Exercises on Grammars

With solutions

1. Consider the following context-sensitive grammar.

$$\begin{array}{rcl} X & \to & \texttt{a}Y\texttt{b} \\ \texttt{a}Y & \to & \texttt{ab}Y \\ Y\texttt{b} & \to & \texttt{ab} \end{array}$$

   Describe the language generated by this grammar, assuming X is the start symbol.

   Solution: The language contains just two words: `abab` and `aab`.

2. Consider the following context-sensitive grammar.

$$\begin{array}{rcl} X & \to & \texttt{a}Y\texttt{b} \\ \texttt{a}Y & \to & \texttt{ba}Y \\ Y\texttt{b} & \to & \texttt{ab} \end{array}$$

   Describe the language generated by this grammar, assuming X is the start symbol.

   Solution: The language can be described by the regular expression $\texttt{b}^*\texttt{aab}$.

3. The following grammar describes C-style expressions that involve array elements and addition. Assume that **X** is a terminal that stands for a variable identifier and **N** is a terminal that stands for a numeric constant. The other terminals are `[`, `]`, and `+`.

$$E \quad ::= \quad \mathbf{X}\,\texttt{[}E\texttt{]} \mid E\texttt{+}E \mid \mathbf{X} \mid \mathbf{N}$$

   An example of an expression generated by this grammar is `a[i+2]+b`, assuming `a` and `b` are in **X** and `2` is in **N**.

   (a) Give an example of an expression generated by this grammar that admits two different parse trees, and thus demonstrates that the grammar is ambiguous.

   Solution: Something like `1+2+3` would suffice.

[0] I'd like to thank Maia Ramambason for finding and fixing some errors in an earlier version of this document.

(b) Resolve the ambiguity in the grammar, being sure to follow the standard rules of associativity and precedence for the standard arithmetic operators.

Solution: This should do the trick:

$$\begin{aligned} E &::= E\texttt{+}T \mid T \\ T &::= \mathbf{X}\,\texttt{[}E\texttt{]} \mid \mathbf{X} \mid \mathbf{N} \end{aligned}$$

(c) Describe the $NULLABLE$, $FIRST$, and $FOLLOW$ sets for each non-terminal in your grammar.[1]

Solution: For the grammar above:

$$\begin{aligned} NULLABLE(E) &= false \\ NULLABLE(T) &= false \\ FIRST(E) &= \{\mathbf{X}, \mathbf{N}\} \\ FIRST(T) &= \{\mathbf{X}, \mathbf{N}\} \\ FOLLOW(E) &= \{\$, \texttt{]}, \texttt{+}\} \\ FOLLOW(T) &= \{\$, \texttt{]}, \texttt{+}\} \end{aligned}$$

(d) Explain why your grammar is not suited to top-down parsing.

Solution: Because it's left-recursive.

(e) Rewrite your grammar so that it becomes suited to top-down parsing.

Solution: This should do the trick:

$$\begin{aligned} E &::= TE' \\ E' &::= \texttt{+}TE' \mid \epsilon \\ T &::= \mathbf{X}\,\texttt{[}E\texttt{]} \mid \mathbf{X} \mid \mathbf{N} \end{aligned}$$

(f) Describe the $NULLABLE$, $FIRST$, and $FOLLOW$ sets for each non-terminal in your new grammar.

Solution: For the grammar above:

$$\begin{aligned} NULLABLE(E) &= false \\ NULLABLE(E') &= true \\ NULLABLE(T) &= false \\ FIRST(E) &= \{\mathbf{X}, \mathbf{N}\} \end{aligned}$$

[1] Note that some authors avoid the $NULLABLE$ set. Instead, to indicate that a non-terminal X can generate the empty word, they put ϵ into $FIRST(X)$. Personally, I find this a bit of a hack, so I prefer to keep the $FIRST(X)$ set for the symbols that can actually begin a word generated from X, and separately to use $NULLABLE(X)$ to record whether X can generate the empty word.

$$
\begin{aligned}
FIRST(E') &= \{\texttt{+}\} \\
FIRST(T) &= \{\mathbf{X}, \mathbf{N}\} \\
FOLLOW(E) &= \{\$, \texttt{]}\} \\
FOLLOW(E') &= \{\$, \texttt{]}\} \\
FOLLOW(T) &= \{\texttt{+}, \$, \texttt{]}\}
\end{aligned}
$$

(g) Fill in the following predictive parsing table with a row for each non-terminal and a column for each terminal (including \$ for the end-of-input symbol). For each cell in row X and column t, identify which production rule should be used if the parser is seeking to parse the non-terminal X and the current terminal is t.

	+	[	]	**X**	**N**	\$
E						
⋮						

Do you find that your parser does not know what to do when it meets an **X**? This problem can either be fixed by using an extra token of lookahead, which complicates your parsing table with an extra dimension, or by just rewriting your grammar (yet again!).

Solution: The problem with the current grammar is that when we meet an **X** while trying to parse a T, we do not know whether **X** is a scalar-valued variable or an array-valued variable that will be followed by an array index in brackets. That is, we don't know whether to use the $T \rightarrow \mathbf{X}$ rule or the $T \rightarrow \mathbf{X}\,\texttt{[}E\texttt{]}$ rule.

This can be fixed by *left-factoring* the grammar like so:

$$
\begin{aligned}
E &::= TE' \\
E' &::= \texttt{+}TE' \mid \epsilon \\
T &::= \mathbf{X}T' \mid \mathbf{N} \\
T' &::= \texttt{[}E\texttt{]} \mid \epsilon
\end{aligned}
$$

The $NULLABLE$, $FIRST$, and $FOLLOW$ sets are extended with:

$$
\begin{aligned}
NULLABLE(T') &= true \\
FIRST(T') &= \{\texttt{[}\} \\
FOLLOW(T') &= \{\texttt{+}, \$, \texttt{]}\}
\end{aligned}
$$

The predictive parsing table is:

	+	[	]	**X**	**N**	$
E				(1)	(1)	
E'	(2)		(3)			(3)
T				(4)	(5)	
T'	(7)	(6)	(7)			(7)

where the production rules are numbered like so:

$$
\begin{array}{lrcl}
(1) & E & \to & TE' \\
(2) & E' & \to & +TE' \\
(3) & E' & \to & \epsilon \\
(4) & T & \to & \mathbf{X}T' \\
(5) & T & \to & \mathbf{N} \\
(6) & T' & \to & [E] \\
(7) & T' & \to & \epsilon
\end{array}
$$

(h) We shall now turn our attention to bottom-up (shift/reduce) parsing. Return to the unambiguous grammar that you produced in part (b), and add a new start symbol, $S \to E$.

i. Write out the initial item-set, which is obtained by taking the closure of the item $S \to \bullet E$.
Solution:
$S_0 = \{S \to \bullet E, E \to \bullet E{+}T, E \to \bullet T, T \to \bullet \mathbf{X}\,[E]\,, T \to \bullet \mathbf{X}, T \to \bullet \mathbf{N}\}$.

ii. Write out the rest of the item-sets, labelling the transitions between them with the terminals and non-terminals of your grammar.
Solution:

$$
\begin{array}{rcl}
S_0 & = & \{S \to \bullet E, E \to \bullet E{+}T, E \to \bullet T, T \to \bullet \mathbf{X}\,[E]\,, T \to \bullet \mathbf{X}, T \to \bullet \mathbf{N}\} \\
S_1 & = & \{S \to E\bullet, E \to E\bullet{+}T\} \\
S_2 & = & \{E \to T\bullet\} \\
S_3 & = & \{T \to \mathbf{X}\bullet\,[E]\,, T \to \mathbf{X}\bullet\} \\
S_4 & = & \{T \to \mathbf{N}\bullet\} \\
S_5 & = & \{E \to E{+}\bullet T, T \to \bullet \mathbf{X}\,[E]\,, T \to \bullet \mathbf{X}, T \to \bullet \mathbf{N}\} \\
S_6 & = & \{T \to \mathbf{X}\,[\bullet E]\,, E \to \bullet E{+}T, E \to \bullet T, T \to \bullet \mathbf{X}\,[E]\,, T \to \bullet \mathbf{X}, T \to \bullet \mathbf{N}\} \\
S_7 & = & \{E \to E{+}T\bullet\} \\
S_8 & = & \{T \to \mathbf{X}\,[E\bullet]\,, E \to E\bullet{+}T\} \\
S_9 & = & \{T \to \mathbf{X}\,[E]\,\bullet\}
\end{array}
$$

with transitions $\{(S_0, E, S_1), (S_0, T, S_2), (S_0, \mathbf{X}, S_3), (S_0, \mathbf{N}, S_4), (S_1, +, S_5), (S_3, [, S_6), (S_5, T, S_7), (S_5, \mathbf{X}, S_3), (S_5, \mathbf{N}, S_4), (S_6, E, S_8), (S_6, \mathbf{X}, S_3), (S_6, \mathbf{N}, S_4), (S_8,], S_9), (S_8, +, S_5)\}$.

iii. Fill in the following shift/reduce parsing table, with one 'action' column for each terminal (including \$), one 'goto' column for each non-terminal, and one row for each item-set.

	action						*goto*				
	+	[	]	**X**	**N**	\$	S	E	E'	T	T'
0											
1											
⋮											

Solution: The shift/reduce parsing table is

	action						*goto*			
	+	[	]	**X**	**N**	\$	E	E'	T	T'
0				$s3$	$s4$		1		2	
1	$s5$					acc				
2	$r2$		$r2$			$r2$				
3	$r4$	$s6$	$r4$			$r4$				
4	$r5$		$r5$			$r5$				
5				$s3$	$s4$				7	
6							8			
7	$r1$		$r1$			$r1$				
8	$s5$		$s9$							
9	$r3$		$r3$			$r3$				

where the production rules are numbered like so:

$$
\begin{array}{lrcl}
(1) & E & \rightarrow & E\texttt{+}T \\
(2) & E & \rightarrow & T \\
(3) & T & \rightarrow & \mathbf{X}\,\texttt{[}E\texttt{]} \\
(4) & T & \rightarrow & \mathbf{X} \\
(5) & T & \rightarrow & \mathbf{N}
\end{array}
$$