CSCI3136
Assignment 4

Instructor: Alex Brodsky

Due: 9:00am, Friday, June 14, 2019

1. Consider the grammar in Figure 1:

   (a) **[5 marks]** Give a parse tree for the following program fragment.

```
def fib( n ) {
  if ( less_than( n, 2 ) ) {
    return n
  }
  return add( fib( sub( n, 1 ) ), fib( sub( n, 2 ) ) )
}
```

   See parse tree at end of solution.

   (b) **[5 marks]** Is this grammar ambiguous? Give an intuitive justification.

   This grammar is not ambiguous because at any point in the parse there is only one choice on how to expand the given nonterminal in order to yield the correct derivation. E.g., There is only one way to expand *Arglist* or *Staements* or *ParamList*.

   (c) **[10 marks]** Prove that this grammar is not LL(1). Hint: You can do this by constructing the FIRST, FOLLOW, and PREDICT sets.

   This is not an LL(1) grammar because the *Else* and *Expression* productions have nondisjoint PREDICT sets. (See below).

FIRST

Symbol	FIRST(Symbol)
`def`	{‘`def`′}
`id`	{`id`}
`if`	{‘`if`′}
`else`	{‘`else`′}
`return`	{‘`return`′}
`integer`	{‘`integer`′}
`,`	{‘,′}
`{`	{‘{′}
`}`	{‘}′}
`(`	{‘(′}
`)`	{‘)′}
MethodDecl	{‘`def`′}
ParamList	{`id`, ϵ}
ParamListTail	{‘,′, ϵ}
Param	{`id`}
Block	{′{′}
Statements	{`if`, `return`, ϵ}
Statement	{`if`, `return`}
IfStatement	{`if`}
Else	{`else`, ϵ}
ReturnStatement	{`return`}
Expression	{`id`, `integer`}
ArgList	{`id`, ϵ}
ArgListTail	{‘,′, ϵ}
Arg	{`id`, `integer`}

FOLLOW

Symbol	FOLLOW(Symbol)
MethodDecl	{ϵ}
ParamList	{‘)′}
ParamListTail	{‘)′}
Param	{‘)′, ‘,′}
Block	{`if`, `return`, ϵ}
Statements	{‘}′}
Statement	{`if`, `return`, ‘}′}
Expression	{‘)′, ‘,′}
ArgList	{‘)′}
ArgListTail	{‘)′}
Arg	{‘)′, ‘,′}

PREDICT

Production	PREDICT(Production)
MethodDecl → ′`def`′ *id* ′(′ *ParamList* ′)′ *Block*	{‘`def`′}
ParamList → ϵ	{′)′}
ParamList → *Param ParamListTail*	{`id`}
ParamListTail → ϵ	{′)′}
ParamListTail → ′,′ *Param ParamListTail*	{′,′}
Param → *id*	{`id`}
Block → ′{′ *Statements* ′}′	{′{′}
Statements → ϵ	{′}′}
Statements → *Statement Statements*	{`if`, `return`}
Statement → *IfStatement*	{`if`}
Statement → *ReturnStatement*	{`return`}
IfStatement → ′`if`′ ′(′ *Expression* ′)′ *Block Else*	{`if`}
Else → ϵ	{`if`, `return`, ′}}
Else → ′`else`′ *Block*	{`else`}
Else → ′`else`′ *IfStatement*	{`else`}
ReturnStatement → ′`return`′ *Expression*	{`return`}
Expression → *integer*	{`integer`}
Expression → *id*	{`id`}
Expression → *id* ′(′ *ArgList* ′)′	{`id`}
ArgList → ϵ	{′)′}
ArgList → *Expression ArgListTail*	{`id`}
ArgListTail → ϵ	{′)′}
ArgListTail → ′,′ *Arg ArgListTail*	{`id`}
Arg → *Expression*	{`id`}

(d) **[10 marks]** Modify the grammar so that it is LL(1).

To rectify this grammar, we need to fix the *Else* and *Expression* productions:

$$\begin{aligned}
\textit{Else} &\rightarrow \epsilon \\
\textit{Else} &\rightarrow \text{'}\texttt{else}\text{'}\ \textit{ElseBody} \\
\textit{ElseBody} &\rightarrow \textit{Block} \\
\textit{ElseBody} &\rightarrow \textit{IfStatement} \\
\textit{Expression} &\rightarrow \textit{integer} \\
\textit{Expression} &\rightarrow \textit{id}\ \textit{MethodArgs} \\
\textit{MethodArgs} &\rightarrow \epsilon \\
\textit{MethodArgs} &\rightarrow \text{'('}\ \textit{ArgList}\ \text{')'}
\end{aligned}$$

2. **[10 marks]** Give a context-free grammar that generates the language of properly nested brackets, such that if the last token is a right square bracket, it closes all remaining open left brackets. I.e., $\Sigma = \{\text{`('}, \text{`)'}, \text{`]'}\}$ and words such as `(()())` and `(()(]` are in the language, but words such as `(])`, `)(`, and `())` are not.

   The following grammar, starting on S, generates the language of nested brackets.

$$\begin{aligned}
S &\rightarrow A \\
S &\rightarrow B\text{']'} \\
A &\rightarrow \epsilon \\
A &\rightarrow A\ A \\
A &\rightarrow \text{'('}\ A\ \text{')'} \\
B &\rightarrow \epsilon \\
B &\rightarrow B\ B \\
B &\rightarrow \text{'('}\ B \\
B &\rightarrow \text{'('}\ B\ \text{')'}
\end{aligned}$$

3. **[10 marks]** Give a context-free grammar that generates the language

$$L = \{\sigma \in \{a, b, c, d\}^* \mid (2|\sigma|_a = |\sigma|_b) \vee (|\sigma|_a = 2|\sigma|_b)\}$$

   Note: The notation $|\sigma|_a$ means the number of a's in σ.

The following grammar, starting on S, generates L.

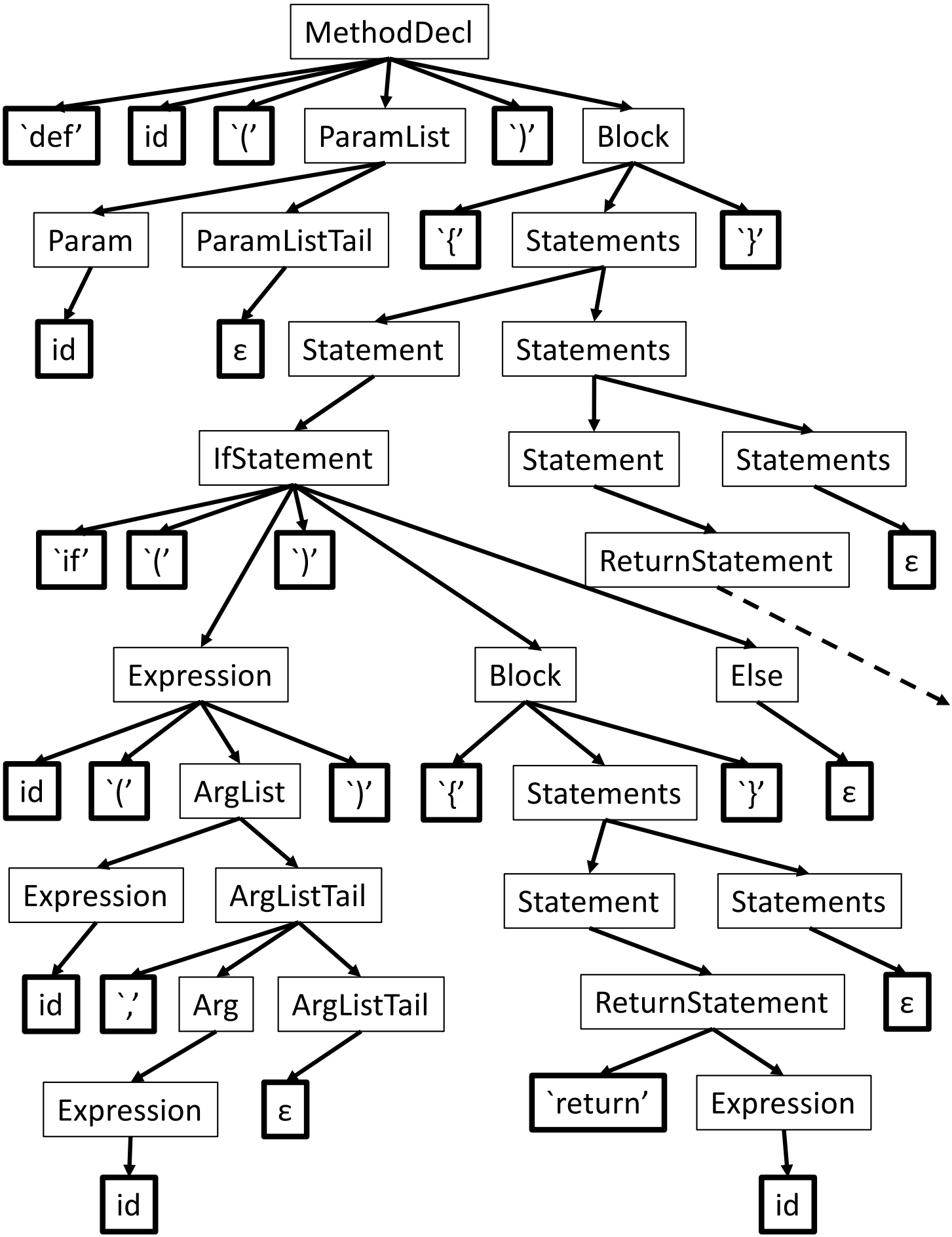
$$
\begin{array}{lcl}
S & \to & A \\
S & \to & B \\
A & \to & \epsilon \\
A & \to & A\ A \\
A & \to & \texttt{'c'}\ A \\
A & \to & \texttt{'d'}\ A \\
A & \to & A\ \texttt{'a'}\ A\ \texttt{'a'}\ A\ \texttt{'b'}\ A \\
A & \to & A\ \texttt{'a'}\ A\ \texttt{'b'}\ A\ \texttt{'a'}\ A \\
A & \to & A\ \texttt{'b'}\ A\ \texttt{'a'}\ A\ \texttt{'a'}\ A \\
B & \to & \epsilon \\
B & \to & B\ B \\
B & \to & \texttt{'c'}\ B \\
B & \to & \texttt{'d'}\ B \\
B & \to & B\texttt{'a'}\ B\ \texttt{'b'}\ B\ \texttt{'b'}\ B \\
B & \to & B\texttt{'b'}\ B\ \texttt{'b'}\ B\ \texttt{'a'}\ B \\
B & \to & B\texttt{'b'}\ B\ \texttt{'a'}\ B\ \texttt{'b'}\ B
\end{array}
$$

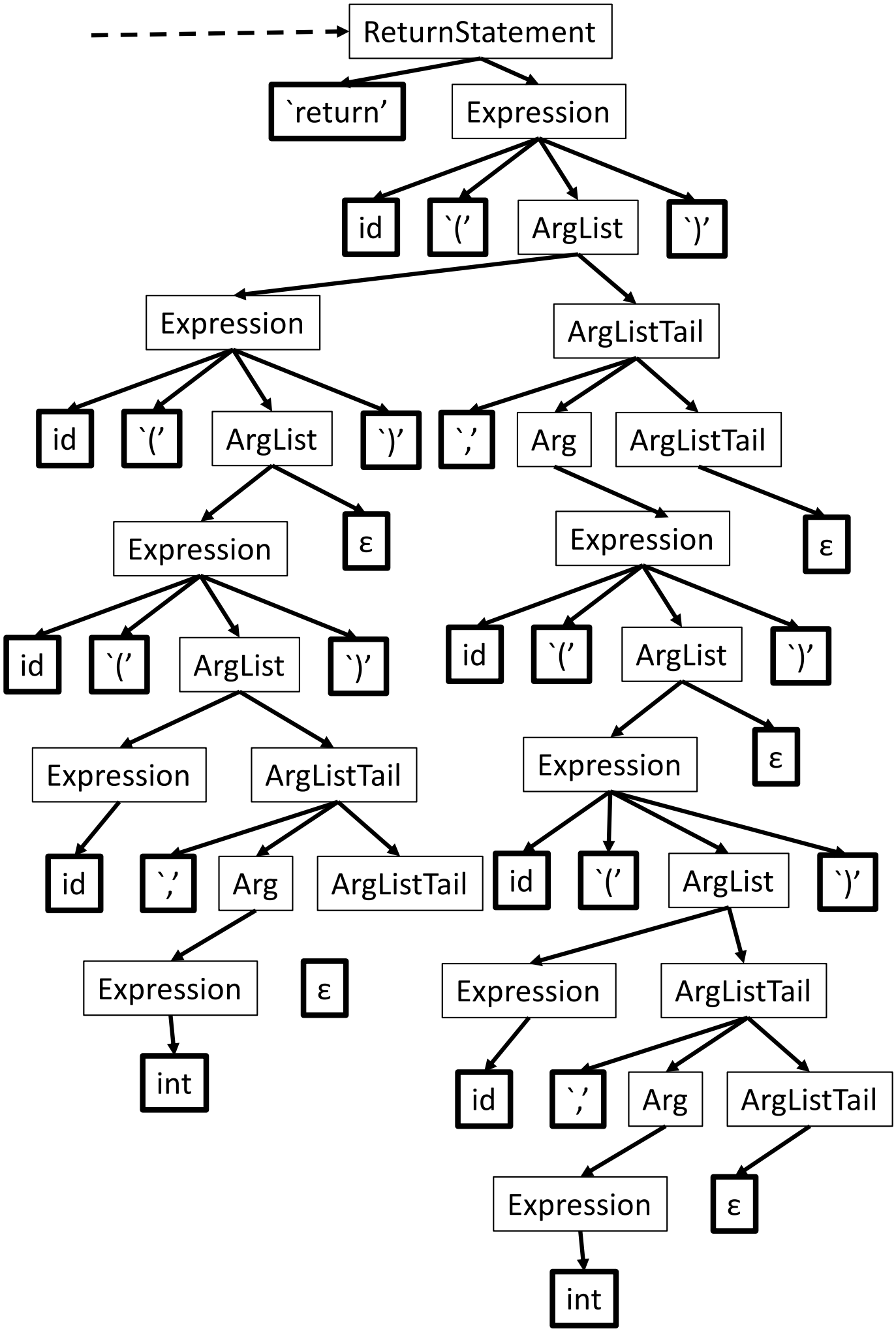
The idea behind this grammar is:

(a) `c` and `d` symbols are just filler and can be ignored.
(b) The grammar decides from S if it is deriving a string with twice as many `a`'s as `b`'s (A) or vice versa.
(c) The two parts are the same except the place of `a`'s and `b`'s are flipped.
(d) Without loss of generality, assume the string we are deriving has twice as many `a`'s as `b`'s, and has no `c`'s or `d`'s. Then there exists a substring that is of the form `aab`, `aba`, or `baa`. Replacing this substring with A, we now have a shorter string that still has as twice as many `a`'s ad `b`'s. Thus, we are now one step away from deriving the string. A simple argument by induction can be made to show that in this fashion we can derive the entire string starting from the symbole A.

$$
\begin{array}{rcl}
\textit{MethodDecl} & \rightarrow & \texttt{'def'}\ \textit{id}\ \texttt{'('}\ \textit{ParamList}\ \texttt{')'}\ \textit{Block} \\
\textit{ParamList} & \rightarrow & \epsilon \\
\textit{ParamList} & \rightarrow & \textit{Param}\ \textit{ParamListTail} \\
\textit{ParamListTail} & \rightarrow & \epsilon \\
\textit{ParamListTail} & \rightarrow & \texttt{','}\ \textit{Param}\ \textit{ParamListTail} \\
\textit{Param} & \rightarrow & \textit{id} \\
\textit{Block} & \rightarrow & \texttt{'\{'}\ \textit{Statements}\ \texttt{'\}'} \\
\textit{Statements} & \rightarrow & \epsilon \\
\textit{Statements} & \rightarrow & \textit{Statement}\ \textit{Statements} \\
\textit{Statement} & \rightarrow & \textit{IfStatement} \\
\textit{Statement} & \rightarrow & \textit{ReturnStatement} \\
\textit{IfStatement} & \rightarrow & \texttt{'if'}\ \texttt{'('}\ \textit{Expression}\ \texttt{')'}\ \textit{Block}\ \textit{Else} \\
\textit{Else} & \rightarrow & \epsilon \\
\textit{Else} & \rightarrow & \texttt{'else'}\ \textit{Block} \\
\textit{Else} & \rightarrow & \texttt{'else'}\ \textit{IfStatement} \\
\textit{ReturnStatement} & \rightarrow & \texttt{'return'}\ \textit{Expression} \\
\textit{Expression} & \rightarrow & \textit{integer} \\
\textit{Expression} & \rightarrow & \textit{id} \\
\textit{Expression} & \rightarrow & \textit{id}\ \texttt{'('}\ \textit{ArgList}\ \texttt{')'} \\
\textit{ArgList} & \rightarrow & \epsilon \\
\textit{ArgList} & \rightarrow & \textit{Expression}\ \textit{ArgListTail} \\
\textit{ArgListTail} & \rightarrow & \epsilon \\
\textit{ArgListTail} & \rightarrow & \texttt{','}\ \textit{Arg}\ \textit{ArgListTail} \\
\textit{Arg} & \rightarrow & \textit{Expression}
\end{array}
$$

Figure 1: A Grammar for function definitions.


MethodDecl
`def'
id
`('
ParamList
`)'
Block
Param
ParamListTail
`{'
Statements
`}'
id
ε
Statement
Statements
IfStatement
Statement
Statements
`if'
`('
`)'
ReturnStatement
ε
Expression
Block
Else
id
`('
ArgList
`)'
`{'
Statements
`}'
ε
Expression
ArgListTail
Statement
Statements
id
`,'
Arg
ArgListTail
ReturnStatement
ε
Expression
ε
`return'
Expression
id
id


ReturnStatement
`return'
Expression
id
`('
ArgList
`)'
Expression
ArgListTail
id
`('
ArgList
`)'
`,'
Arg
ArgListTail
Expression
ε
Expression
ε
id
`('
ArgList
`)'
id
`('
ArgList
`)'
Expression
ArgListTail
Expression
ε
id
`,'
Arg
ArgListTail
id
`('
ArgList
`)'
Expression
ε
Expression
ArgListTail
int
id
`,'
Arg
ArgListTail
Expression
ε
int

CSCI3136: Assignment 4

Summer 2019

Student Name	Login ID	Student Number	Student Signature

	Mark
Question 1a	/5
Question 1b	/5
Question 1c	/10
Question 1d	/10
Question 2	/10
Question 3	/10
Total	**/50**

Comments:

Assignments are due by 9:00am on the due date. Assignments *must* be submitted electronically via Brightspace. Please submit a PDF for the written work. You can do your work on paper and then scan in and submit the assignment.

Plagiarism in assignment answers will not be tolerated. By submitting their answers to this assignment, the authors named above declare that its content is their original work and that they did not use any sources for its preparation other than the class notes, the textbook, and ones explicitly acknowledged in the answers. Any suspected act of plagiarism will be reported to the Facultys Academic Integrity Officer and possibly to the Senate Discipline Committee. The penalty for academic dishonesty may range from failing the course to expulsion from the university, in accordance with Dalhousie Universitys regulations regarding academic integrity.