Programming Languages and Computation

Week 2: Context Free Grammars

* 1. Consider the following CFG G with start symbol R:

$$\begin{array}{lcl} R & ::= & XRX \mid S \\ S & ::= & aTb \mid bTa \\ T & ::= & XTX \mid X \mid \epsilon \\ X & ::= & a \mid b \end{array}$$

(a) What are the non-terminals of G?

(b) What are the terminals of G?

(c) Give three strings in $L(G)$.

(d) Give three strings not in $L(G)$.

(e) True or false: $T \rightarrow aba$.

(f) True or false: $T \rightarrow^* aba$.

(g) True or false: $T \rightarrow T$.

(h) True or false: $T \rightarrow^* T$.

(i) True or false: $XXX \rightarrow^* aba$.

(j) True or false: $X \rightarrow^* aba$.

(k) True or false: $T \rightarrow^* XX$.

(l) True or false: $T \rightarrow^* XXX$.

(m) True or false: $S \rightarrow^* \epsilon$.

* 2. Figure 1 contains a grammar for the Brischeme language from this week's lab sheet.

You will need to read the section Grammars can Express Sequences from the end of in the course notes to understand the notation $Form^*$, $SExpr^*$, $Ident^*$ and $IdChar^*$ from this grammar. Unfortunately, I did not have time to cover this in the lecture.

The syntax of the *Brischeme* programming language (ignoring whitespace) is given by the CFG with:

- Terminals: $0, 1, \ldots, 9$, $a, b, \ldots, z, A, B, \ldots, Z$, _, !, ?, <, =, +, −, ∗, /, (,), define, lambda, not, or, and, #t, #f.
- Nonterminals: *Prog*, *Form*, *SExpr*, *Ident*, *Literal*, *Num*, *Bool*, *Digit*, *LChar*, *IdChar*, *Primop*.
- The production rules are:

$$
\begin{array}{rcl}
\textit{Prog} & ::= & \textit{Form}^* \\
\textit{Form} & ::= & \textit{SExpr} \\
 & | & \text{(define } \textit{Ident SExpr } \text{)} \\
\textit{SExpr} & ::= & \textit{Literal} \\
 & | & \textit{Ident} \\
 & | & \text{(} \textit{SExpr SExpr}^* \text{)} \\
 & | & \text{(} \textit{Primop SExpr}^* \text{)} \\
 & | & \text{(lambda (} \textit{Ident}^* \text{) } \textit{SExpr } \text{)} \\
\textit{Ident} & ::= & \textit{LChar IdChar}^* \\
\textit{LChar} & ::= & \text{a} \mid \text{b} \mid \cdots \mid \text{z} \\
\textit{IdChar} & ::= & a \mid b \mid \cdots \mid z \mid A \mid B \mid \cdots \mid \text{Z} \mid \text{!} \mid \text{?} \mid \text{\_} \\
\textit{Literal} & ::= & \textit{Bool} \mid \textit{Num} \\
\textit{Bool} & ::= & \text{\#t} \mid \text{\#f} \\
\textit{Num} & ::= & \textit{Digit Digit}^* \\
\textit{Digit} & ::= & 0 \mid 1 \mid \cdots \mid 9 \\
\textit{Primop} & ::= & \text{+} \mid \text{-} \mid \text{*} \mid \text{/} \mid \text{and} \mid \text{or} \mid \text{not} \mid \text{<} \mid \text{=}
\end{array}
$$

Figure 1: Brischeme (ignoring whitespace).

Which of the following are valid Brischeme programs (i.e. strings in the language of that grammar)? You do not have to give the derivations (but you should work through them in your head).

(a) (define x 32) (+ 4 x)

(b) (define x 32) (+ 4 y)

(c) (define x 32) (+ 4 5x)

(d) (define aC3! 32) (+ 4 aC3!)

(e) (define AC3! 32) (+ 4 AC3!)

(f) (+ (define x 32) x 4)

(g) (lambda x (+ x x))

(h) (define f (lambda (x y) (+ x (* 2 y))))

(i) (lambda (x b) (+ x (not b)))

(j) (lambda (x b) (if x))

(k) (lambda (x b) ((if b + -) 2 3))

(l) ((lambda () 3))

** 3. Design CFGs for the following programming language lexemes over the ASCII alphabet. You will find it convenient to use abbreviations like $\cdots$ to help present the expressions compactly.

(a) A *C program identifier* is any string of length at least 1 containing only letters ('a'–'z', lower and uppercase), digits ('0'–'9') and the underscore, and which begins with a letter or the underscore.

(b) An *integer literal* is any string taking one of the following forms:

- a non-empty sequence of digits (decimal)
- a non-empty sequence of characters from '0'–'9','a'–'e' (upper or lowercase) that are preceded by "0x" (hexadecimal)
- a non-empty sequence of bits '0' and '1' that are preceded by "0b" (binary)

* 4. Consider the following grammar, which describes the structure of statements in an imperative

programming language.

Prog	::=	*Stmt Stmts*	(1)
Stmt	::=	if exp then *Stmt* else *Stmt*	(2)
	|	while exp do *Stmt*	(3)
	|	skip	(4)
	|	id ← exp	(5)
	|	{*Stmt Stmts*}	(6)
Stmts	::=	; *Stmt Stmts*	(7)
	|	ϵ	(8)

In it expressions appear only as a terminal symbols exp because the structure of expressions is not important to the exercises. In total, the terminal symbols are: if, then, else, while, do, skip, id, exp, the left and right braces, the end-of-input marker and the semicolon. The rules are numbered to make constructing the parse tables easier. The language of this grammar (start symbol *Prog*) includes strings such as:

while exp do id ← exp

i.e. strings that show the control structure of the program without specifying the particular expressions involved.
The nullable, first and follow maps for the nonterminals in this grammar are as follows:

Nonterminal	Nullable?	First	Follow
Prog	×	if, while, skip, id, {	
Stmt	×	if, while, skip, id, {	else, ;, }
Stmts	✓	;	}

(a) For each rule $X ::= \alpha$ numbered (1) – (8), compute First(α), the set of terminal symbols that can start any string derivable from the rule right-hand side α.

(b) Construct the parsing table for the grammar.

(c) Is the grammar LL(1)?

* 5. Consider now the following grammar:

Prog	::=	*Stmt Stmts*	(1)
Stmt	::=	if bexp then *Stmt* else *Stmt*	(2)
	|	while bexp do *Stmt*	(3)
	|	skip	(4)
	|	id ← *aexp*	(5)
	|	*Stmt Stmts*	(6)
Stmts	::=	; *Stmt Stmts*	(7)
	|	ϵ	(8)

This grammar is the same as the previous one, except that braces have been removed in rule 6.

The Nullable, First and Follow maps for this grammar can be tabulated as follows.

Nonterminal	Nullable?	First	Follow
Prog	×	if, while, skip, id	
Stmt	×	if, while, skip, id	;, else
Stmts	✓	;	;, else

(a) For each rule $X ::= \alpha$ numbered (1) – (8), compute First(α), the set of terminal symbols that can start any string derivable from the rule right-hand side α.

(b) Construct the parsing table for this grammar.

(c) Is the grammar LL(1)?

** 6. Give CFGs for the following languages. The later parts are more difficult than 2-star.

(a) All odd length strings over $\{a, b\}$.

(b) All strings over $\{a, b\}$ that contain aab as a substring.

(c) The set of strings over $\{a, b\}$ with more a than b. Hint: every string w with at least as many a as b (possibly the same number of a as b) can be characterised inductively as follows. Either:

- w is just a
- or, w is of shape avb with v containing at least as many a as b
- or, w is of shape bva with v containing at least as many a as b
- or, w is of shape v_1v_2 with v_1 and v_2 each separately containing at least as many a as b
- or, w is the empty string

(d) The complement of the language $\{a^n b^n \mid n \geq 0\}$ over $\{a, b\}$. Hint: express "not of shape $a^n b^n$ for some n" into one or more positive (i.e. without using *not* or similar) conditions.

(e) $\{v\#w \mid v, w \in \{a, b\}^*$ and the reverse of v is a substring of $w\}$, over $\{a, b, \#\}$.