Business Analysis

Finite automaton equivalent to given regular expression

Patrycja Skorupa

Document metric			
Project:	Finite automaton equivalent to given regular expression	**Company:**	WUT
Document name:	Business Analysis		
Topics:	Introduction, Requirements, Computation,Summary		
Author:	Patrycja Skorupa		
File:	skorupap_buisness_analysis_individual_project.docx		
Version no:	1.3 **Status:** Final	**Opening date:**	**22.02.2013**
Summary:	Requirements of the Finite automaton equivalent to given regular expression		
Authorized by:	dr L. Stapp	**Last modification date:**	**15.03.2013**

History of changes			
Version	**Date**	**Author**	**Description**
0.5	22.02.2013	Patrycja Skorupa	Introduction, Requirements
0.7	25.02.2013	Patrycja Skorupa	Requirements, GUI, Summary
1.0	27.02.2013	Patrycja Skorupa	Final Correction
1.1	09.03.2013	Patrycja Skorupa	Removing GUI, Little Corrections
1.2	14.03.2013	Patrycja Skorupa	Extending requirements, adding definition f of alphabet
1.3	15.03.2013	Patrycja Skorupa	Partition requirements into new requirements and

			~~compuation~~computation

Table of contents

Introduction

The aim of this project is to design an application, which creates finite automaton which accepts the regular language. This language is ~~created~~ denoted by regular expressions. This application should verify if the expression is regular and then build the automaton based on this regular expression. The program will be user friendly with a clear to read interface.

Requirements

The functionality of this program is to accept regular language and build automaton based on this language. The user will insert a string of characters as an input to the program and an application will validate if the input is an regular expression.

An alphabet is a standard set of letters (basic written symbols or graphemes)

Regular expression over the alphabet Σ is defined as a set of strings, consisting of symbols $\varnothing, \epsilon, +, ^*,), ($ and symbols a_i from the alphabet Σ in the following way:

1. $\varnothing, \epsilon$ (empty word) is an regular expression
2. All symbols $a_i \in \Sigma$ are regular expressions;
3. If e_1, e_2 are regular expressions, then the following are as well:
   - e_1^* (Kleene's star)
   - $e_1 e_2$ (concatenation)

- $e_1 + e_2$ (sum)
- (e_1) (grouping)

4. All regular expressions are in the form described in points 1 – 3.

Regular languages are those generated by regular expressions and only those.

Let L(w) describes language defined by w. Then:

- $L(\epsilon) = \{\epsilon\}$ (set containing only empty word)
- $L(\varnothing) = \varnothing$ (empty set)
- $L(a) = \{a\}$ for any a from the alphabet

However for the construction of the expressions we use three symbols:

- $L(w + v) = L(w) \cup L(v)$ (union)
- $L(w^*) = (L(w))^*$ (Kleene's star)
- $L(wv) = \{xy : x \in L(w) \wedge y \in L(v)\}$ (concatenation of languages)

Kleene's star operation is the strongest one, then concatenation, and at the end the union.

Computation

The method of the computation is called left quotient.

If L_1 and L_2 are formal languages, then the left quotient of L_1 with L_2 is the language consisting of strings *w* such that *xw* is in L_2 for some string *x* in L_1. In symbols, we write:

$$L_1 \backslash L_2 = \{w \mid \exists x((x \in L_1) \wedge (xw \in L_2))\}$$

You can regard the left quotient as the set of postfixes that complete words from L_1, such that the resulting word is in L_2.

The initial state is the input inserted by the user. We take the quotients from deriving the initial state. In that process we use all possible language characters from entered regular expression. As the divisor we will take one character each time. The results of the process will be considered as another state. For all new quotients, we repeat the process until we cannot find any new states. Each time, the user will have to decide whether the new founded state is the same as the one already generated. If the user decides that it is the same, then he/she must select from the list to which is it equivalent to. Another task of the user will be to decide if the state can generate an empty word i.e. if it is a finite state.

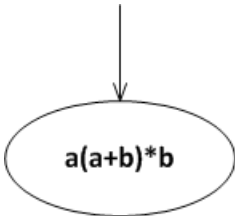
If the word belongs to the regular language, the user will get a message that the word is accepted by the automaton. Otherwise the user receive message that word is not in the language.

Here is the example of process of solving the problem by the program. The input of this problem is as follows:

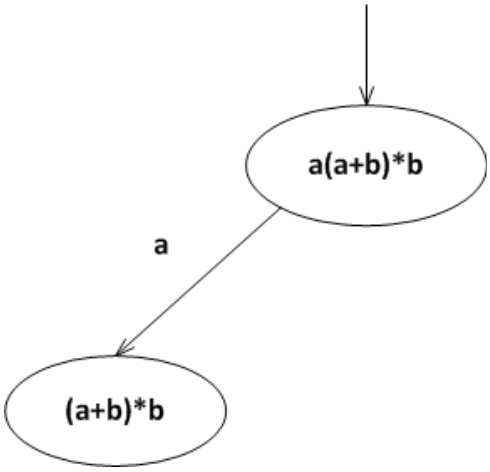
a(a+b)*b

And here are the steps of how it will be solved:

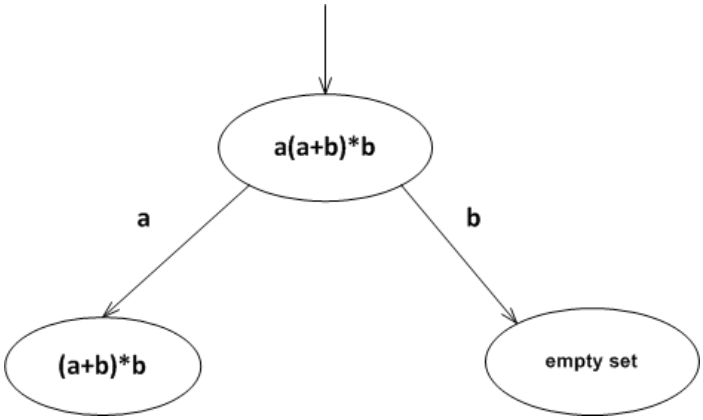
1. a(a+b)*b



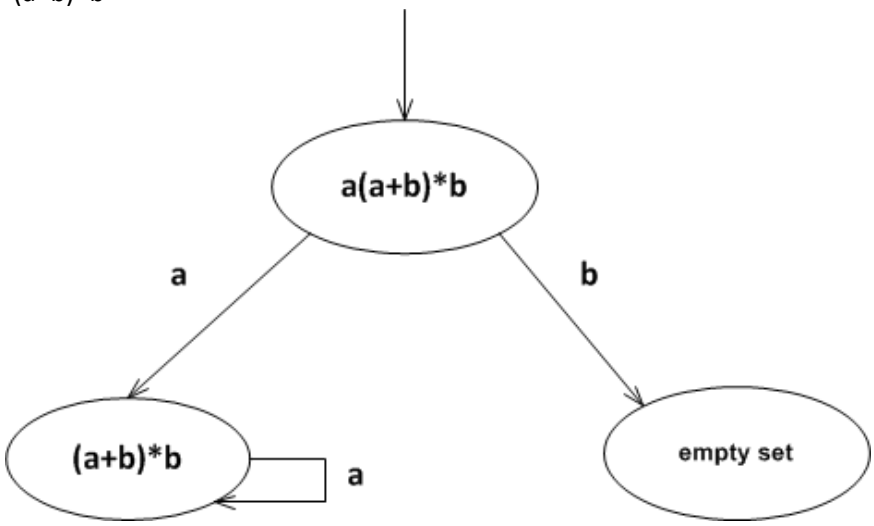
2. a(a+b)*b\a=(a+b)*b



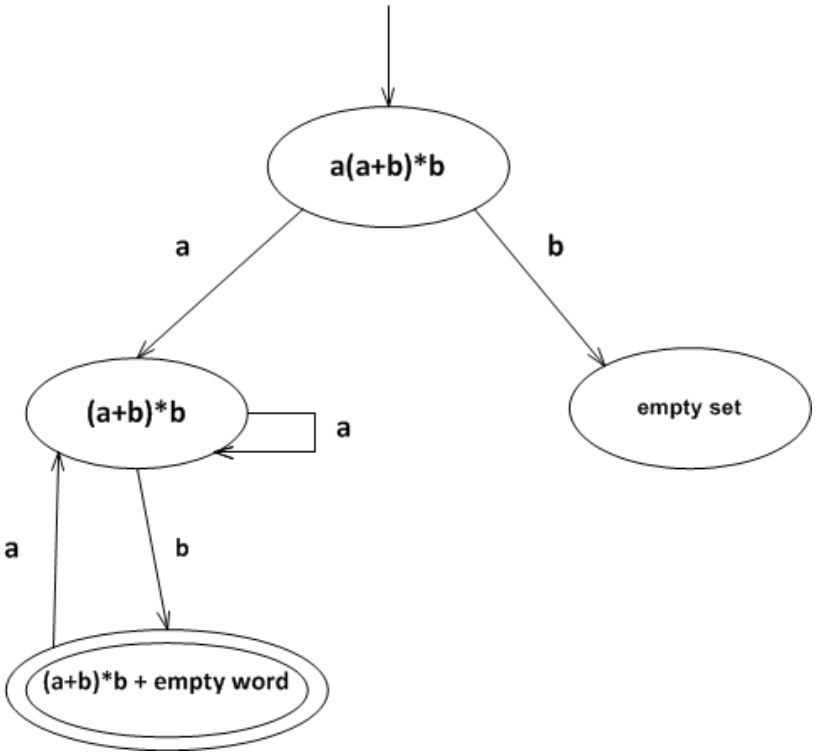
3. a(a+b)*b\b=empty set



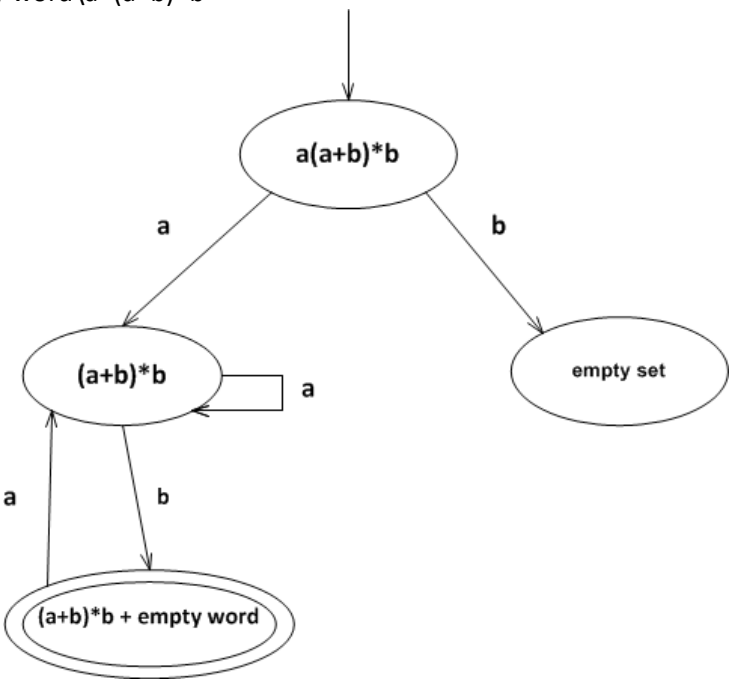
4. (a+b)*b\a=(a+b)*b



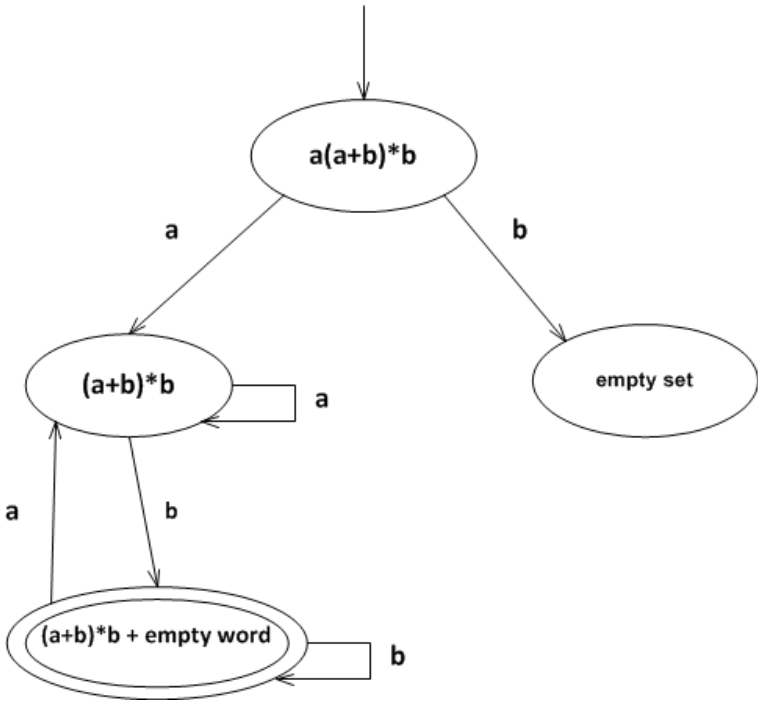
5. (a+b)*b\b=(a+b)*b + empty word



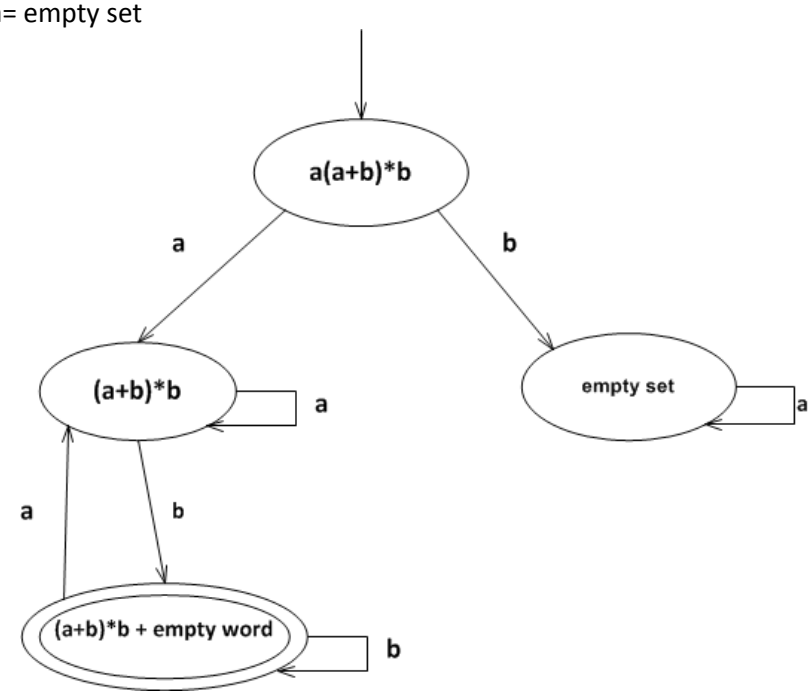
6. (a+b)*b + empty word\a=(a+b)*b



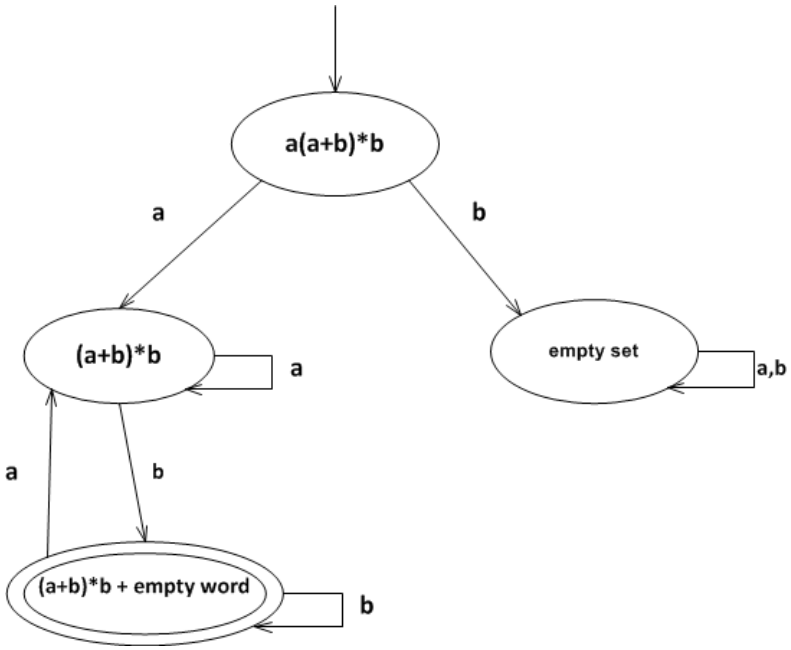
7. (a+b)*b + empty word\b=(a+b)*b + empty word



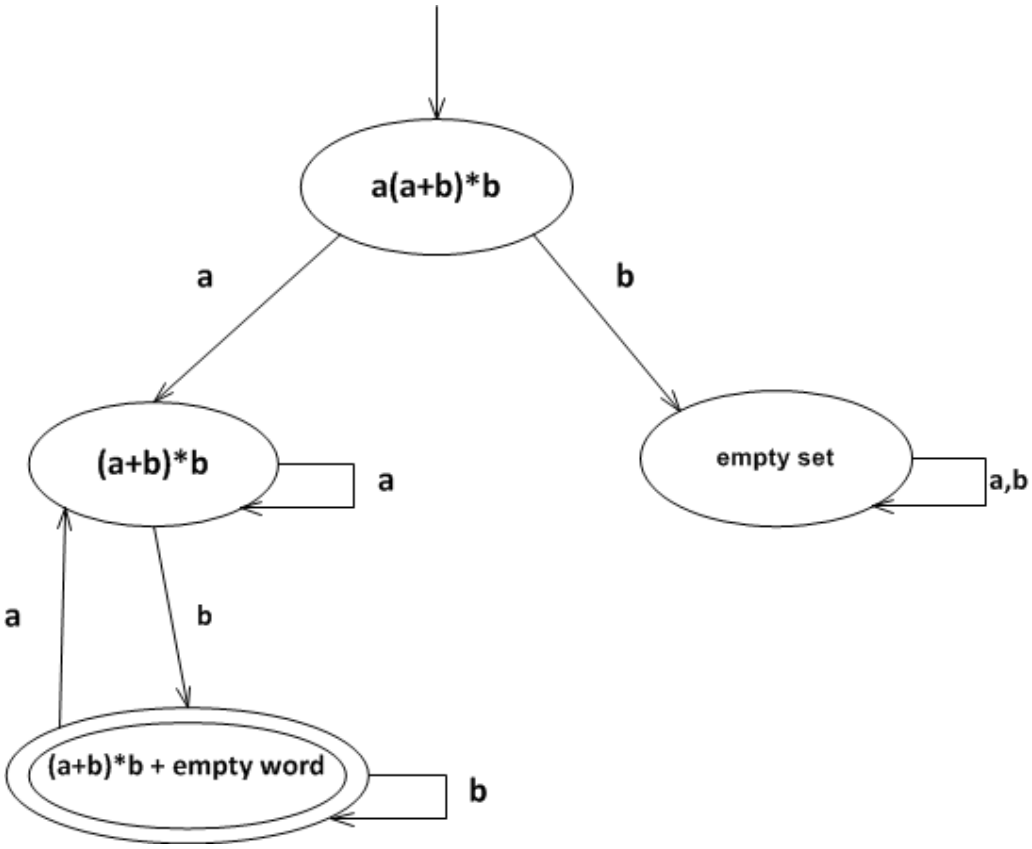
8. empty set\a= empty set



9. empty set\b= empty set



and the final automaton (the solution) will look like this:



Second requirement

The second requirement of the application is possibility to check, how the automata build accordind to algorithm described in previous section works. The user can introduce a word over given alphabet and then observe – in one of two described below – modes, how the automata works,

These two modes are as follows:

- step by step mode – the user observe the steps of computation
- only result – the user observe only the final effect of computation

In both cases at the end tha appropriate message is generated (“accepted” or “non accepted”)

Summary

The aim of this program is to help users create finite automata based on inputted regular language by the user. Also, checking the inserted language will be useful to many users since not all end-users know what is a regular language. This application will be simple to use for the user

and it will be clear to read. The answers of the problem will not be too complicated and they will be easy to understand.