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

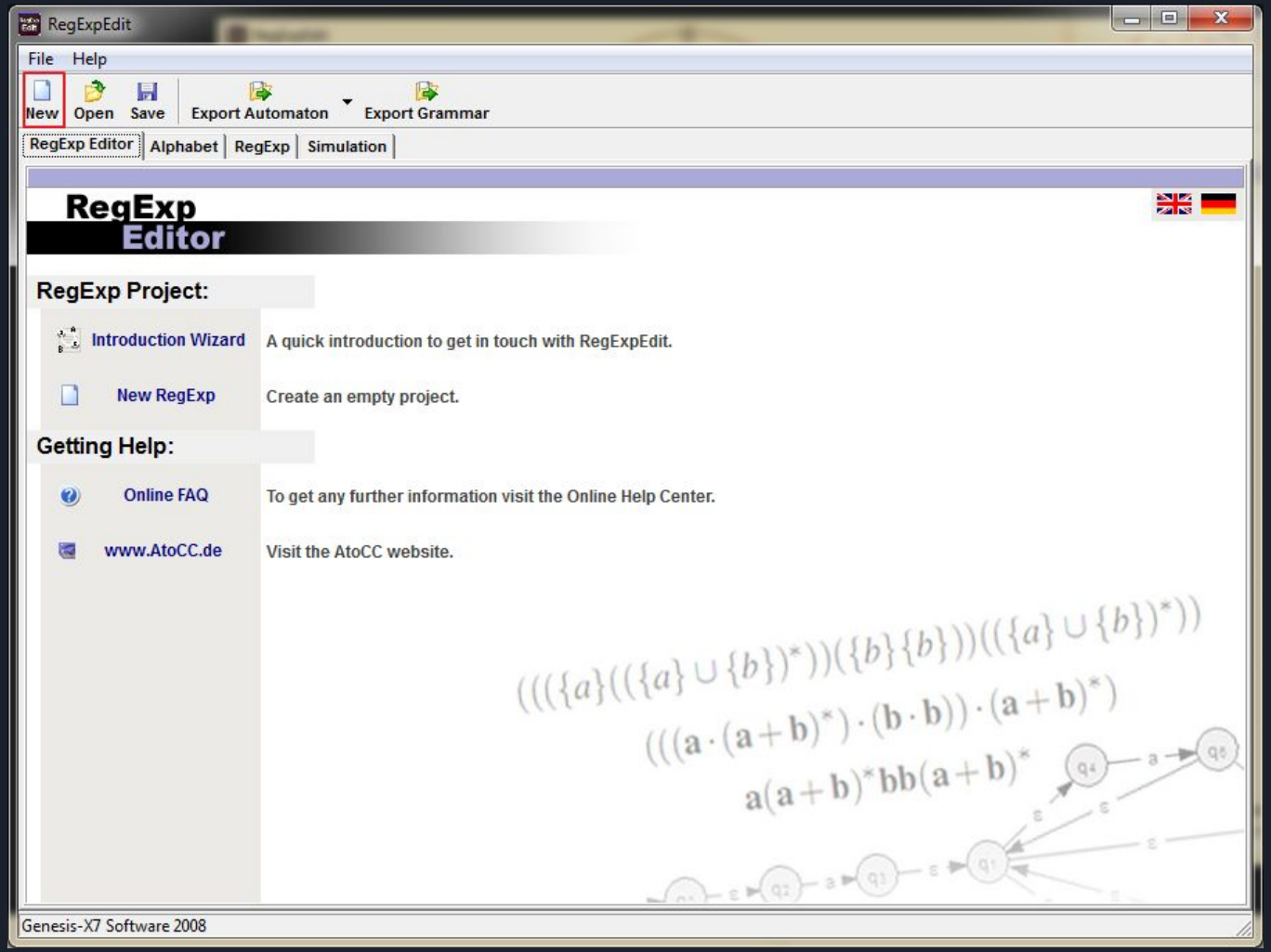
The learning environment can be of use in teaching abstract automata, formal languages, and some of its applications in compiler construction. From a teacher's perspective AtoCC aims to address a broad range of different learning activities forcing the students to actively interact with the subjects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

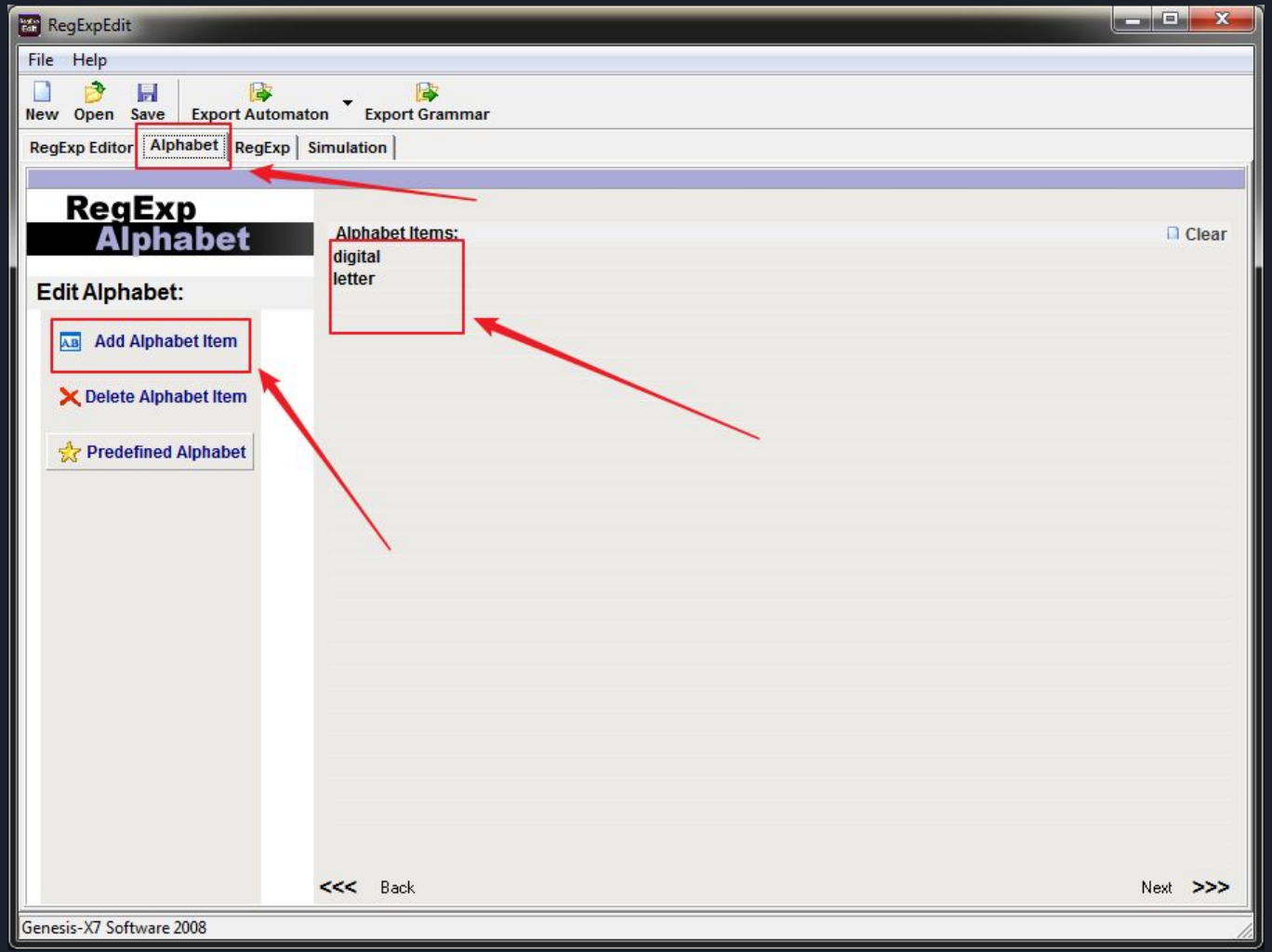
Note: We will need to use it for assignment 2 for grammar verification (will explain into detail when you receive assignment 2)

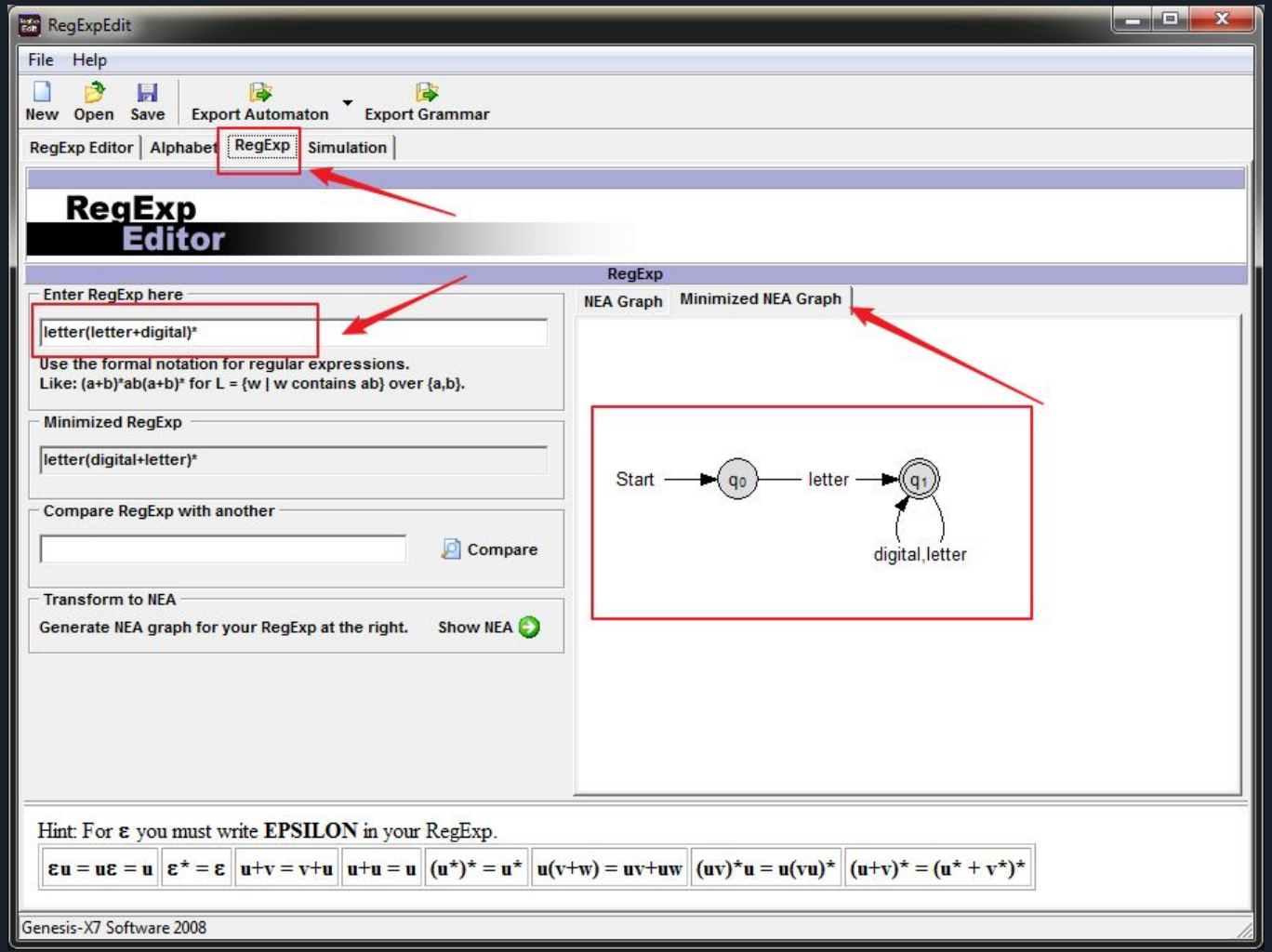
http://www.atocc.de/cgi-bin/atocc/site.cgi?lang=en&site=main

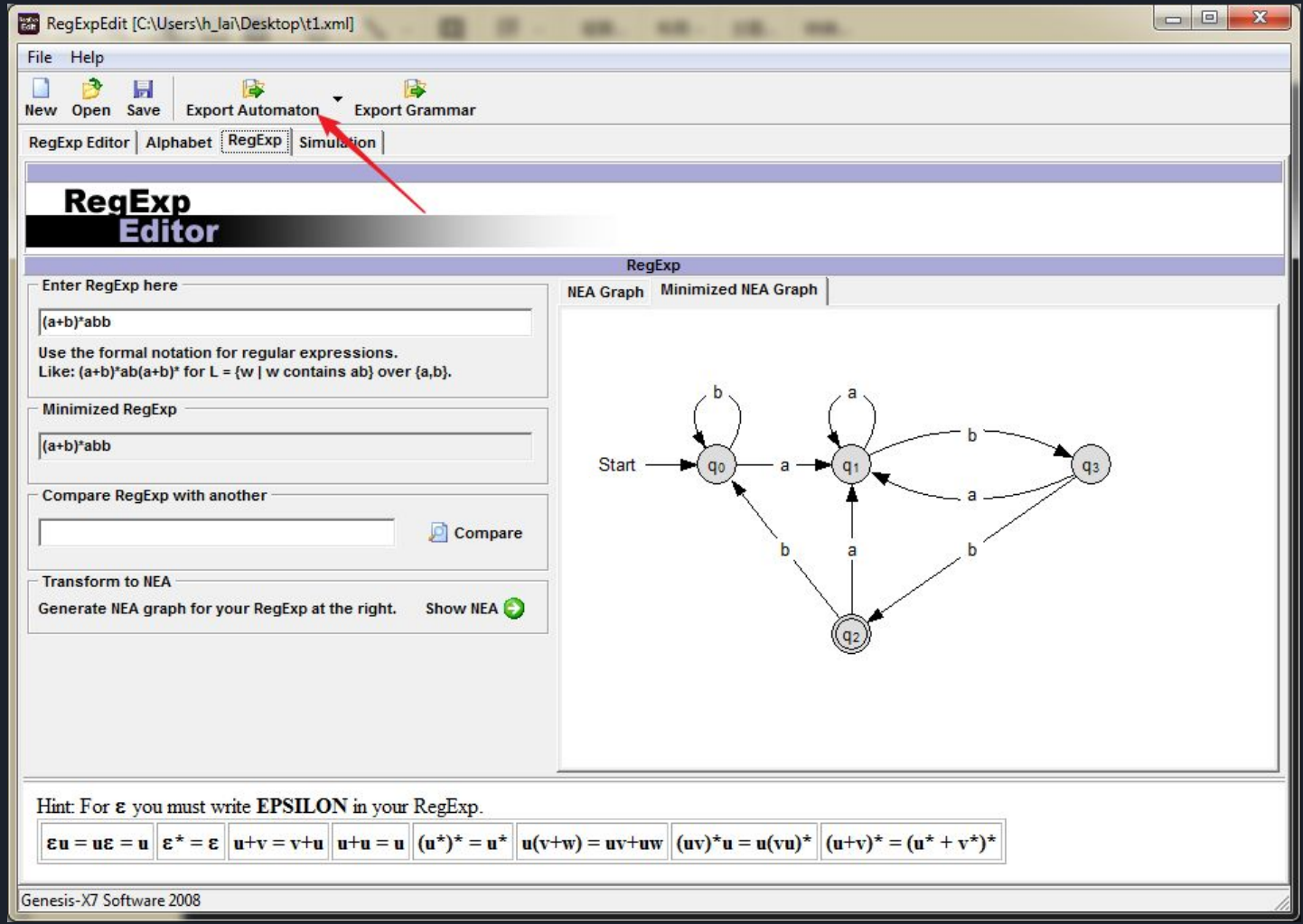
AtoCC --- RegExp Edit

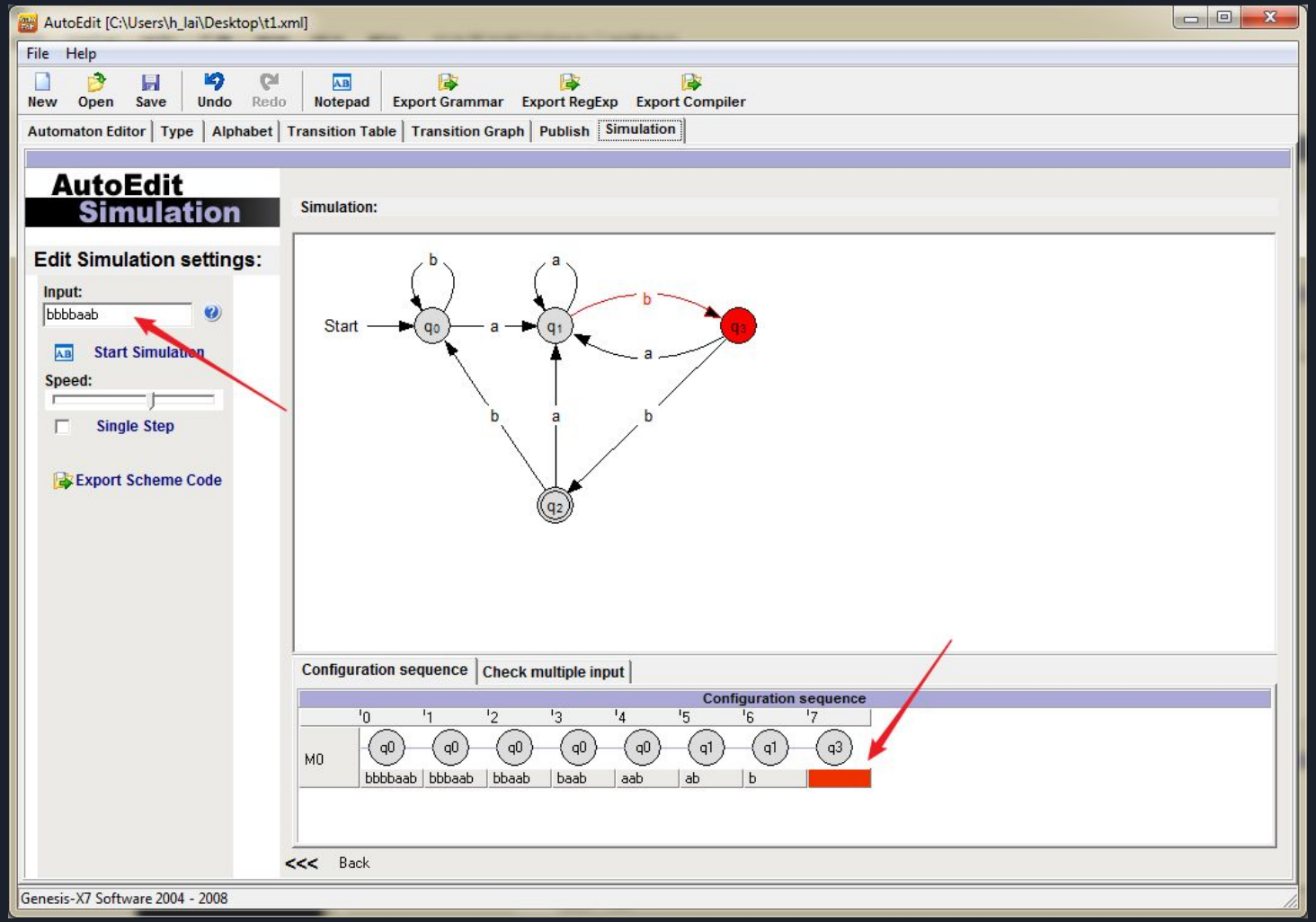
It is a powerful tool that we can use to generate DFA from regular expression and validate your work. In the following slides you will find screenshots on how to use this tool in order to create a DFA from a regular expression that should conform to the lexical specification of the language.

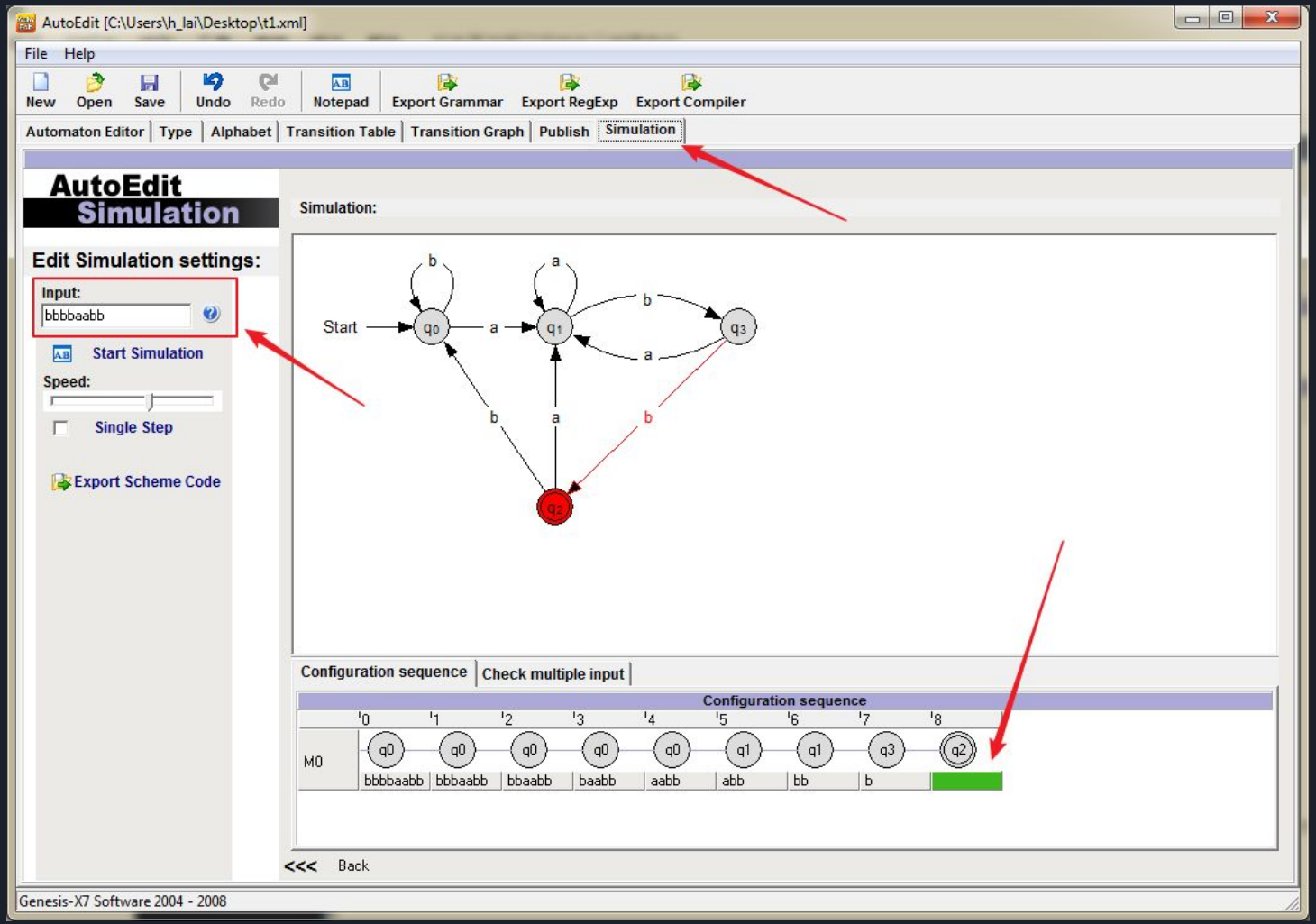

RegExpEdit
File
Help
New
Open
Save
Export Automaton
Export Grammar
RegExp Editor
Alphabet
RegExp
Simulation
RegExp
Editor
RegExp Project:
Introduction Wizard
A quick introduction to get in touch with RegExpEdit.
New RegExp
Create an empty project.
Getting Help:
Online FAQ
To get any further information visit the Online Help Center.
www.AtoCC.de
Visit the AtoCC website.
Genesis-X7 Software 2008


RegExpEdit
File
Help
New
Open
Save
Export Automaton
Export Grammar
RegExp Editor
Alphabet
RegExp
Simulation
RegExp
Alphabet
Edit Alphabet:
Add Alphabet Item
Delete Alphabet Item
Predefined Alphabet
Alphabet Items:
digital
letter
Clear
<<< Back
Next >>>
Genesis-X7 Software 2008


RegExpEdit
File Help
New Open Save Export Automaton Export Grammar
RegExp Editor Alphabet RegExp Simulation
RegExp Editor
RegExp
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
NEA Graph
Minimized NEA Graph
Start
q0
letter
q1
digital,letter
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


RegExpEdit [C:\Users\h_lai\Desktop\t1.xml]
File Help
New Open Save Export Automaton Export Grammar
RegExp Editor Alphabet RegExp Simulation
RegExp Editor
RegExp
Enter RegExp here
(a+b)*abb
Use the formal notation for regular expressions.
Like: (a+b)*ab(a+b)* for L = {w | w contains ab} over {a,b}.
Minimized RegExp
(a+b)*abb
Compare RegExp with another
Compare
Transform to NEA
Generate NEA graph for your RegExp at the right.
Show NEA
NEA Graph
Minimized NEA Graph
Start
q0
q1
q2
q3
a
b
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


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:
bbbbaab
Start Simulation
Speed:
Single Step
Export Scheme Code
Simulation:
Start
q0
q1
q2
q3
Configuration sequence
Check multiple input
Configuration sequence
M0
bbbbaab
bbbaab
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:
bbbbaabb
Start Simulation
Speed:
Single Step
Export Scheme Code
Simulation:
Start
q0 q1 q2 q3
b a b a b a b
Configuration sequence
Check multiple input
Configuration sequence
M0
q0 q0 q0 q0 q0 q1 q1 q3 q2
bbbbaabb bbbaabb bbaabb baabb aabb abb bb b
<<< Back
Genesis-X7 Software 2004 - 2008

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

AutoEdit Publish

Edit Publish settings:

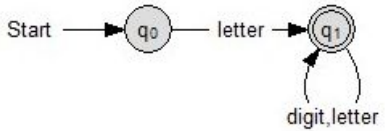
- Change Font
- Export Graph ...
- Export Definition ...
- Export Transitions ...
- Export as HTML

HTML Publish:

Automaton

Type: NEA

Transition Graph:



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

Transition Table:

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

Grammar:

G = (N,T,P,s)

G = ({q0,q1},{digit,letter},P,q0)

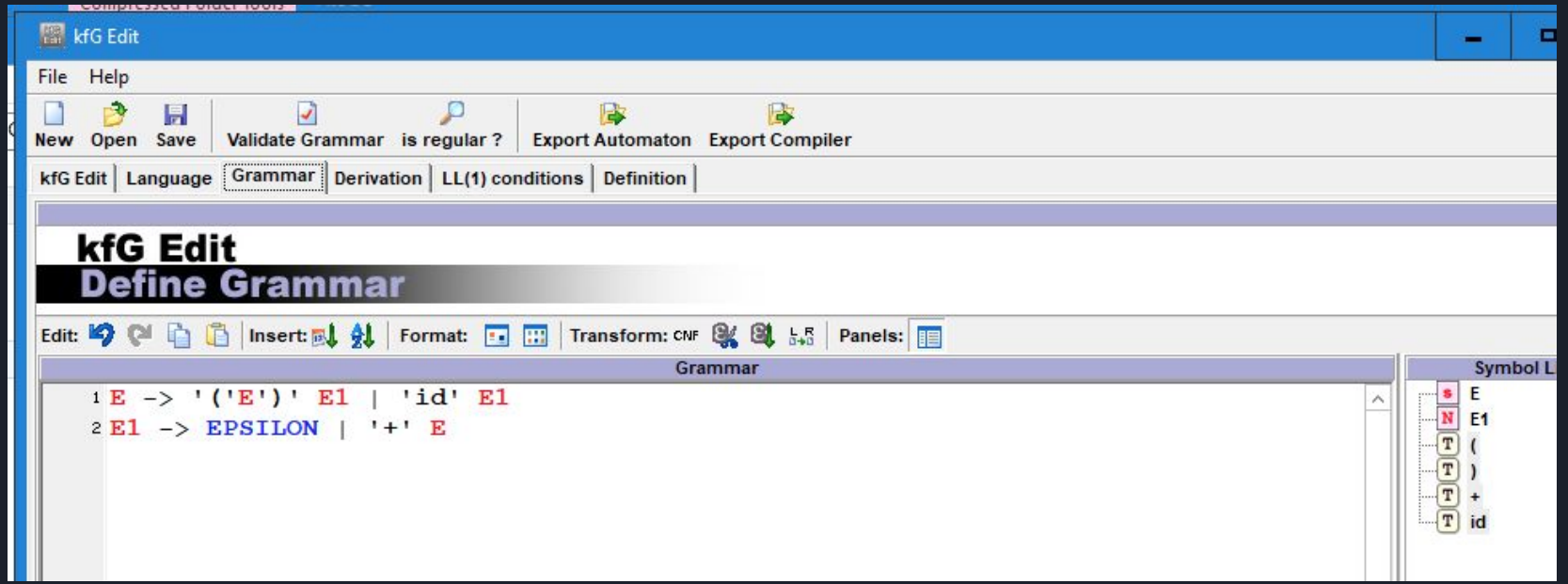
P = {
  q0 -> letter q1 | letter
  q1 -> digit q1 | digit | letter q1 | letter
}

RegExp: letter(digit+letter)*

<<< Back Next >>>

AtoCC Format

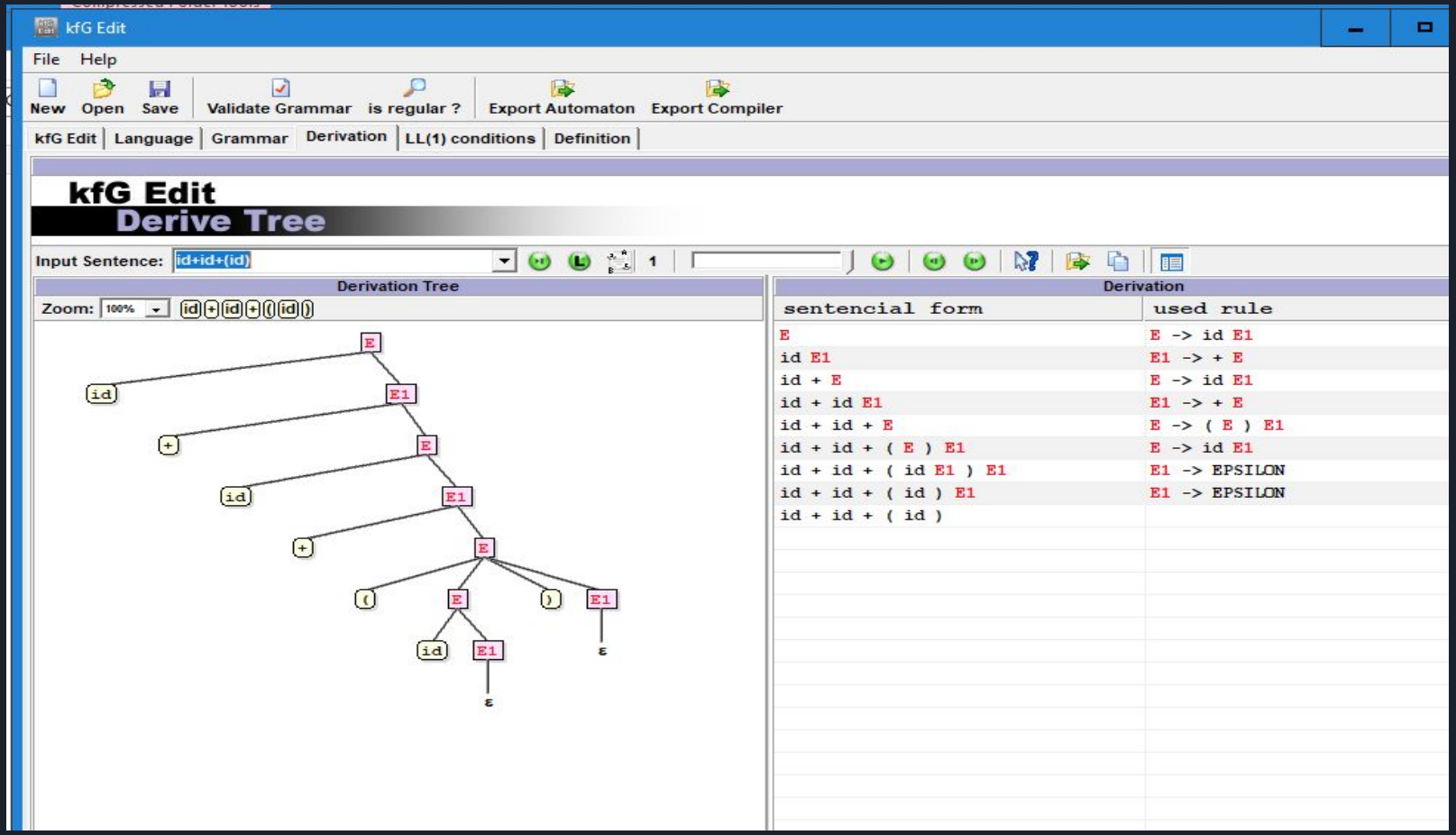
In assignment 2, you will have to use AtoCC to verify your grammar. For example, you will enter your grammar in the kfgEdit tool like this (simple grammar shown):



Continued --- AtoCC Format

Then, one very convenient thing that this tool allows is to verify a string against the grammar, by inputting a string in the "input sentence" field in the Derivation tab window (see the image in the next page)

This allows the tool to verify if this string is parsable or not, and if it doesn't generate a tree and a derivation for it. What I want the lexical analyzer to output is a string that you can copy in the"input sentence" box. This way, you can verify if your grammar is correct by using your lexical analyzer to output a string representing the token stream, and the kfgEdit tool to verify that you grammar can parse it.


kfG Edit
File Help
New Open Save Validate Grammar is regular ? Export Automaton Export Compiler
kfG Edit | Language | Grammar | Derivation | LL(1) conditions | Definition
kfG Edit
Derive Tree
Input Sentence: id+id+(id)
Derivation Tree
Zoom: 100%
Derivation
sentencial form | used rule
E | E -> id E1
id E1 | E1 -> + E
id + E | E -> id E1
id + id E1 | E1 -> + E
id + id + E | E -> (E) E1
id + id + (E) E1 | E -> id E1
id + id + (id E1) E1 | E1 -> EPSILON
id + id + (id) E1 | E1 -> EPSILON
id + id + (id)

Implementation of lexical analyzer

Two ways to implement the lexical analyzer:

1. Table driven (but constructing a transition table by hand is not an easy job)
2. Handwritten (it 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 can be accessed when the nextToken() method being called.

Continued --- Implementation

- 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)?

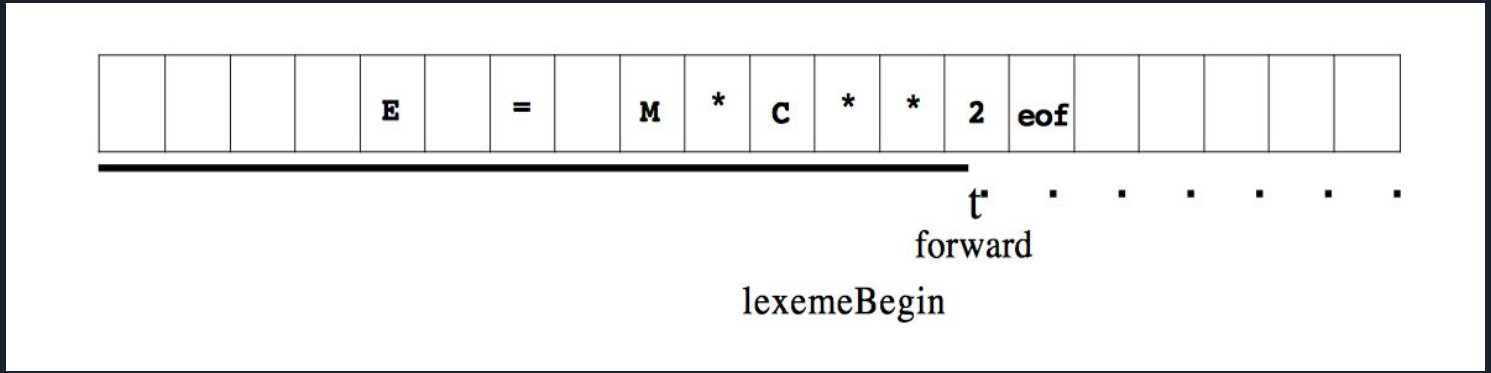
Continued --- Implementation

Obviously, we need to use buffers instead of reading when necessary, and there are two ways provided by the “Dragon book” (its real name is *Compilers: Principles, Techniques, and Tools*):

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.

Buffer in Pair



Involves two buffers that are alternately reloaded, Each buffer is of the same size N, and N is usually the size of a disk block, e.g., 4096 bytes.

we see forward has passed the end of the next lexeme, ** (the Fortran exponentiation operator), and must be retracted one position to its left.

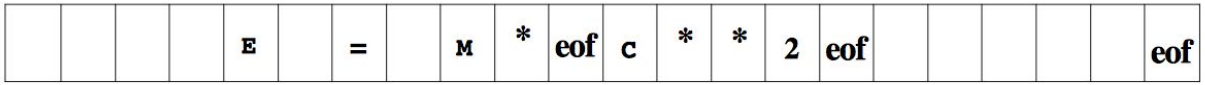
Continued --- Buffer in Pair

Two pointers to the input are maintained:

1. Pointer **lexemeBegin,** marks the beginning of the current lexeme, whose extent we are attempting to determine.

2. Pointer **forward** scans ahead until a pattern match is found; the exact strategy whereby this determination is made will be covered in the balance of this chapter.

Sentinel



forward

lexemeBegin

```
switch ( *forward++ ) {
        case eof:
                if (forward is at end of first buffer ) {
                        reload second buffer;
                        forward = beginning of second buffer;
                }
                else if (forward is at end of second buffer ) {
                        reload first buffer;
                        forward = beginning of first buffer;
                }
                else /* eof within a buffer marks the end of input */
                        terminate lexical analysis;
                break;
        Cases for the other characters
}
```

Thanks!

You are not allowed to use any tool like Lex can generate a Scanner automatically.