: as we read the source program, we need to decide what productions to use
- Step 1: find the tokens that can tell which production P (of the form $A \rightarrow X_1X_2 \ldots X_m$) applies

$$\text{Predict}(P) =$$

$$\begin{cases} \text{First}(X_1 \ldots X_m) & \text{if } \lambda \notin \text{First}(X_1 \ldots X_m) \\ (\text{First}(X_1 \ldots X_m) - \lambda) \cup \text{Follow}(A) & \text{otherwise} \end{cases}$$

- If next token is in Predict(P), then we should choose this production

32