

CS131 Compilers: Writing Assignment 2
Due Sunday, April 15, 2018 at 23:29

Zhiqiang Xie - 77892769

This assignment asks you to prepare written answers to questions on context-free grammars and parsing. Each of the questions has a short answer. You may discuss this assignment with other students and work on the problems together. However, your write-up should be your own individual work. and you should indicate in your submission who you worked with, if applicable. You should use the Latex template provided at the course web site to write your solution.

I worked with: (Li Dinghong, 27663262), (Name,ID)...

1. $2 \times 3 = 6$ **pts** Give context-free grammar (CFG) for each of the following languages:

- (a) The set of all finite strings over the alphabet $\{0, 1\}$ with an equal number of 0's and 1's.

$$S \rightarrow 0 S 1 S \mid 1 S 0 S \mid \epsilon$$

- (b) The set of all finite strings over the alphabet $\{0, 1\}$ with the number of 0's greater than the number of 1's.

$$\begin{aligned} S &\rightarrow \text{equal } 0 \text{ equal} \mid \text{equal } 0 S \\ \text{equal} &\rightarrow 0 \text{ equal } 1 \text{ equal} \mid 1 \text{ equal } 0 \text{ equal} \mid \epsilon \end{aligned}$$

- (c) The set $L_3 = L_1 \cap L_2$, where L_1 and L_2 are defined below. Let L_1 be the finite strings consisting of all non-empty *palindromes* over the alphabet $\{a, b\}$. That is L_1 consists of all sequences of a's and b's that read the same forward or backward. For example, $abba$, $aabbbbaa \in L_1$, but $abb \notin L_1$. Let L_2 be the language over $\{a, b\}$ representing the language of the regular expression $b(a + b)^*$.

$$\begin{aligned} S &\rightarrow b P b \\ P &\rightarrow a P a \mid b P b \mid \epsilon \end{aligned}$$

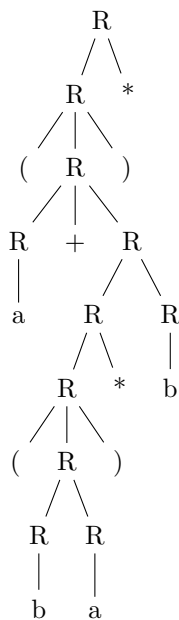
2. $3 \times 2 = 6$ **pts** Consider the following CFG with terminals $\{ (,), +, *, a, b \}$ (+ represents union) that is used to represent regular expressions over alphabet $\{a, b\}$:

$$R \rightarrow R + R \mid RR \mid (R) \mid R^* \mid a \mid b$$

- (a) Using the above CFG, provide a right-most derivation for the following input string $(a + (ba)^*b)^*$.

$$\begin{aligned} R &\Rightarrow R^* \Rightarrow (R)^* \\ &\Rightarrow (R + R)^* \\ &\Rightarrow (R + RR)^* \Rightarrow (R + Rb)^* \Rightarrow (R + R^*b)^* \Rightarrow (R + (R)^*b)^* \\ &\Rightarrow (R + (RR)^*b)^* \Rightarrow (R + (Ra)^*b)^* \Rightarrow (R + (ba)^*b)^* \Rightarrow (a + (ba)^*b)^* \end{aligned}$$

- (b) For the derivation in above solution, provide the corresponding parse tree.



3. $3 \times 3 = 9$ pts Consider the following CFG.

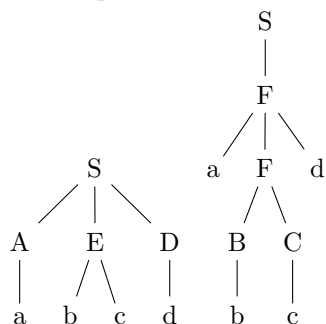
$$\begin{aligned} S &\rightarrow AED \mid F \\ A &\rightarrow Aa \mid a \\ B &\rightarrow Bb \mid b \\ C &\rightarrow Cc \mid c \\ D &\rightarrow Dd \mid d \\ E &\rightarrow bEc \mid bc \\ F &\rightarrow aFd \mid BC \end{aligned}$$

(a) What is the language generated by this grammar?

$$L = \{a^i b^j c^j d^k, a^i b^j c^k d^i \mid i, j, k \geq 1\}$$

(b) Is the grammar as given ambiguous? If yes, give an example of an expression with two parse trees under this grammar. If not, explain why that is the case.

It's ambiguous, because a string can both have the two features for parsing, which are the number of b is equal to the number of c , the number of a is equal to d . For example $abcd$:



(c) Transform the CFG given above by eliminating ambiguity and left recursion, if needed.

The main idea is to make AED and F disjoint:

$$\begin{aligned} S &\rightarrow AED \mid F \\ A &\rightarrow a \mid aA \\ B &\rightarrow b \mid bB \\ C &\rightarrow c \mid cC \\ D &\rightarrow d \mid dD \\ E &\rightarrow bEc \mid bc \\ F &\rightarrow aFd \mid BE \mid EC \end{aligned}$$

4. $3 \times 3 = 9$ pts Consider the following CFG.

$$\begin{aligned} A &\rightarrow [AB] \mid a \\ B &\rightarrow \epsilon \mid +AC \mid Cb \\ C &\rightarrow \epsilon \mid -ABc \end{aligned}$$

(a) Compute the First and Follow sets for the grammar.

$$\begin{aligned} FIRST([AB]) &= \{[\} \\ FIRST(a) &= \{a\} \\ FIRST(A) &= \{[, a\} \\ FIRST(\epsilon) &= \{\epsilon\} \\ FIRST(+AC) &= \{+\} \\ FIRST(-ABc) &= \{-\} \\ FIRST(C) &= \{\epsilon, -\} \\ FIRST(Cb) &= \{b, -\} \\ FIRST(B) &= \{\epsilon, +, -, b\} \end{aligned}$$

$$\begin{aligned} FOLLOW(B) &= \{], c\} \\ FOLLOW(C) &= \{b,], c\} \\ FOLLOW(A) &= \{\$, +, -, b, c,]\} \end{aligned}$$

(b) Give the LL(1) parsing table for the grammar.

| NON- TERMINAL | Input | | | | | | | |
|------------------|----------------------|--------------------------|-------------------|--------------------------|--------------------------|---------------------|----------------------|----|
| | [|] | a | b | c | + | - | \$ |
| A | $A \rightarrow [AB]$ | | $A \rightarrow a$ | | | | | |
| B | | $B \rightarrow \epsilon$ | | $B \rightarrow Cb$ | $B \rightarrow \epsilon$ | $B \rightarrow +AC$ | $B \rightarrow Cb$ | |
| C | | $C \rightarrow \epsilon$ | | $C \rightarrow \epsilon$ | $C \rightarrow \epsilon$ | | $C \rightarrow -ABc$ | |

Table 1: LL(1) parsing table

(c) Is this grammar LL(1)? and Why.

Yes, it is LL(1) grammar, which satisfies:

For any two distinct productions $A \rightarrow \alpha \mid \beta$,

For no terminal a do both α and β derive strings beginning with a .

At most one of α and β can derive the empty string.

If $\beta \xRightarrow{*} \epsilon$, then α does not derive any string beginning with a terminal in $FOLLOW(A)$. Likewise,

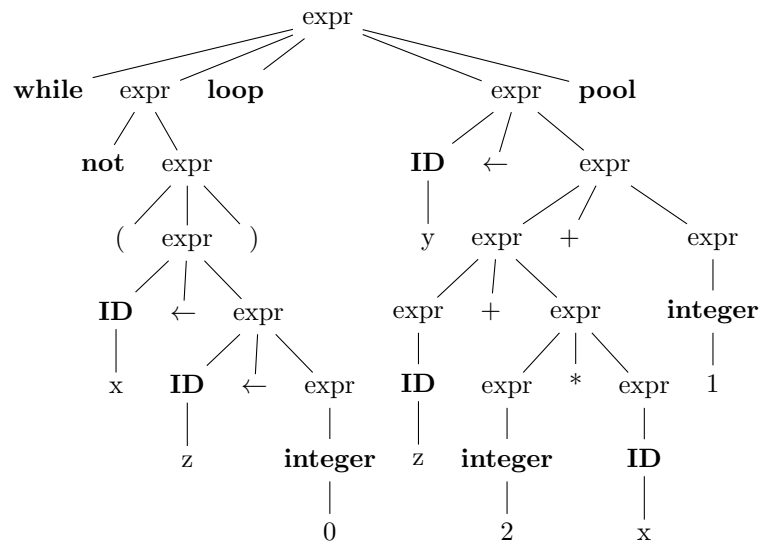
if $\alpha \xRightarrow{*} \epsilon$, then β does not derive any string beginning with a terminal in $FOLLOW(A)$.

In short, here're no 'conflicts' in the entries of the table.

5. 8 pts Using the context-free grammar for Cool given in Section 11 of the Cool manual, draw a parse tree for the following expression.

```
while not (x <- z <- 0) loop
  y <- z + 2 * x + 1
pool
```

Note that the context-free grammar by itself is ambiguous, so you will need to use the precedence and associativity rules in Section 11.1 to get the correct tree.



6. $4 \times 4 = 16$ **pts** Consider the following grammar describing a certain sort of nested lists:

$$\begin{aligned} S &\rightarrow T; S \mid \epsilon \\ T &\rightarrow U \star T \mid U \\ U &\rightarrow x \mid y \mid [S] \end{aligned}$$

S , T , and U are nonterminals, while others are terminals.

(a) Left-factor this grammar.

$$\begin{aligned} S &\rightarrow T; S \mid \epsilon \\ T &\rightarrow UT' \\ T' &\rightarrow \star T \mid \epsilon \\ U &\rightarrow x \mid y \mid [S] \end{aligned}$$

(b) Give the First and Follow sets for each nonterminal in the grammar obtained in part (a).

$$\begin{aligned} FIRST(U) &= \{x, y, [\} \\ FIRST(T) &= \{x, y, [\} \\ FIRST(S) &= \{x, y, [, \epsilon\} \\ FIRST(T') &= \{\star, \epsilon\} \end{aligned}$$

$$\begin{aligned} FOLLOW(S) &= \{\$, \} \\ FOLLOW(T) &= \{; \} \\ FOLLOW(T') &= \{; \} \\ FOLLOW(U) &= \{;, \star\} \end{aligned}$$

(c) Using this information, construct an LL(1) parsing table for the grammar obtained in part (a).

| NON- TERMINAL | Input | | | | | | |
|------------------|----------------------|--------------------------|----------------------|----------------------|--------------------------|---------------------------|--------------------------|
| | [|] | x | y | \star | ; | $\$$ |
| S | $S \rightarrow T; S$ | $S \rightarrow \epsilon$ | $S \rightarrow T; S$ | $S \rightarrow T; S$ | | | $S \rightarrow \epsilon$ |
| T | $T \rightarrow UT'$ | | $T \rightarrow UT'$ | $T \rightarrow UT'$ | | | |
| T' | | | | | $T' \rightarrow \star T$ | $T' \rightarrow \epsilon$ | |
| U | $U \rightarrow [S]$ | | $U \rightarrow x$ | $U \rightarrow y$ | | | |

Table 2: LL(1) parsing table

(d) Suppose we generated an LL(1) parser for the grammar using the table you constructed. What would go wrong if it tried to parse the following input string?

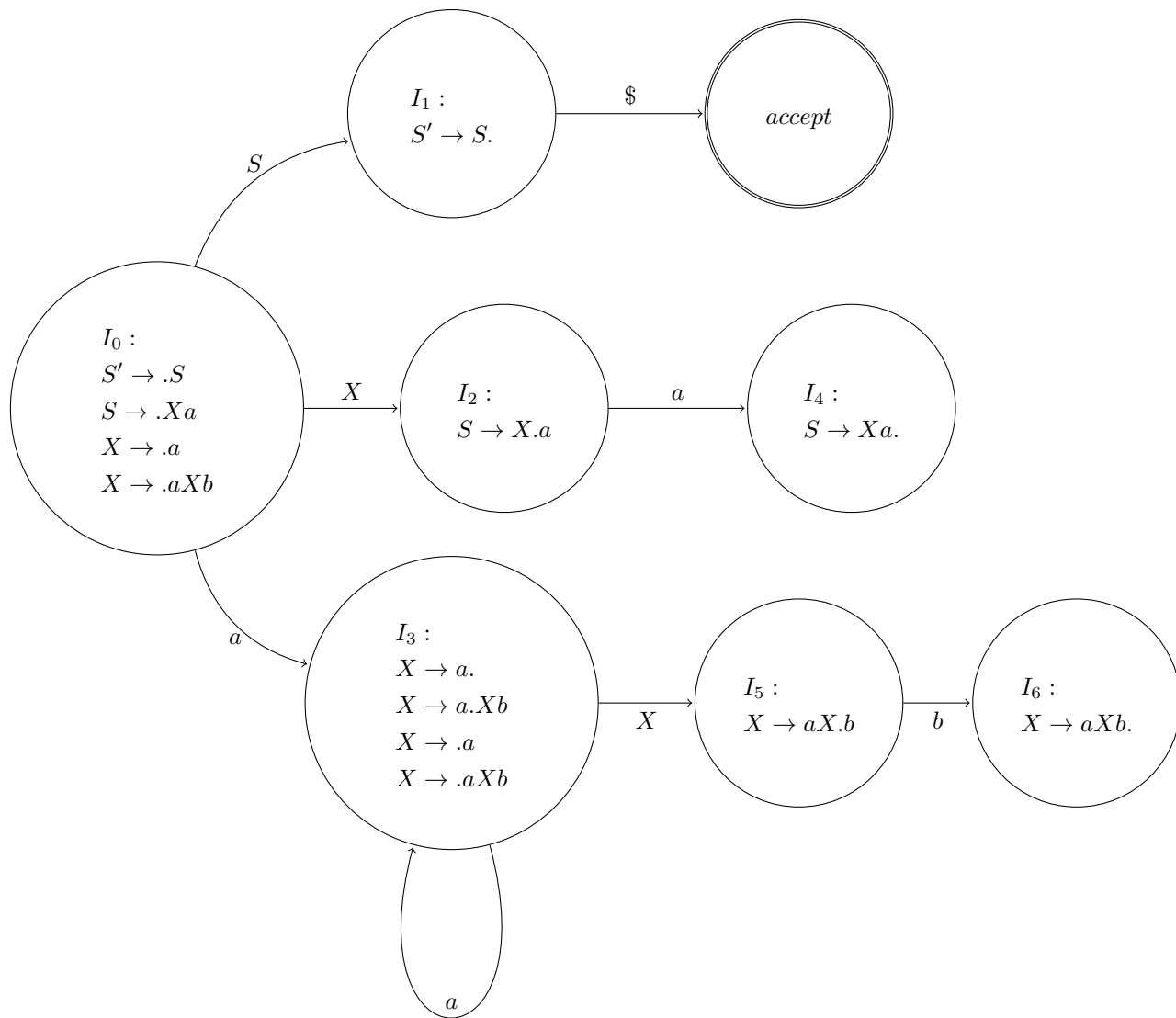
$$[x; y] \star [;$$

The error will occur at the time of inputting the last ';', which is in the state S and the ';' is illegal.

7. $3 \times 2 + 5 \times 2 = 16$ **pts** Consider the following CFG, which has the set of terminals $T = \{\mathbf{a}, \mathbf{b}\}$.

$$\begin{aligned} S &\rightarrow X\mathbf{a} \\ X &\rightarrow \mathbf{a} \mid \mathbf{a}X\mathbf{b} \end{aligned}$$

(a) Construct a DFA for viable prefixes of this grammar using LR(0) items.



- (b) Identify a shift-reduce conflict in this grammar under the SLR(1) rules.

| State | Actions | | | Goto | |
|-------|--------------|-----|------|------|-----|
| | a | b | $\$$ | S | X |
| 0 | s3 | | | s1 | s2 |
| 1 | | | acc | | |
| 2 | s4 | | | | |
| 3 | s3/r2 | r2 | | | s5 |
| 4 | | | r1 | | |
| 5 | | s6 | | | |
| 6 | r3 | r3 | | | |

Table 3: Parsing table

In the state I_3 , we can either reduce a to X or keep shifting when a is inputting.

- (c) Assuming that an SLR(1) parser resolves shift-reduce conflicts by choosing to shift, show the operation of such a parser on the input string **aaba**.

| Stack | Input | Action |
|-------|--------|-------------------------------|
| \$ | aaba\$ | shift |
| \$a | aba\$ | shift |
| \$aa | ba\$ | reduce by $X \rightarrow a$ |
| \$aX | ba\$ | shift |
| \$aXb | a\$ | reduce by $X \rightarrow aXb$ |
| \$X | a\$ | shift |
| \$Xa | \$ | reduce by $S \rightarrow Xa$ |
| \$S | \$ | accept |

- (d) Suppose that the production $X \rightarrow \epsilon$ is added to this grammar. Identify a reduce-reduce conflict in the resulting grammar under the SLR(1) rules.

Still in the state I_3 , we can either reduce a to X or applying $X \rightarrow \epsilon$