

AutoGram

Functional specifications document

Jakub Drahoš, Peter Matta, Dominika Králiková

2018

Table of contents

1	General properties	1
1.1	Allowed characters	1
1.2	Input	1
1.2.1	Finite automaton	1
1.2.2	Pushdown automaton	2
1.2.3	Grammar	2
1.2.4	Regular expression	3
1.3	Output	4
1.3.1	Automaton	4
1.3.2	Grammar	4
1.3.3	Regular expression	4
1.4	Text input/output	5
2	Visualization of automata and grammars	6
2.1	Automata	6
2.1.1	Finite automaton	6
2.1.2	Pushdown automaton	6
2.2	Grammar	7
2.2.1	Regular and context-free grammar	7
3	Comparison	8
3.1	Comparing finite automata, regular grammars and regular expressions . . .	8
4	Transformations	9
4.1	Transforming finite automata, regular grammars and regular expressions .	9
5	Algorithms	10
5.1	General course of algorithm run	10
5.2	Elimination of \mathcal{E} -transitions	10
5.3	Determinization	10
5.4	Minimization	10
5.5	Derivatives of regular expressions	10
5.6	Context-free grammar reduction	11
5.7	Elimination of \mathcal{E} -rules	11
5.8	Elimination of unit rules	11
5.9	Chomsky normal form	11
5.10	Removing left recursion	11
5.11	Cocke-Younger-Kasami algorithm	12

1 General properties

1.1 Allowed characters

Users can choose arbitrary string of allowed characters for naming non-terminal symbols of grammars and automata states. Terminal symbols and an alphabet of automaton or regular expression consist of one and only one allowed character. Allowed characters are: all *alphanumeric symbols*, *underscore* and *dash*.

Whitespace characters will be ignored (*a b* is considered as *ab*).

1.2 Input

1.2.1 Finite automaton

Finite automaton (from now on referred as **FA**) is entered via input table - rows representing states and columns representing alphabet of the automaton. Users can add both rows and columns. There's also a possibility of adding special column for ϵ -transitions.

After clicking on any state in the leftmost column, three flags will appear: *I* for initial state, *F* for final state and *X* for deleting the whole row. FA can be defined via table so there's no need to enter the whole quintuplet (it's not an option anyway). All user defined automata are considered non-deterministic FA (**NFA**) by default.

	δ	a	b	
	\rightarrow 0	0	0, 1	
	1	2	2	
	2	3	3	
	\leftarrow 3			
<div><div>s</div><div>F</div><div>X</div></div>	\leftarrow <input type="text"/>			

Figure 1: FA input

Following situations can occur:

- Result of transition function for given state and symbol is set of states. In that case the result states will be written in one cell of the table, separated by commas. They won't be in parentheses.
- State resulting from composition of multiple states (e.g. determinization) will be labeled as concatenation of these in ascending order.

- An empty cell represents undefined state transition for a given symbol.

Before sending a request to the server for automaton processing, validation takes place and possibly asks user for correct input. Validating:

- Transitions to existing states only
- Existence of initial state
- Existence of at least one state
- Disjointed sets of states and alphabet symbols
- Usage of valid symbols (see [Allowed characters](#))

1.2.2 Pushdown automaton

Pushdown automaton (**PDA**) is entered via table with two columns representing transition function. First column as states of the automaton, second column as transitions in format: $(\langle \text{entry symbol} \rangle, \langle \text{pop from stack} \rangle / \langle \text{push to stack} \rangle, \langle \text{target state} \rangle)$, separated by pipes „|“. E.g. $(a, 1/\varepsilon, Q) \mid (\varepsilon, Z_0/\varepsilon, F)$

Next to the table is button for entering ε symbol. Input table is also supplemented with form specifying initial stack symbol, top of the pushdown store (either left or right) and acceptance mode (either accepting by final states or by empty stack). When choosing accepting by final states, the same flags as mentioned in [Finite automaton](#) will occur after clicking on any state.

Validation and special situations are dealt with the same way as with FA.

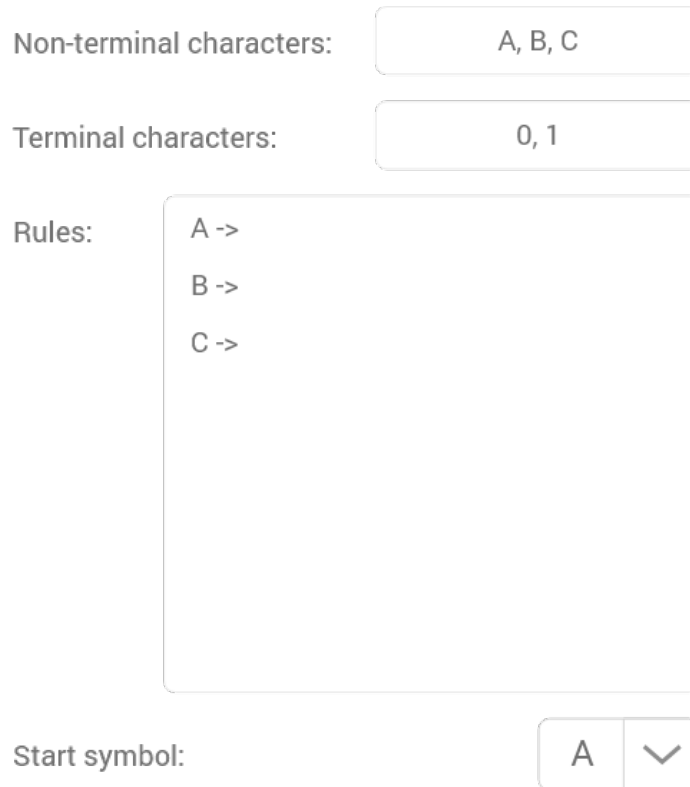
1.2.3 Grammar

Application accepts only regular (**RG**) and context-free grammars (**CFG**). Context-sensitive or unrestricted grammars are not supported. RG and CFG are entered via prepared forms allowing to define the whole quaternion (**N**, **Σ** , **P**, **S**).

- **N** for non-terminal symbols, separated by commas. Non-terminals consisting of multiple characters must be surrounded by $\langle \rangle$, e.g. $\langle ABC \rangle$ (Where $\langle \rangle$ are used for escaping and are not part of valid naming syntax).
- **Σ** for terminal symbols, separated by commas
- **P** for production rules. Application prepares the form based on entered non-terminals. For each non-terminal symbol there will be pre-filled beginning of the given rule (e.g. $S \rightarrow$). Set of rules for one non-terminal is entered in format: $\langle \text{rule} \rangle \mid \langle \text{rule} \rangle \mid \langle \text{rule} \rangle$. E.g. $S \rightarrow a \mid aA \mid S$.

Button for entering ε is located next to the input field.

- **S** for start symbol, which is selected from drop-down menu based on entered non-terminals.



Non-terminal characters: A, B, C

Terminal characters: 0, 1

Rules:

- A ->
- B ->
- C ->

Start symbol: A

Figure 2: Grammar input

Before sending a request to the server for grammar processing, validation takes place and possibly asks user for correct input. Validating:

- Only terminals and non-terminals defined in relevant sets can be used in production rules.
- Valid syntax format of the rules.
- Usage of allowed symbols for naming terminals and non-terminals (see [Allowed characters](#)).
- Regularity of given grammar, if needed.

1.2.4 Regular expression

Regular expression (**RE**) is entered via text field. Concatenation can be written down using dot (e.g. a.b) or just naturally as string (e.g. ab). Alteration is represented by $+$ symbol, Kleene star by $*$. Next to the input field lies buttons facilitating entering special symbols like ε and \emptyset .

Before sending a request to the server for expression processing, validation takes place and possibly asks user for correct input. Validating:

- Valid syntax of expression.
- Usage of allowed symbols (see [Allowed characters](#)).

ϵ

\emptyset

Expression:

$(a + bc^*)^* + b(cac)^* + \epsilon$

Figure 3: Regular expression input

1.3 Output

Unless said differently, output of every algorithm is uniform and always displayed on new page. Otherwise it will be specified at particular case.

1.3.1 Automaton

Automaton is generally displayed as table representing transition function. In case of PDA the table is supplemented with information about initial stack symbol, top of the pushdown store and string acceptance mode.

It is possible to display the automaton also as a digram (hovering over an eye icon)



Figure 4: Button for displaying automaton diagram

1.3.2 Grammar

Grammar is generally displayed as quaternion: set of non-terminals, set of terminals, set of production rules and start symbol.

1.3.3 Regular expression

Regular expression is always displayed as a string.

1.4 Text input/output

Application is able to import input from or export output to strictly defined text format. All inputs can be replaced with this text format and loaded in as a plain text file and vice versa (meaning that all inputs entered directly in applications forms can be exported as text file). This feature offers not only quick and simple solution for chaining different algorithms, but also the possibility to save some important input/output for later.

2 Visualization of automata and grammars

2.1 Automata

Visualization of FA and PDA shares the same initial page. User can choose from drop-down menu which type he/she wants and the corresponding table will be shown.

2.1.1 Finite automaton

After entering input and successful validation as mentioned above (see [Finite automaton](#)), user is redirected to the visualization page, where he/she enters a sentence to be consumed by the automaton.

Visualization is taking place in form of state diagram. As the automaton consumes each of input string characters, all possible states are highlighted. While reading next character from input, all edges that can be used are highlighted. Also each currently consumed character is highlighted in the input sentence.

Visualization can go both directions, forward and backwards. User can also go back to the input and enter new automaton at any time.

Picture of diagram is in SVG format.

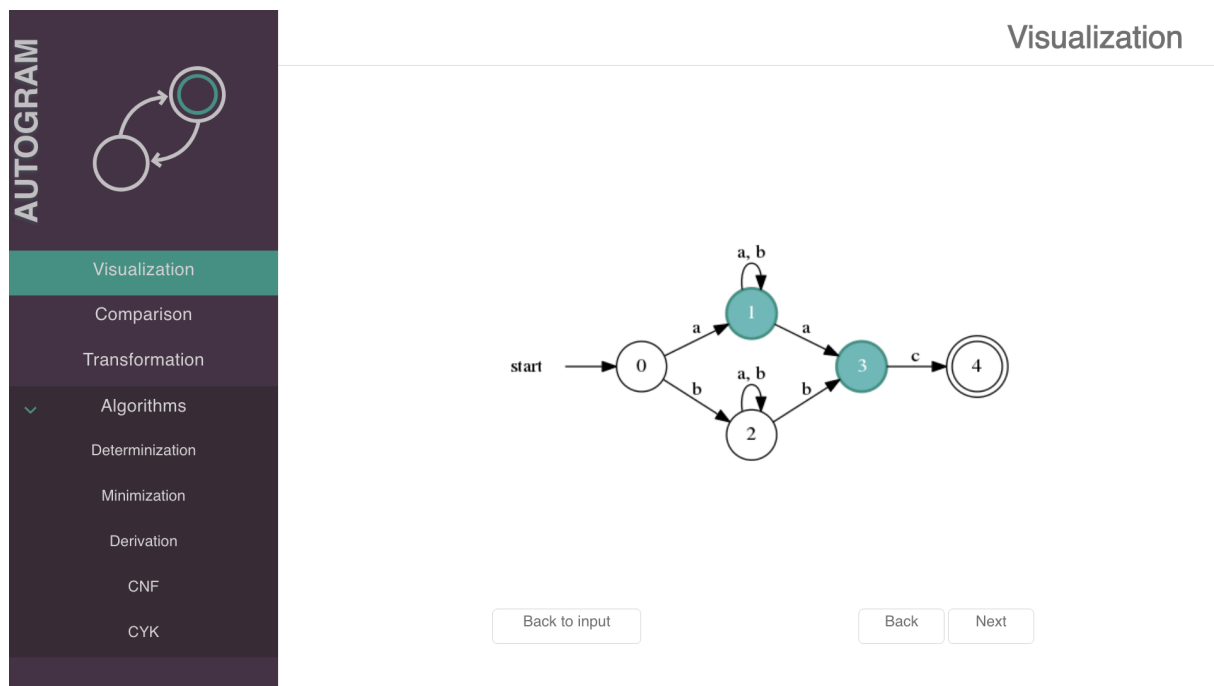


Figure 5: FA visualization page

2.1.2 Pushdown automaton

PDA visualization uses the same paradigm as [FA visualization](#) mentioned above. The only difference is in showing current state of stacks of all possible states - the current stack is shown on mouse hover/click at given state.

2.2 Grammar

2.2.1 Regular and context-free grammar

Users can create any sentential form that is possibly generated by given grammar via grammar visualization. After entering input and successful validation as mentioned above (see [Grammar](#)), user is redirected to the visualization page, where following is found:

- Production rules
- Text field for generating sentential forms
- Text field for derivation with sequence of rule numbers used for generating given sentential form

Users can choose, how should be a non-terminal symbol rewritten by clicking on it in sentential form and then clicking on one of the highlighted (these that can be currently used) production rules. After this the generated sentential form is update immediately.

Users can also go unlimited number of steps back, reset the whole process or go back to the input and edit it.

SCI-FI: displaying derivation tree.

3 Comparison

3.1 Comparing finite automata, regular grammars and regular expressions

Application can compare FA, RG and RE between each other and tell whether they describe the same formal language or not.

There are two input fields on the comparison screen. User can choose from drop-down menu, what is to be compared and according to this choice the correct input fields will be displayed. After filling the forms (when entering RG, there will be a check for its regularity), user runs the comparison and information about equality is shown.

Application does not display any kind of procedure or visualization.

The screenshot displays the 'Comparison' page of the AUTOGRAM application. On the left is a dark sidebar with the 'AUTOGRAM' logo and a navigation menu where 'Comparison' is highlighted. The main content area has a title 'Comparison' in the top right. It contains several input fields: 'Non-terminal characters' with the value 'A, B, C', 'Terminal characters' with '0, 1', and 'Rules' with a list of rules: 'A ->', 'B ->', and 'C ->'. Below these is a 'Start symbol' dropdown set to 'A'. A 'Grammar' dropdown is also present. A large, empty light-gray box is on the right side of the input area. At the bottom center is a 'Compare' button. Below the button, the text 'Regular expressions are equivalent' is shown next to a green circular checkmark icon. On the bottom right, there is a dropdown menu currently showing 'Automata', with 'Grammar' and 'Regular Expression' as other options.

Figure 6: Comparison page

4 Transformations

4.1 Transforming finite automata, regular grammars and regular expressions

Application can transform FA, RG and RE between each other. There's not an option to transform for example grammar to grammar though.

User can choose from drop-down menu, what is to be transformed (and to what) and according to this choice the correct input field will be displayed. After filling the forms (when entering RG, there will be a check for its regularity), user is redirected to the result page. There's an option to go back and edit the input of course.

Application does not display any kind of procedure or visualization.

5 Algorithms

5.1 General course of algorithm run

In general, after choosing an algorithm, input field is displayed. After validation the algorithm is run and user is redirected to a new page with displayed result. It's possible to go back and edit the input.

In some cases there's an option to go to a new page to see each step of the algorithm one by another (different step sizes, can go back and forth). From here user can also go back to the input page.

5.2 Elimination of ε -transitions

FA is expected as input of this algorithm. Input table contains a column for entering ε -transitions by default. It is not possible to display any procedure or visualization.

5.3 Determinization

FA without ε -transitions is expected as input of this algorithm. So there's no option in the input table to add column for ε -transitions. It is possible to display each step of determinization:

- There are two tables, one with original automaton and second gradually filled with deterministic automaton.
- For each row which is currently filled, all corresponding rows, that are being combined, are highlighted in former automaton.

5.4 Minimization


DFA is expected as input of this algorithm. There's a check whether the entered automaton is deterministic. It is possible to display each step of minimization:

- Long table, that is gradually filled, representing transition from former states to new groups, is displayed.
- First the unreachable and useless states are dismissed.
- Then the states are divided into groups according to transition function (at the beginning there are 2 groups - final and non-final states) and new transition table is filled (for every state separately; beginning with transitions to the first state). This step is repeated until the whole process of minimization is done.
- At the end a result transition function of minimized automaton is displayed (via table).

5.5 Derivatives of regular expressions

RE is expected as input of this algorithm. Next to the input field for RE, there's also another field for string to be used. It is possible to display each step of derivation only if we differentiate with respect of string, not only one character. Each step is representing a derivative with respect of one character from input string.

AUTOGRAM



Visualization

Comparison

Transformation

Algorithms

Determinization

Minimization

Derivation

CNF

CYK

Minimization

	δ	a	b	\sim	a	b
→	A	B	F	A		
	B	G	C	A		
←	C	A	I	C		
	E	H	F	A		
	F	I	G	A		
	G	G	E	A		
	H	G	I	A		
←	I	E	C	C		

Back to input

Back

Next

Figure 7: Minimization page

5.6 Context-free grammar reduction

CFG is expected as input of this algorithm. It is not possible to display any procedure or visualization.

5.7 Elimination of ε -rules

CFG is expected as input of this algorithm. It is not possible to display any procedure or visualization.

5.8 Elimination of unit rules

CFG is expected as input of this algorithm. It is not possible to display any procedure or visualization.

5.9 Chomsky normal form

CFG is expected as input of this algorithm. It's possible to display each step of conversion to CNF:

- Steps are representing respective sub-results of smaller algorithms contained inside this one. Namely *Context-free grammar reduction*, *Elimination of ε -rules*, *Elimination of unit rules* and conversion to CNF itself.

5.10 Removing left recursion

CFG is expected as input of this algorithm. It's possible to display each step of removing left recursion:

- Steps are representing respective sub-results of smaller algorithms contained inside this one. Namely *Context-free grammar reduction*, *Elimination of ε -rules*, *Elimination of unit rules* and removing the left recursion itself.

5.11 Cocke-Younger-Kasami algorithm

Grammar in CNF and string are expected as input of this algorithm. There's a check whether the entered grammar is in CNF or not. Filled table and decision, whether the entered string is generated by given grammar or not, is displayed as a result. It's possible to display each step of CYK:

- CYK table and given grammar is displayed.
- Each step represents filling one cell in the CYK table. Cells in which we search for non-terminal pairs are highlighted, as well as corresponding production rules of the grammar.