Oversætter - Week 2

1 - Writing Context-Free Grammars

Write unambiguous grammars for the following languages over the alphabet $\Sigma = \{a, b, c\}$

a

Words that match regular expression a^*b^* which contain more a's than b's

$$
\begin{aligned}
S &\to A \\
A &\to aA \\
A &\to aB \\
B &\to aBb \\
B &\to \varepsilon
\end{aligned}
$$

b

Palindromes

$$
\begin{aligned}
S &\to T \\
T &\to aTa \\
T &\to bTb \\
T &\to cTc \\
T &\to a|b|c \\
T &\to \varepsilon
\end{aligned}
$$

c

Write mosmlyacc grammar files for your grammars to check them. A grammar which does not cause conflicts is certain to be unambiguous.
However, for (b), it will not be possible to get a grammar without conflicts. Why?

Grammer files are uploaded separately as "1a.grm" and "1b.grm".
Since there is a look-ahead on 1, if you have a palindrome that has an odd

length, it's not possible to know what production you need to make, e.g it's not possible to know if it's $T \rightarrow aTa$ or $T \rightarrow a$.

2 - LL(1)-Parser Construction

Construct an LL(1) parser, taking the following grammar as a starting point:

$$\begin{aligned} Z &\rightarrow b \mid XYZ \\ Y &\rightarrow \varepsilon \mid c \\ X &\rightarrow Y \mid a \end{aligned}$$

with the terminal symbols a, b, and c.

a

Determine which nonterminals are nullable and calculate first sets of all right-hand sides of the productions.

Y is nullable since

$$NULLABLE(Y) = NULLABLE(\varepsilon) \vee NULLABLE(c) = true$$

X is nullable because

$$NULLABLE(X) = NULLABLE(Y) \vee NULLABLE(a) = true$$

Z is not nullable since

$$\begin{aligned} NULLABLE(Z) &= NULLABLE(a) \vee NULLABLE(XYZ) \\ &= NULLABLE(a) \vee \\ &\quad (NULLABLE(X) \wedge NULLABLE(Y) \wedge NULLABLE(Z)) \\ &= false \vee (true \wedge true \wedge NULLABLE(Z)) \\ &= NULLABLE(Z) \end{aligned}$$

which is an infinite loop, so Z is not nullable.

Or it can be calculated by fixed-point iteration

Right-hand side	Initialisation	Iteration 1	Iteration 2	Iteration 3
b	false	false	false	false
XYZ	false	false	false	false
ε	false	true	true	true
c	false	false	false	false
Y	false	false	true	true
a	false	false	false	false
Nonterminals				
Z	false	false	false	false
Y	false	true	true	true
X	false	false	true	true

We want to find first sets of the right hand sides of the production. We get that

$$FIRST(b) = \{b\}$$
$$FIRST(XYZ) = \{a, b, c\}$$
$$FIRST(\varepsilon) = \emptyset$$
$$FIRST(c) = \{c\}$$
$$FIRST(Y) = \{c\}$$
$$FIRST(a) = \{a\}$$

These can be calculated by fixed point iteration as well

Right-hand side	Initialisation	Iteration 1	Iteration 2	Iteration 3
b	$\emptyset$	{b}	{b}	{b}
XYZ	$\emptyset$	$\emptyset$	{a,b,c}	{a,b,c}
ε	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
c	$\emptyset$	{c}	{c}	{c}
Y	$\emptyset$	$\emptyset$	{c}	{c}
a	$\emptyset$	{a}	{a}	{a}

b

Calculate follow sets for all nonterminals (adding an extra start production to recognise the end of the input, denoted by "$")

We use the algorithm from page 59 in the book and use it on the grammar with $Z' \rightarrow Z\$$ added to it. We then get these constraints

$$\begin{aligned}
\{\$\} &\subseteq FOLLOW(Z) \\
\{a, b, c\} &\subseteq FOLLOW(X) \\
FOLLOW(Z) &\subseteq FOLLOW(X) \\
\{a, b, c\} &\subseteq FOLLOW(Y) \\
FOLLOW(X) &\subseteq FOLLOW(Y)
\end{aligned}$$

Now solving the constraints give us

$$\begin{aligned}
FOLLOW(X) &= \{a, b, c, \$\} \\
FOLLOW(Y) &= \{a, b, c, \$\} \\
FOLLOW(Z) &= \{\$\}
\end{aligned}$$

c

Determine the look-ahead sets of all productions and put together a parse table for a predictive parser.

Since $X \rightarrow Y$ and $Y \rightarrow \varepsilon$ are nullable, we get the following look-ahead sets for the productions

$$LA(Z' \rightarrow Z\$) = \{a, b, c, \$\} \tag{1}$$
$$LA(Z \rightarrow b) = \{b\} \tag{2}$$
$$LA(Z \rightarrow XYZ) = \{a, b, c\} \tag{3}$$
$$LA(Y \rightarrow \varepsilon) = \{a, b, c, \$\} \tag{4}$$
$$LA(Y \rightarrow c) = \{c\} \tag{5}$$
$$LA(X \rightarrow Y) = \{a, b, c, \$\} \tag{6}$$
$$LA(X \rightarrow a) = \{a\} \tag{7}$$

We create the parse table

Stack	a	b	c	$
Z'	Z$, 1	Z$, 1	Z$, 1	Z$, 1
X	Y, 6 $\vee$ a, 7	Y, 6	Y, 6	Y, 6
Y	ε, 4	ε, 4	ε, 4 $\vee$ c, 5	ε, 4
Z	XYZ, 3	XYZ, 3 $\vee$ b, 2	XYZ, 3	error
a	pop	error	error	error
b	error	pop	error	error
c	error	error	pop	error
$	error	accept	error	accept

There are 3 conflicts.

3 - SLR Parser Construction

Make up a very small grammar which contains left-recursion, to demonstrate that left-recursion is not a problem for LR-Parsing.

a

Show that your grammar does not generate conflicts (by providing a parse table).

I have created a grammar below which is using left recursion

$$S \to A\$ \qquad (0)$$
$$A \to Aa \qquad (1)$$
$$A \to b \qquad (2)$$

From the output of "3a.grm" we can create the parse table.

	a	b	$	A	S
0	s1	s1	s2	g2	g2
1		s3		g4	
2			a		
3		r2			
4	s5		r3		
5	r1				

And we can tell there are no conflicts.

b

Compare your grammar to an equivalent one that uses right-recursion. How does the parse stack grow when parsing input?

The equivalent right recursive grammar is

$$S \to A\$ \qquad (0)$$
$$A \to bB \qquad (1)$$
$$B \to aB \qquad (2)$$
$$B \to \varepsilon \qquad (3)$$

Parse stack for the left recursive grammar

Stack	Input	Action
ε	baa$	shift
b	aa$	reduce 2
A	aa$	shift
Aa	a$	reduce 1
A	a$	shift
Aa	$	reduce 1
A	$	reduce 0
S	$	accept

And the parse stack for the right recursive grammar

Stack	Input	Action
ε	baa$	shift
b	aa$	shift
ba	a$	shift
baa	$	reduce 3
baaB	$	reduce 2
baB	$	reduce 2
bB	$	reduce 1
A	$	reduce 0
S	$	accept

So right-recursive reduces when it has read the whole input while it for left-recursive reduces along the way. Thus, the stack is smaller for left recursive.