Samuel Hoogewind

CS 214

Project 2

1. Using these ideas, define the constructs below with BNF productions. Underline the terminals in your productions, to distinguish them from the nonterminals. You may assume that the following have already been defined:
2. <letter> ::= A | B | C | D | E | F | G | H | I | J | K | L
3. | M | N | O | P | Q | R | S | T | U | V | W
4. | X | Y | Z | a | b | c | d | e | f | g | h
5. | i | j | k | l | m | n | o | p | q | r | s
6. | t | u | v | w | x | y | z
7. <digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

* 1. A Java character literal  
       
     <java\_char> ::= ‘ <character> ‘

<character> ::= <letter> | <digit> | <symbol>

<symbol> ::= ! | " | # | $ | % | & | ' | ( | ) | \* | + | , | \ | : | ; | < | = | > | ? | @ | [ | \ | ] | ^ \ | \_ | ' | { | } | ~

* 1. A Java character string literal  
       
     <java\_string> ::= “ <string\_sequence> “

<string\_sequence> ::= <letter> <string\_sequence> | <digit> <string\_sequence> | <symbol> <string\_sequence> | epsilon

<symbol> ::= ! | " | # | $ | % | & | ' | ( | ) | \* | + | , | \ | : | ; | < | = | > | ? | @ | [ | \ | ] | ^ \ | \_ | ' | { | } | ~

* 1. A Java integer literal  
       
     <java\_int> ::= <first\_digit> <digit\_sequence>

<first\_digit> ::= - | <digit>

<digit\_sequence> ::= <digit> <digit\_sequence> | epsilon

* 1. A Java real (floating point) literal  
       
     <java\_float> ::= <first\_digit> <digit\_sequence> . <digit> <digit\_sequence>

<first\_digit> ::= - | <digit>

<digit\_sequence> ::= <digit> <digit\_sequence> | epsilon

* 1. A Java identifier:  
       
     <java\_identifier> ::= <first\_char> <char\_sequence>  
     <first\_char> ::= <letter> | \_ | $  
     <char\_sequence> ::= <letter> <char\_sequence> | <digit> <char\_sequence> | epsilon
  2. A Java function declaration (prototype):  
       
     <java\_dec> ::= <modifier> <type> <java\_identifier> ( <parameter\_list> ) {}

<modifier> ::= abstract | final | interface | native | private | protected | public | static | strict | synchronized | transient | volatile

<type> ::= byte | short | int | long | float | double | boolean | char | string | void

<java\_identifier> ::= <first\_char> <char\_sequence>

<first\_char> ::= <letter> | \_ | $

<char\_sequence> ::= <letter> <char\_sequence> | <digit> <char\_sequence> | \_ <char\_sequence> | epsilon

<parameter\_list> ::= <parameter> | <parameter> , <parameter\_list > | epsilon

<parameter> ::= <type> <java\_identifier>

* 1. A Java if statement (you may assume that the nonterminals <statement> and <expression> are defined elsewhere):  
       
     <java\_if> ::= if ( <expression> ) { <statement> } <elseif\_part> <else\_part>

<elseif\_part> ::= else if ( <expression> ) { <statement> } <elseif\_part> <else\_part> | epsilon

<else\_part> ::= else { <statement> } | epsilon

* 1. A Java while statement (you may assume that the nonterminals <statement> and <expression> are defined elsewhere):

<java\_while> ::= while ( <expression> ) { <statement> }

1. Prove that the following grammar is ambiguous:

<S> ::= <A>  
<A> ::= <A> + <A> | <id>  
<id> ::= a | b | c

A grammar is ambiguous if you can have a statement be developed by two different parse trees.

<S> ::= <A>

<S> ::= <A> + <A>

<S> ::= <id> + <A>

<S> ::= a + <A>

<S> ::= a + <id>

<S> ::= a + b

<S> ::= <A>

<S> ::= <A> + <A>

<S> ::= <A> + <id>

<S> ::= <A> + b

<S> ::= <id> + b

<S> ::= a + b

1. Give a left-most derivation for A = A \* (B + C) using the following BNF grammar:

<assign> ::= <id> = <expr>  
<id> ::= A | B | C  
<expr> ::= <expr> + <term> | <term>  
<term> ::= <term> \* <factor> | <factor>  
<factor> ::= ( <expr> ) | <id>

<assign> ::= <id> = <expr>

<assign> ::= A = <expr>

<assign> ::= A = <term>

<assign> ::= A = <term> \* <factor>

<assign> ::= A = <factor> \* <factor>

<assign> ::= A = <id> \* <factor>

<assign> ::= A = A \* <factor>

<assign> ::= A = A \* ( <expr> )

<assign> ::= A = A \* (<expr> + <term>)

<assign> ::= A = A \* (<term> + <term>)

<assign> ::= A = A \* (<factor> + <term>)

<assign> ::= A = A \* (<id> + <term>)

<assign> ::= A = A \* (B + <term>)

<assign> ::= A = A \* (B + <factor>)

<assign> ::= A = A \* (B + <id>)

<assign> ::= A = A \* (B + C)