

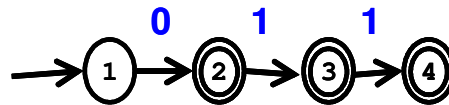
CMSC 330 Spring 2009 Midterm 1 (SOLUTIONS)

1. Regular expressions and finite automata

- a. (4 pts) Explain what we are referring to when we say regular expressions and finite automata are equivalent.

Regular expressions and finite automata are equivalent in that they can be used to describe/recognize the same set of strings (regular language), and may be converted into the opposite format and recognize the same language.

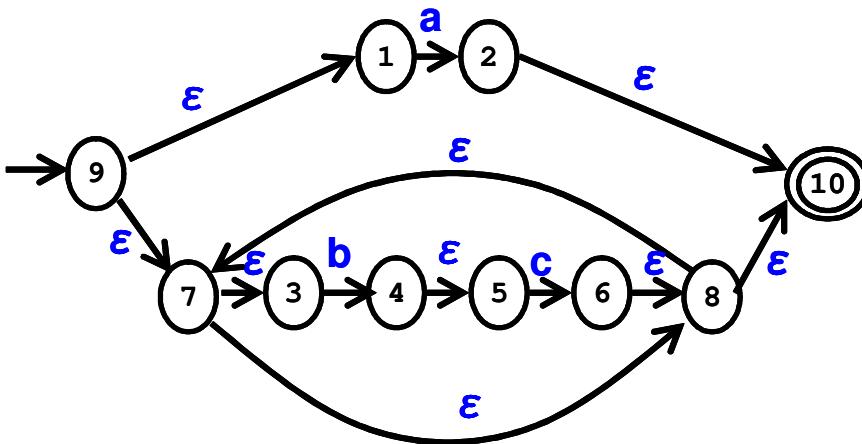
- b. (4 pts) Give a regular expression equivalent to the following finite automaton.



0|01|011 or 0(ϵ |1|11)

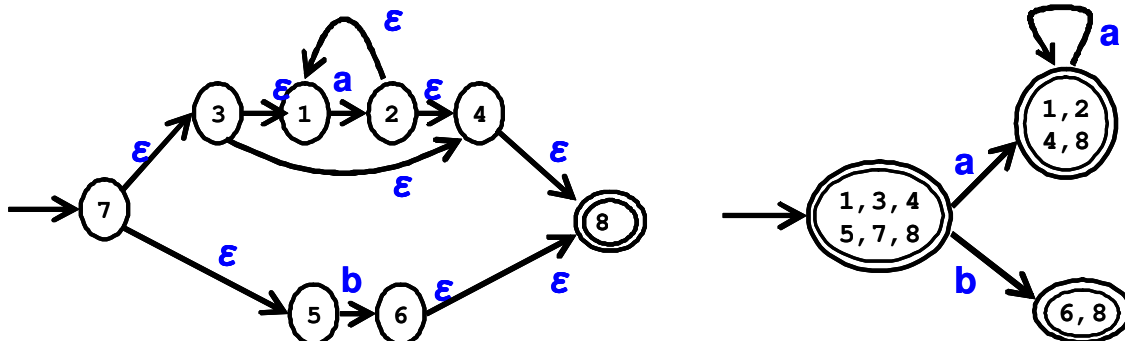
2. (12 pts) Regular expression to NFA

Create a NFA for the regular expression $a(bc)^*$ using construction method from class.



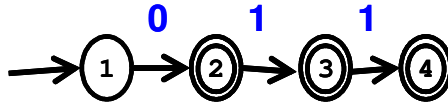
3. (16 pts) NFA to DFA

Apply the subset construction algorithm to convert the resulting NFA to a DFA. Show the NFA states associated with each state in your DFA.

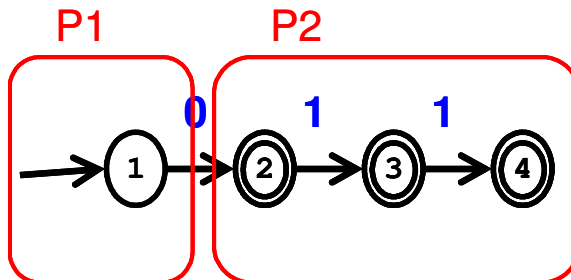


4. (16 pts) DFA minimization

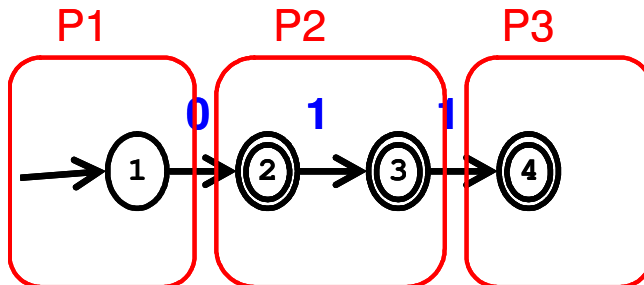
Apply Hopcroft minimization to the following DFA to generate a minimized DFA. List the DFA states in each partition created, and explain why the partition will or will not be split by the minimization algorithm, and list the DFA states in each resulting partition if the partition is split.



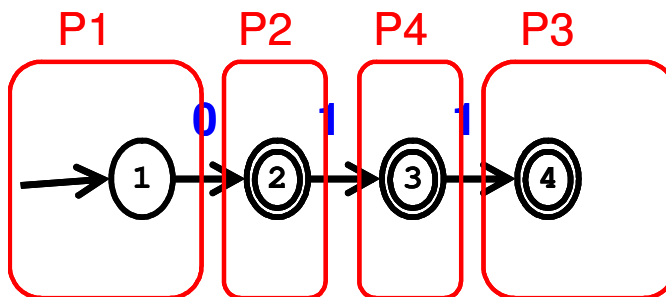
Initially states are split into two partitions, { 1 } for non-final states, and { 2, 3, 4 } for final states.



Partition { 2, 3, 4 } will be split into { 2, 3 } and { 4 } because behavior is different for input=1, since 2 & 3 stay in the partition but 4 rejects.



Partition { 2, 3 } will be split into { 2 } and { 3 } because behavior is different for input=1, since 2 will stay in the partition but 3 will enter new partition { 4 }.



Result is original DFA, which was already minimized.

5. (6 pts) Programming languages

- a. Name a disadvantage of dynamic types in terms of program correctness. Provide a code example in Ruby showing the disadvantage.

Type errors are not caught at compile time. // 3 pts

For example // 3 pts

```
x = 4
x = "a" if ( ... )
x = x / 2
```

is not a compile-time error. The error is hidden until if (...) = true

6. (12 pts) Ruby features

What is the output (if any) of the following Ruby programs? Write FAIL if code does not execute.

- a. `a = []` // nil b 3 // 3 pts

```
a[1] = "b"
a[2] = 3
a.each{ |x| puts x }
```

- b. `a = []` // FAIL // 3 pts

```
a[1] = "b"
a["2"] = 3
a.each{ |x| puts x }
```

- c. `a = { }` // 1 2 or 2 1 // 3 pts

```
a[1] = "b"
a["2"] = 3
a.each{ |x,y| puts x }
```

- d. `if ("terps rule!" =~ /[a-z]+/)` // terps nil // 3 pts

```
puts "#{$1}"
puts $2
else
  puts "None"
end
```

7. (30 pts) Ruby programming

Consider the following programming problem. Suppose you want to sort the list of alphabet symbols in output file of the Finite Automata (FA) project. Your Ruby program should read each line of in the file containing the output. For lines that represent a list of symbols in the alphabet of the FA, you should print the symbols in alphabetical order. Any other line you should leave alone and print the line as is.

Write a complete Ruby program that performs this sorting. The format of the output file of the FA project is shown in the following example. The portions of the output file in boldface are always fixed. Only the non-boldface portions of the FA

description may change. You may assume that all symbols in the alphabet are lowercase letters between a and z, and that they are always preceded by a single space in the output file.

For (10 pts) extra credit, you may also have your program print the list of states & final states in numerical order. You may assume that all states are numbers greater or equal to 0.

Helpful functions:

File.new(...)	// opens file
File.eof?	// at end of file?
File.readline()	// read single line from file
String.scan(...)	// finds patterns in String
String.to_i(...)	// converts String to int
print(...)	// prints arguments w/o newline

Input	Output	Extra Credit Output
% Start 9	% Start 9	% Start 9
% Final { 10 9 }	% Final { 10 9 }	% Final { 9 10 }
% States { 11 9 10 }	% States { 11 9 10 }	% States { 9 10 11 }
% Alphabet { b a }	% Alphabet { a b }	% Alphabet { a b }
% Transitions {	% Transitions {	% Transitions {
% (9 a 11)	% (9 a 11)	% (9 a 11)
...
% (9 10)	% (9 10)	% (9 10)
% }	% }	% }

```

file = File.new(ARGV[0], "r") // ARGV[0] file name      3 pts
until file.eof? do          // open file & read        3 pts
  line = file.readline      // read lines from file  3 pts
  if line =~ /Alphabet/     // action for Alphabet  3 pts
    a = line.scan(/ [a-z]/) // extract all symbols   8 pts
    a.sort!                 // sort symbols         3 pts
    print("% Alphabet {"") // print out symbols    4 pts
    a.each { |x| print(x) }
    print " }\n"
  elsif line =~ /(FinalStates)/ // action for Final & States 1 pts
    stateName = $1          // remember name        1 pts
    a = []
    line.scan(/\d+/) { |x| a.push(x.to_i) } // extract all state #s 3 pts
    a.sort!                 // sort on int value    4 pts
    print("% ", stateName, " { ") // print out states    1 pts
    a.each { |x| print(x, " ") }
    print " }\n"
  else
    puts line                // no action otherwise  3 pts
  end
end
end

```