

COMP314W2  
Assignment 2 - 2020

1. Purpose

The purpose of this assignment is to write a Java program which will allow a *s-grammar*  $G = (V, T, S, P)$  (see slides 3 and 4 of the slide set *Context-Free Grammars and Languages II*), and strings to be inputted. The program should check that the grammar is a valid *s-grammar* and then parse the strings to determine whether the strings are members of the language that is defined by the grammar.

All elements in the set  $V$  of variables and set  $T$  of terminals consist of single characters. Variables in  $V$  consist of upper-case alphabetic characters. Terminals in  $T$  consist of lower-case alphabetic characters and special characters like  $\{ = + \text{ etc.}$  The set  $P$  of production rules do not include  $\epsilon$ -productions, but otherwise can include any valid *s-grammar* rules.

2. Program Details

The *s-grammars* and strings to be parsed must be entered via a text file *TestData.txt*. You can specify the format for the text file for yourselves.

The program should then parse the entered strings, i.e. check whether they belong to the language that the grammar defines, by performing a leftmost derivation on them. The program must also have a facility to display each step of the leftmost derivation as it is occurring.

You might find it useful to check out *JFlap's Grammar* option which has quite a nice GUI interface, though you don't have to provide such an interface yourself. Any *s-grammar* (like grammars  $G_1$  and  $G_2$  in section 3 below) is actually usable in *JFLap* as an LL(1) grammar, since *s-grammars* are special cases of LL(1) grammars.

3. Testing

You must test your code on the following two *s-grammars*, though of course you can (and should) create or find others as well on which you can test your code. Note that all productions in these grammars follow the requirements for *s-grammars* in that

- there is a single variable on the left hand side of a production.
- the right hand side of a production starts with a single terminal and is followed by 0 or more variables.
- if there are two or more productions with the same variable on the left hand side, the right hand of each of these productions starts with a different terminal.

$G_1$  is a *s-grammar*

$$G_1 = ( \{ S, A, B \}, \{ a, b \}, S, P )$$

for the language  $L = \{a^n b^{n+1} : n \geq 1\}$ , with the elements of P being the productions

S	→	aAB
A	→	aAB   b
B	→	b

$G_2$  is a *s-grammar*

$$G_2 = ( \{ S, R, L, C, A, Y, P, Z, X, T, D \}, \{ \{ \}, x, =, y, +, z, i, t, e, d \}, S, P )$$

for a fragment of a programming language, with the elements of P being the productions

S	→	{LR
R	→	}
L	→	xAYPZ   iXTLC
C	→	d   eLD
A	→	=
Y	→	y
P	→	+
Z	→	z
X	→	x
T	→	t
D	→	d

A program fragment takes the form

$\{ Stmt \}$

where *Stmt* can have the one of the forms

- (i)  $x = y + z$
- (ii) **if**  $x$  **then** *Stmt* **endif**
- (iii) **if**  $x$  **then** *Stmt* **else** *Stmt* **endif**

E.g a program fragment might be

```
{ if x then
    x = y + z
else
    if x then
        x = y + z
    endif
endif }
```

Obviously, since both variables and terminals are specified to consist of single characters only, any multi-character variables or terminals like *Stmt* or **endif** must be represented by single characters. The following scheme is used

word	symbol
------	--------

<i>Stmt</i>	L
if	i
then	t
else	e
endif	d

The above program fragment is then represented by the string

```
{ixtx=y+zeixtx=y+zdd}
```

You could decide in your implementation that whitespace is not significant within strings to be parsed. If that were the case, then the string might have the (slightly) more user-friendly format

```
{ i x t x=y+z e i x t x=y+z d d }
```

#### 4. Data Structures and Algorithms

In any leftmost derivation, the essential requirement is to replace the leftmost variable in the current sentential form with the right hand side of a production with that variable on the left hand side. In an *s-grammar*, the choice of which production to use is made very simple. If *a* is the next symbol in the input string being parsed, and *A* is the leftmost variable in the current sentential form, we have to find a production with *A* on the left hand side whose right hand side starts with *a*. By the nature of an *s-grammar* there can be only one such production. If there is no such production the string cannot be a member of the language that the grammar defines.

Design decisions you should consider include

- how sentential forms should be represented so that it is easy to find and replace the leftmost variable in the sentential form, perhaps a stack.
- how productions should be represented so that it is easy to find the production that is to be used in the replacement of the leftmost variable in the sentential form. The restriction that variables must consist of uppercase single characters means there can be no more than 26 variables here, but in a real grammar there may be dozens or even hundreds of variables. A method of representing and retrieving productions efficiently is therefore important.

#### 5. Submission

This is a group project, for which the class has been divided into 28 groups, A -Z, AA and BB. The groups are the same as those for *Assignment 1*, and the same conditions apply.

The completed assignment is to be submitted as a single exported *Eclipse* .zip project file (not .tar or .rar or any other archive format you can think of!), using *Eclipse's* *Export/Archive File* facility. You must ensure that your project file can be imported into *Eclipse* and run without any changes needing to be made to it. The markers don't have the time to do this for you! The name of your project file must be *GroupName*

*\_Assignment\_2.zip*, e.g. *F\_Assignment\_2.zip*. The main method through which the assignment is run must be in a class called *RunAssignment*.

You will need to submit the properly documented source code of your classes, together with a record of the testing you undertook to show the correctness of your code in the form of the input text file mentioned in section 1, e.g. *TestData.txt*. In addition, there should be a brief statement stating for each group member what he or she was responsible for. These must all be in the *Eclipse* project file. Submit through *Moodle* by clicking on the *Assignment 2* topic on the front page and uploading your single zipped project file. You can resubmit as often as you like, but only the last submission is kept.

The due date is 17 December at 17h00.