

COMP 442 / 6421

Compiler Design

Tutorial 1

Instructor:

Dr. Joey Paquet

paquet@cse.concordia.ca

TAs:

Haotao Lai

h_lai@encs.concordia.ca

Jashanjot Singh

s_jashan@cs.concordia.ca



An useful tool --- AtoCC

The learning environment can be of use in teaching abstract automata, formal languages, and some of its applications in compiler construction. From a teacher's perspective AtoCC aims to address a broad range of different learning activities forcing the students to actively interact with the subjects being taught.

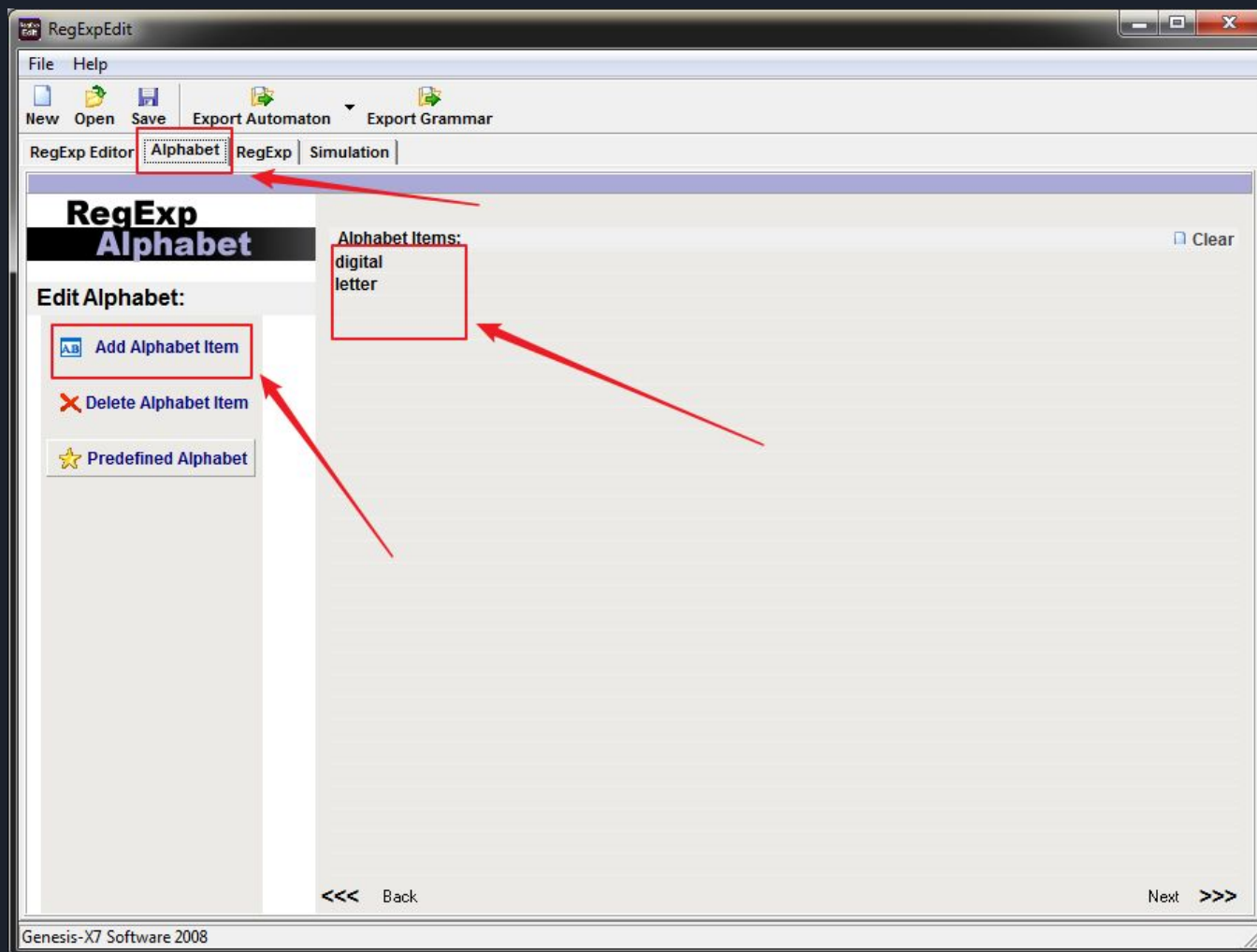
Note: We will need to use it for assignment 2 for grammar verification (will explain into detail when you receive assignment 2)

<http://www.atocc.de/cgi-bin/atocc/site.cgi?lang=en&site=main>



AtoCC --- RegExp Edit

It is a powerful tool that we can use to generate DFA from regular expression and validate your work. In the following slides you will find screenshots on how to use this tool in order to create a DFA from a regular expression that should conform to the lexical specification of the language.



RegExpEdit

File Help

New Open Save

Export Automaton Export Grammar

RegExp Editor | Alphabet | **RegExp** | Simulation

RegExp Editor

Enter RegExp here

letter(letter+digital)*

Use the formal notation for regular expressions.
Like: (a+b)*ab(a+b)* for L = {w | w contains ab} over {a,b}.

Minimized RegExp

letter(digital+letter)*

Compare RegExp with another

Compare

Transform to NEA

Generate NEA graph for your RegExp at the right. Show NEA

RegExp

NEA Graph Minimized NEA Graph

Start → q₀ → letter → q₁ → digital, letter → q₁

Hint: For ε you must write EPSILON in your RegExp.

εu = uε = u

ε* = ε

u+v = v+u

u+u = u

(u*)* = u*

u(v+w) = uv+uw

(uv)*u = u(vu)*

(u+v)* = (u* + v*)*

Genesis-X7 Software 2008

```
graph LR; Start((Start)) --> q0((q0)); q0 -- letter --> q1(((q1))); q1 -- "digital, letter" --> q1
```

RegExpEdit [C:\Users\h_lai\Desktop\t1.xml]

File Help

New Open Save Export Automaton Export Grammar

RegExp Editor Alphabet **RegExp** Simulation


RegExp Editor

Enter RegExp here


Use the formal notation for regular expressions.
Like: $(a+b)^*ab(a+b)^*$ for $L = \{w \mid w \text{ contains } ab\}$ over $\{a,b\}$.

Minimized RegExp

Compare RegExp with another

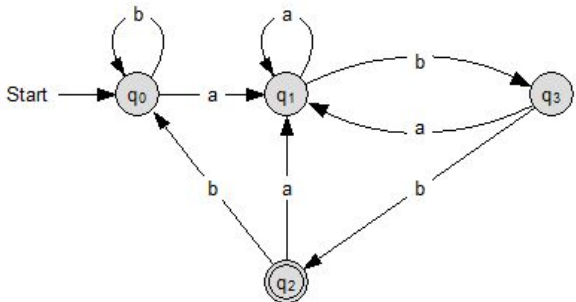
 Compare

Transform to NEA

Generate NEA graph for your RegExp at the right. 

RegExp

NEA Graph Minimized NEA Graph



```

graph LR
    Start((Start)) --> q0((q0))
    q0 -- a --> q1((q1))
    q1 -- b --> q0
    q1 -- a --> q2(((q2)))
    q2 -- b --> q1
    q2 -- b --> q3(((q3)))
    q3 -- a --> q2
    q3 -- b --> q1
  
```

Hint: For ϵ you must write **EPSILON** in your RegExp.

Genesis-X7 Software 2008

AutoEdit [C:\Users\h_jai\Desktop\t1.xml]


File Help


New Open Save Undo Redo Notepad Export Grammar Export RegExp Export Compiler

Automaton Editor | Type | Alphabet | Transition Table | Transition Graph | Publish | **Simulation**

AutoEdit Simulation


Edit Simulation settings:

Input: 

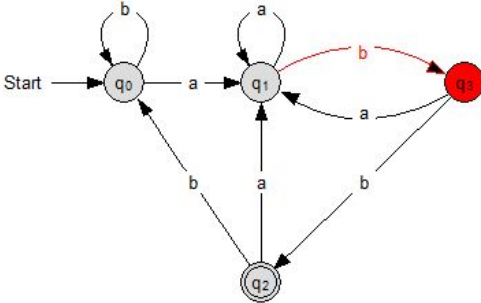
 **Start Simulation**

Speed:

☐ Single Step

 Export Scheme Code

Simulation:



Configuration sequence | Check multiple input |

	'0	'1	'2	'3	'4	'5	'6	'7
M0	q0	q0	q0	q0	q0	q1	q1	q3
	bbbbbaab	bbbbbaab	bbaab	baab	aab	ab	b	

Back

Genesis-X7 Software 2004 - 2008

AutoEdit [C:\Users\h_lai\Desktop\t1.xml]


File Help


New Open Save Undo Redo Notepad Export Grammar Export RegExp Export Compiler

Automaton Editor | Type | Alphabet | Transition Table | Transition Graph | Publish | **Simulation**

AutoEdit Simulation


Edit Simulation settings:

Input: 

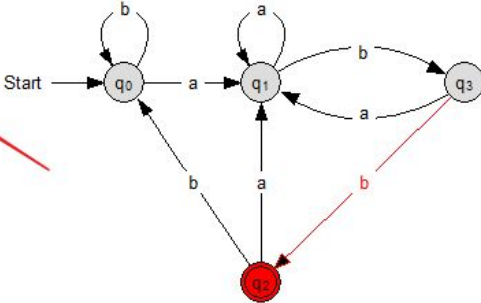
 Start Simulation

Speed:

☐ Single Step

 Export Scheme Code

Simulation:



Configuration sequence | Check multiple input

	'0	'1	'2	'3	'4	'5	'6	'7	'8
M0	q0	q0	q0	q0	q0	q1	q1	q3	q2
	bbbbbaabb	bbbaabb	bbaabb	baabb	aabb	abb	bb	b	

Back

Genesis-X7 Software 2004 - 2008

AutoEdit

Publish

Edit Publish settings:



Change Font



Export Graph ...



Export Definition ...



Export Transitions ...



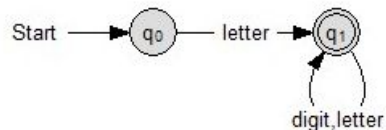
Export as HTML

HTML Publish:

Automaton

Type: NEA

Transition Graph:

Definition: $M = (\{q_0, q_1\}, \{\text{digit}, \text{letter}\}, \delta, q_0, \{q_1\})$

Transition Table:

δ	digit	letter
q_0	$\{\}$	$\{q_1\}$
q_1	$\{q_1\}$	$\{q_1\}$

Grammar: $G = (N, T, P, s)$ $G = (\{q_0, q_1\}, \{\text{digit}, \text{letter}\}, P, q_0)$ $P = \{$ $q_0 \rightarrow \text{letter } q_1 \mid \text{letter}$ $q_1 \rightarrow \text{digit } q_1 \mid \text{digit} \mid \text{letter } q_1 \mid \text{letter}$ $\}$ RegExp: `letter(digit+letter)*`



Implementation of lexical analyzer

Two ways to implement the lexical analyzer:

1. Table driven
2. Handwritten
 - Constructing a transition table by hand is not an easy job in this case at the least and
 - The handwritten approach will require you to be very careful considering all the possible situations.

Note: It is your choice to pick one of the methods to implement and your choice will not affect the prospective assignments. The output of the Scanner is the stream of tokens which will eventually be fed as as input to the next phase.



Continued

- You need to think about the ambiguity problem (you should already know that the solution is i.e. backtracking) and how to implement a backtracking mechanism.
- Also there is an advanced problem, how to make the lexical analyzer faster (read each character from the disk when you need a new one or there is some other ways to do it)?



Continued

Obviously, we need to use buffers instead of reading when necessary, and there are two ways provided by the “Dragon book”:

1. Buffer in Pair
2. Sentinel

You can refer to the book in order to know more about these ways and how they can be leveraged in your project work. Although, you can have your own way to implement that.



Want to know more?

- Two famous tools in this topic (Lex & Yacc):

<http://dinosaur.compilertools.net/>

- A well maintained list of resources relative to compiler study:

<http://aahour.com/awesome-compilers/>