**AUTOMAT**

**MACHINE PROBLEMS 1 and 2**

**1st Term 2015-16**

**Released: November 27, 2015**

General Instructions:

* Submit through google classroom your text/saved files.
* A softcopy of this instruction, as well as the Kara and JFLAP software, is accessible in the AUTOMAT google classroom.
* Put all your files in a ZIP file. For Kara, submit the **world files** and the corresponding **code files**. For Push Down Automata and Turing simulator, submit the saved JFLAP file. Make sure that your files contains the saved data. If there are problems with loading Kara or saving the files, download a working version: <http://www.swisseduc.ch/informatik/karatojava/download.html>
* **Deadline of submission** The MP is very doable within 1 week. Deadline is on December 4, 2015 before the start of the AUTOMAT Finals schedule (11:30AM, December 4, 2015).
* **This is done by groups (maximum of 3 members). Use the same grouping as your case study group.**

1. **(Part of MP 1) Using the Kara Software**

**(Download and use: allkara-en.jar)**

* 1. **Kara – programming with state machines**
     1. Recognition of floating point numbers – kara twist
        1. Input/Starting Configuration



* + - 1. Output/Final Configuration



* + - 1. Description

Refer to the table/legend below:

|  |  |
| --- | --- |
|  | Lady Bug is the “Tape Head” of the Finite State Machine |
| Treated as ‘E’ or ‘e’ |
| Treated as any 1 digit |
| Treated as the decimal point ‘.’ |

Construct the kara program such that given a floating point number (without the signs), kara would be able to accept or reject the series of symbols if it reaches one square after the end of the series. Otherwise, it’s a reject if kara doesn’t achieve the final configuration as show above.

Examples of floating point numbers would be:

1

123.45

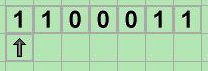
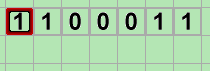
123.45E67

123E45

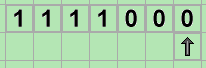
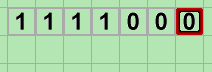
* 1. **(Part of MP 2) TuringKara – 2d Turing machines illustrated**

**(Download and use: allkara-en.jar or turingkara-en.jar)**

* + 1. **Concatenation of strings**
       1. Input/Initial Configuration

 -or- 

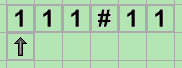
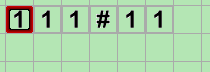
* + - 1. Output/Final Configuration

 -or- 

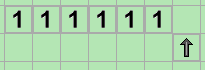
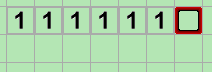
* + - 1. Description

Given a series of 1’s followed by a series of at least 1 0’s followed once again by a series of 1’s, your program would simply concatenate the two series of 1’s

* + 1. **Multiplication**
       1. Input Initial Configuration

 -or- 

* + - 1. Output Final Configuration

 -or- 

* + - 1. Description

Given a series of 1’s followed by # and then another series of 1’s, define the Turing machine that would multiply the length of the first string of 1’s and the length of the second string of 1’s and output the result by writing 1’s that represents the value of the answer. In other words, given 1n#1m, output 1n\*m.

1. **(Part of MP 1) Using the Pushdown Automata in JFLAP**

Construct/define your automata for the following problems

* 1. **2.1 Balancing of Parentheses**

Description: Accept the string if it is a valid sequence of balanced series of parentheses.

**Σ** = { **(**,  **)** }

Example Input: ( ( ( ( ) ) ) ( ) ( ) ( ) )

**2.2. Simple Binary Expressions**

Description: Accept the string if it is a valid sequence of an arithmetic expression

with binary numbers as operands.

**Σ** = { **1**, **0**, **+**, **/**, \*, **-** , **(**,  **)** }

Example Input : 1+0\*1/0+1+ ( 1+0 )

* 1. **3. (Part of MP 2) Using the Turing Machine in JFLAP**

Construct/define your code for the following problems

* 1. **3.1. Balancing of Parentheses** (easy)

Description: Halt and accept (end with a final state) the string if it is a valid sequence of

balanced series of parentheses.

Example Input: ( ( ( ( ) ) ) ( ) ( ) ( ) )

Γ = { **(**, **)**, **B** }

Output : BBBBBBBBBBB

Note: It will halt and go to an accepting/final state if the input string is balanced

* 1. **3.2. Evaluate Binary Expressions** (moderate)

Description: Output a 0 or 1 as a result of evaluating the whole binary expression

where + is evaluated as an OR and \* is evaluated as an AND.

There is no need for precedence (e.g MDAS), so operators are simply evaluated

from left to right, EXCEPT when a sub-expression is inside a parenthesis.

Example Input: 1+0\*1\*0+1+ ( 1+0 )

Preliminary evaluation: 1 + ( 1 )

Final result: 1

Γ = { **1**, **0**, **+**, **\*** , **(**, **)** , **B** } (Note: you may add more symbols as you see fit)

Output: 1

Note: outputs 1 or 0 as a result of evaluating the whole binary expression.

* 1. **3.3. Arithmetic Series** (moderate)

Description: Given a series of 1’s as input, the length of the string is interpreted as n.

It outputs a series of 1’s whose total length is equivalent to .

Example input: 1111

Note: (n = 4, from 4 1’s)

Γ = { **1** } (Note: you may add more symbols as you see fit)

Output: 1111111111

Note: (length of string = 1+2+3+4 = 10 1’s)