**CS350 – Finite Automata**

**Purpose:** A finite automaton is a simple special case of a Turing machine in which the tape is not edited and the head always moves right. This simple computational model uses limited resources and is extremely useful in the computer science discipline. This model is also equivalent to a very useful model called a regular expression that is ubiquitous in computer science. The purpose of this module is for you to use finite automata and regular expressions to recognize languages and to start to explore the bounds of what this model can accomplish.

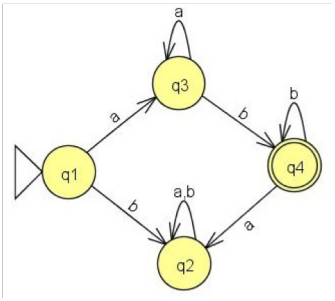
**Knowledge:** This module will help you become familiar with the following content knowledge:

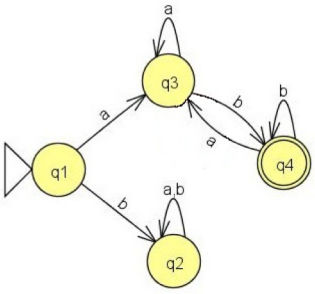
- How to use finite automata to recognize languages.
- How to analyze the equivalence of deterministic and nondeterministic finite automata.

**Activity:** With your group perform the following tasks and answer the questions. You will be reporting your answers back to the class in 60 minutes.

1. A deterministic finite automaton (dfa) can be represented by a state machine.

   Give a description of the language accepted by each of the following dfa's.



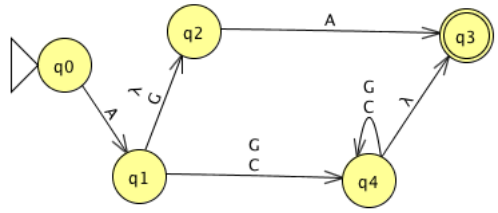
2. Determine whether the following statements are true or false. Defend your answer.

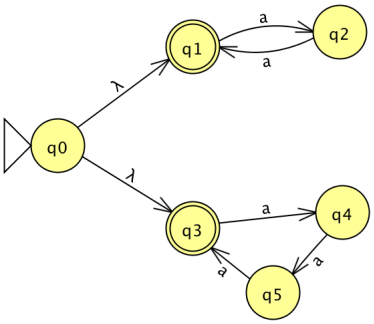
   (a) Is it true or false that dfa's can only accept finite languages.

   (b) Is it true or false that two different dfa's can't accept the same language.

3. A nondeterministic finite automaton (nfa) might have multiple transitions with the same symbol from a state, and/or might have transitions with the empty string which move from one state to another without consuming any input.

   Give a description of the language accepted by each of the following nfa's.



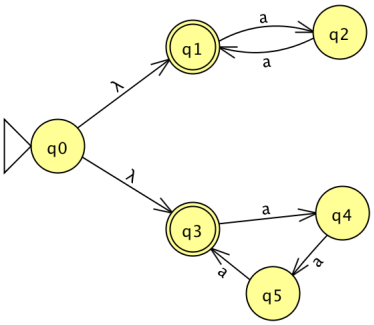



4. We can convert an nfa into an equivalent dfa. The key idea in the conversion is states $s_i$ in the new dfa correspond to sets of states $q_j$ in the nfa. For example, we could have a state $s_2 = \{q_5, q_8, q_9\}$.

   Algorithm:

   1. Create $s_0$ as a set of states $q_j$ that can be reached without reading an input symbol.
   2. For each state $s_i$ whose transitions have not all been finished yet, and for each input symbol $x$, figure out the set of all possible states $q_j$ that can be reached from $s_i$ by reading $x$. This set is a state $s_k$ in the dfa. It might already exist or you may need to create it. In either case, create the $x$-transition from $s_i$ to $s_k$.

   Convert this nfa to an equivalent dfa




5. Another equivalent way to describe a language is a regular expression, often called a regex. The following are all regular expressions:

   (a) $\emptyset$ represents the empty language

   (b) $\epsilon$ represents the language containing just the empty string

   (c) Any symbol $s \in \Sigma$ represents the language $\{s\}$

   We can build additional regular expressions by

   (a) $r_1 r_2$ represents the concatenation of the two languages $r_1$ and $r_2$ together

   (b) $r_1 \mid r_2$ represents the union of the two languages

   (c) $r^*$ represents the Kleene star of the language (which is the concatenation of 0 or more strings in the language). Sometime $r^+$ is used to represent the concatenation of 1 or more strings in the language.

   Give examples of strings in each of the following languages:

   (a) $1^*0$

   (b) $((a \mid 0)b^+c)^*$

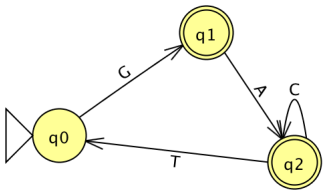
6. A regex can be converted into an nfa.

   How would you create an nfa that accepts the language $\emptyset$?
   How would you create an nfa that accepts the language $s$ where $s \in \Sigma$?
   Given an nfa for $r_1$ and an nfa for $r_2$, how would you put those together to create an nfa for $r_1 r_2$?
   How would you put the nfa's together to create an nfa for $r_1 \mid r_2$?
   How would you create an nfa for $r_1^*$?

7. Finally, we can show that regex and nfa models are equivalent by showing that an nfa can be converted to a regex. We will go through an example of how this can be done.

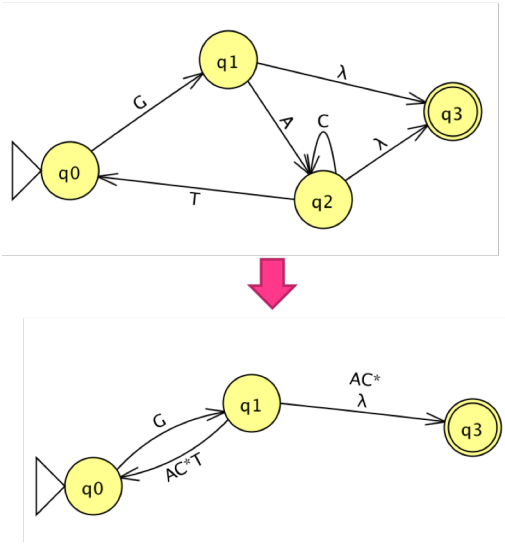
   Step 1. Modify the nfa so that it has just one final state.

   Do this for this nfa:



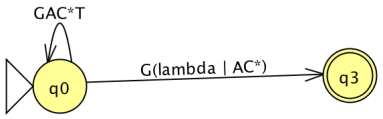

   Step 2. Eliminate states and relabel transitions with a regex.

   Here is how we can eliminate $q_2$:




   Give the nfa after eliminating $q_1$

Step 3. Once you have just a start state and a final state, give the regex for this simple nfa.
What regex does this two state nfa represent?




8. The program `egrep` is an acronym for "extended global regular expressions print" and may be used to search for a string or a more complex pattern in a file. Regular expressions provide a convenient, compact way of expressing patterns. The internal workings of egrep are based on finite automata.

   The command format is:

   ```
   egrep 'regexp' filename
   ```

   Egrep returns all lines in the file which contain a match for the regular expression.

   The format of a regular expression in egrep is as follows:

| regular expression | egrep notation |
|---|---|
| r* | `r*` |
| $r^+$ | `r+` |
| r + $\lambda$ | `r?` |
| r + s | `r\|s` |
| rs | `rs` |
| ( r ) | `(r)` |
| char c | c |
| special char | `\c` |
| any symbol | `.` |
| beginning-of-line | `^` |
| end-of-line | `$` |
| any character listed | `[ ]` |
| any character not listed | `[^ ]` |

   In a terminal window on phoenix try the following commands and then clearly and succinctly describe the pattern expressed by each of the regular expressions:

   (a) `egrep 'depend' /usr/share/dict/words`

   (b) `egrep '^y.*y$' /usr/share/dict/words`

   (c) `egrep '^s..u.t..e$' /usr/share/dict/words`

   (d) `egrep '[qQ][^u]' /usr/share/dict/words`

Complete the following assignments for grading. Each should be done individually but you may consult with a classmate to discuss strategies.

**Assignment 1**:
Create deterministic finite automata in JFLAP to recognize each of the following languages using the alphabet $\Sigma = \{a,b\}$:

1. all strings with exactly one a.
2. all strings with no more than two a's.
3. all strings with at least one b and exactly two a's.

**Criteria for Success:** Test each of your dfa's and make sure that each accepts **exactly** the languages provided.

**Assignment 2**:
Any language that can be accepted by a DFA is called a **regular language**. Show that if $L$ is a regular language then $L - \{\lambda\}$ is regular as well. Note that $\lambda$ represents the empty string.

**Criteria for Success:** You can do this by describing how you would modify a DFA which accepts $L$ so that you get another DFA which accepts $L - \{\lambda\}$. You need to have a general enough explanation so that it will apply to **any** language but it might help you to first examine a couple of particular languages that accept $\lambda$ to see how they may be modified.

**Assignment 3**:
Complete exercise 9.11 on p190 in the text.

**Criteria for Success:** You have a regex for the language and have tested its correctness using JFLAP. (You will have to use the facility to convert the regex to an nfa, export this machine and then run your test inputs).

**Assignment 4**:
Write `egrep` commands for the following:

1. All lines that contain the letter `a` and the letter `b` (either upper or lower case) appearing in any order.
2. All lines that contain the word `a` (either upper or lower case) followed by a word starting with a vowel

**Criteria for Success:**Test your commands with the file `~jillz/cs350/lab7/testfile1` and on any other testfile of your choosing and verify your results.

**Assignment 5**:
Try the regex `[0-9]?[0-9]:[0-9][0-9](am|pm)` on the file
`~jillz/cs350/lab7/testfile2`
Correct this regex so that it does not match illegal times like 99:99pm

**Criteria for Success:** Your regex works for legal times but fails for times where the hour is greater than 12 or the minutes are greater than 59.

Submit your all your files in Canvas for grading.