CS 162FZ: Introduction to Computer Science II

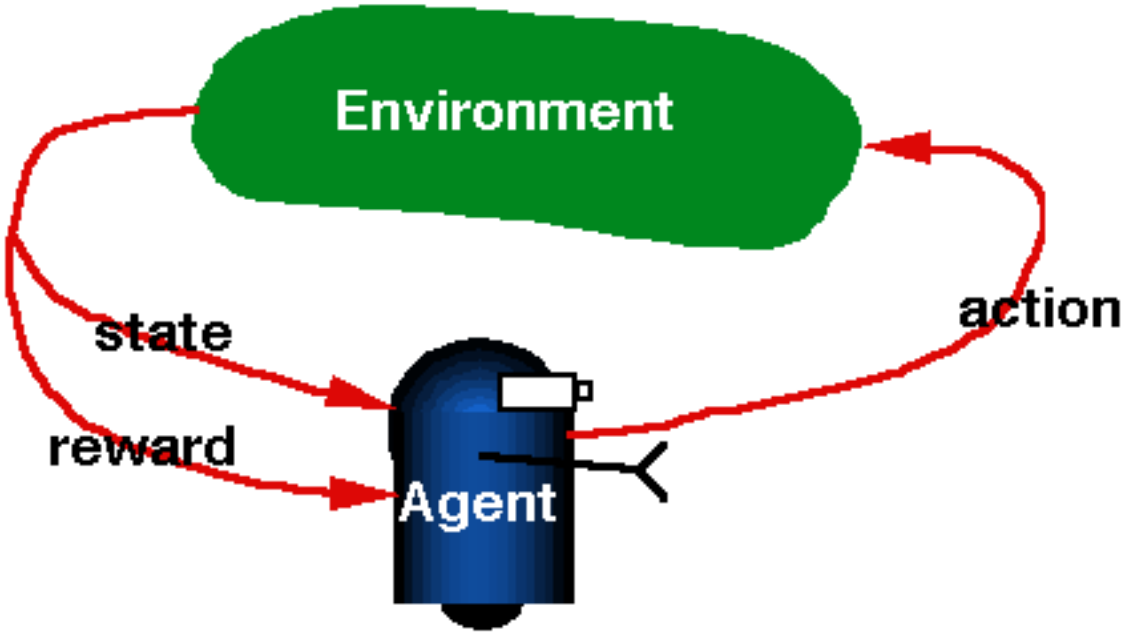
Lecture 08

Finite State Machines(FSM)

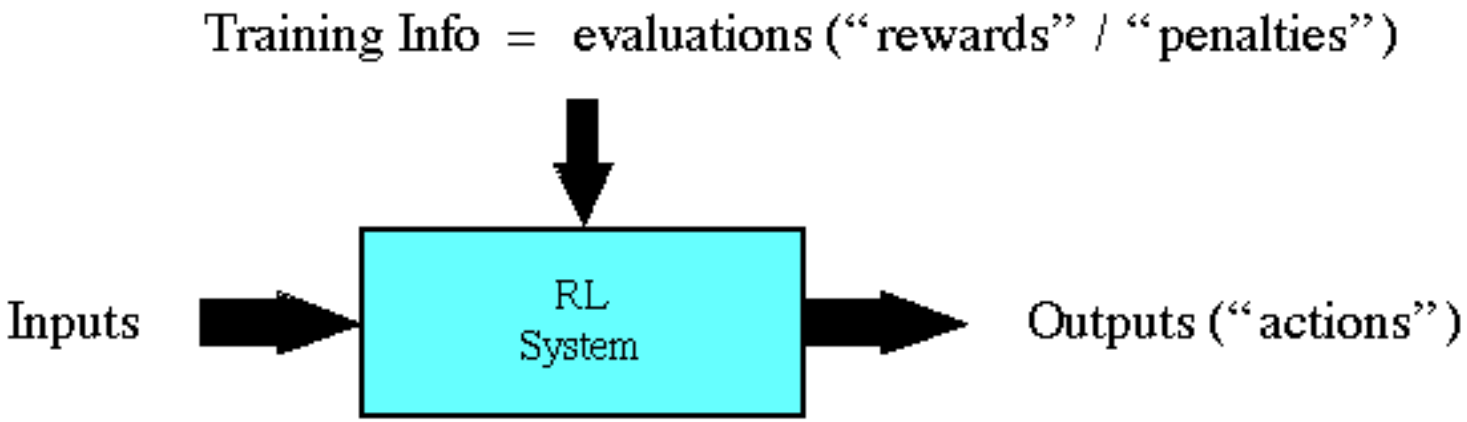
Dr. Chun-Yang Zhang

Complete Agent

- Temporally situated
- Continual learning and planning
- Object is to *affect* the environment
- Environment is stochastic and uncertain



Reinforcement Learning



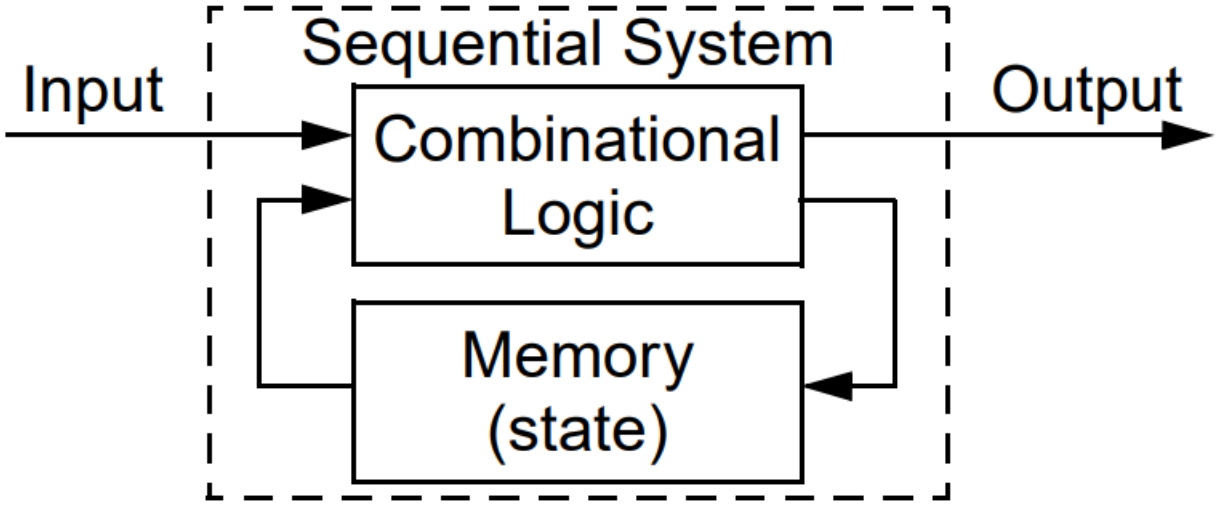
Objective: get as much reward as possible

Finite State Machines

A finite state machine (sometimes called a finite state automaton) is a computation model that can be implemented with hardware or software and can be used to simulate sequential logic and some computer programs. Finite state automata generate regular languages. It can be used to model problems in many fields including mathematics, artificial intelligence, games, and linguistics.

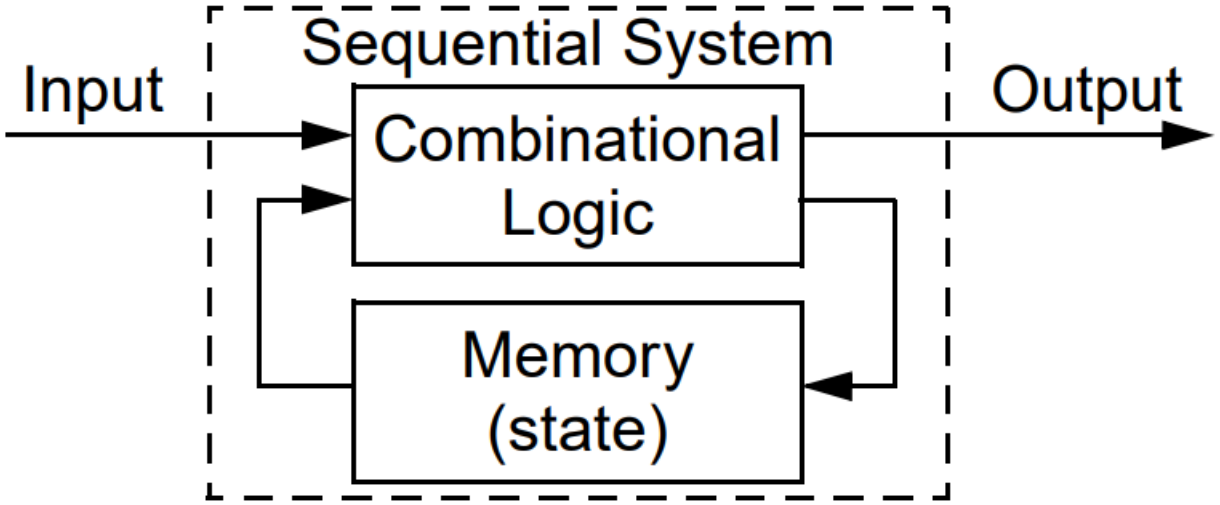
Mealy State Machine

Sequential system where output depends on current input and state.



Moore State Machine

Sequential system where output depends only on current state.



Introduction

Finite State Machines (FSMs) offer a different perspective on regular expressions. A FSM describes the behaviour of a Regular Expression (RegEx). Each FSM is composed of the following 5-tuple (Q, Σ, δ, q0, F) where:

- *Q:* a finite set of states
- Σ*:* a finite alphabet set
- *δ*: Q X Σ → Q: transition function - a set of maps from states and inputs into states
- q0: an initial state (q0 ∈ Q)
- F: a set of final/accepting states (F ⊆ Q)

FSMs are abstract machines that can only be in one of a finite number of states (Q) at any given time. The FSM can transition from one state to another in response to some external inputs.

States

The set of states usually describes the current state of the machine. The current state of the machine is usually dependent on the previous state of the machine. There are always two special states:

1. **Start State**

- The FSM always begins in this state. The start state is usually represented with an arrow "pointing at it from anywhere".

States

2. **Finish** State(s)

- The FSM must finish in this state (or one of these states) if the language is to be recognized. Finish states are those where the machine reports accepts the input. Accepting states are usually represented by double circles.

All other states are regarded as **intermediate states**. States are usually denoted using qx where x is some number.

Alphabet & Transition Functions

Alphabet

An alphabet describes the accepted inputs for a machine. For instance, an alphabet could be:

{0,1} – the language of only 0's and 1's

{a,b,c} – the language of only a's, b's, and c's

Transition Function

The transition function for any state and alphabet symbol is one or more next states to transition into.

Alphabet & Transition Functions

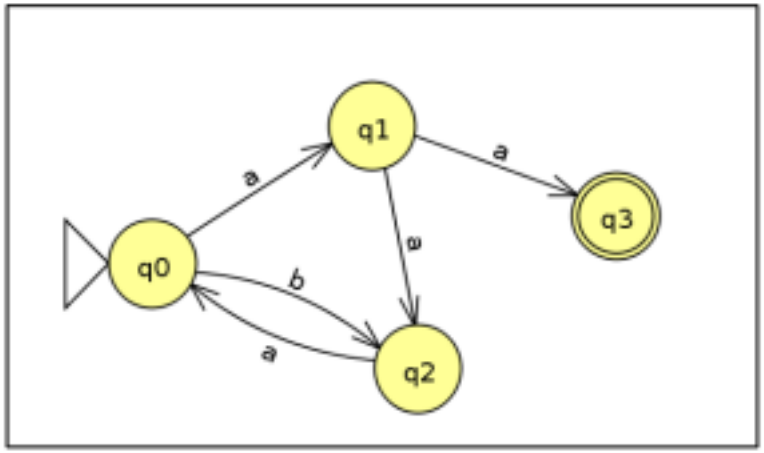
In the previous example:

- The set of states are {q0,q1,q2,q3}

  - q0 is the start state (symbolised with the triangle arrow in to it)

  - q3 is the final state (symbolised the with the double circle)

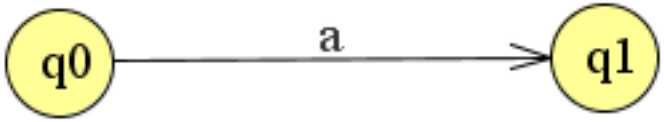
- The alphabet is {a,b}



Alphabet & Transition Functions

- The transitions are represented by the arrows with the alphabet symbol above them.

For example **:**



means that we leave state q0 and go to q1 if we get an “a” as input.

Alphabet & Transition Functions

When using FSMs we often test strings from a language to see if they are accepted or rejected. We define a string *s* that is accepted by a FSM as:

- It begins in the start state,
- Takes a transition for each symbol in the string,
- Finishes in a final state after every symbol in the string has been processed.

Transition Table

An FSM can be represented in a table. This table will contain the states, alphabet and transitions. Taking the example above we can represent the FSM in a transition table as follows:

States \ Symbols	a	b
q0	$q1$	$q2$
q1	$q2, q3$	–
q2	$q0$	–
q3	–	–

Note that if we are in state q1 and are presented with the symbol a, we can transition to either q2 or q3 as shown in the Transition Table.

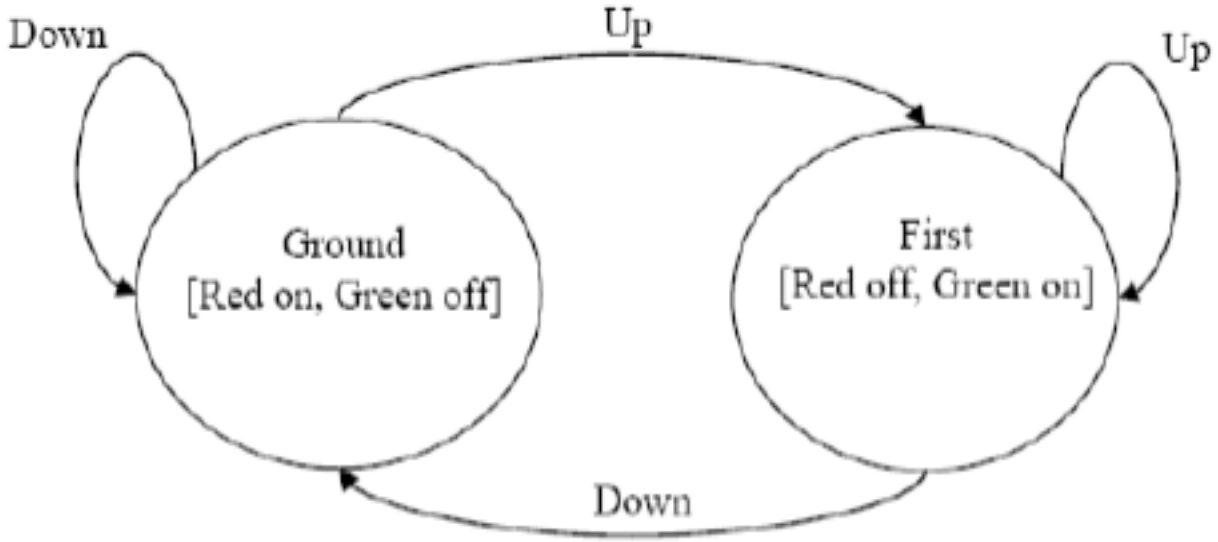
Example FSM

Let us start with an example of a controller for an elevator. The conditions are:

- The elevator can be at one of two floors: Ground floor or First floor.
- There is a button controlling the elevator, and it has two values: Up or Down.
- There are two lights in the elevator that indicate the current floor: Red for Ground, and Green for First.
- At each time step, the controller checks the current floor and current input, changes floors and lights if required.

Example FSM

All this information may be represented in a state diagram.



Example FSM

If we represent Ground as 0 and First as 1, and Down as 0 and Up as 1 the transition table for this FSM is:

CurrentState	Input	NextState	Red	Green
0	0	0	1	0
0	1	1	1	0
1	0	0	0	1
1	1	1	0	1

What is the starting state? What is the accepting state?

Lexicographical Ordering

When enumerating all possible strings in an alphabet, we display them in lexicographic order.

This means listing **shorter strings first, and for those with the same length,** listing them in alphabetical or numerical order.

An example of lexicographical ordering would be:

- Given an alphabet Σ = $\{a, b\}$, list all strings of less than length four in L, where L is all possible strings made from Σ = $\{a, b\}$.
- L={λ, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb}

  (λ *is the empty string*)

Lexicographical Ordering

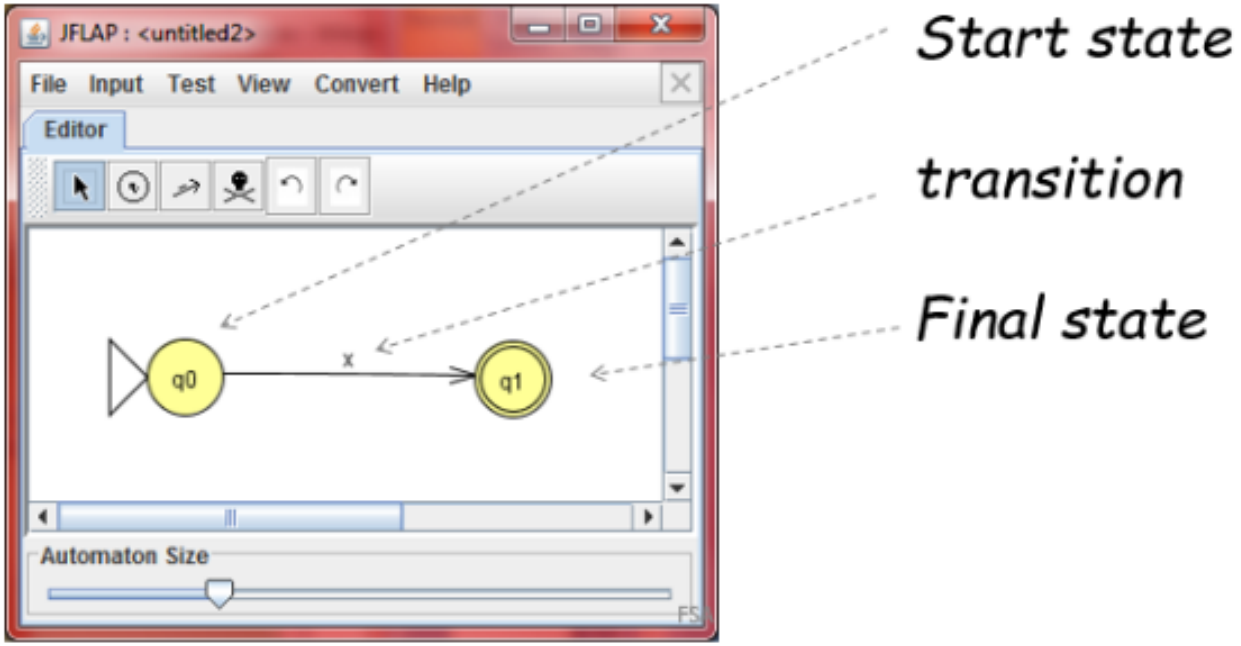
All accepted strings for FSM's should be ordered lexicographically.

To do:

Given an alphabet Σ = $\{a, b\}$, list all strings of less than length five in L, where L is all strings beginning with b and ending in b. Some valid words are: bb, bab, bbb, baab, …

JFLAP

- JFLAP is a tool which allows us to model the operations of FSMs. You can download it from http://www.cs.duke.edu/csed/jflap/
- Here is a simple JFLAP representation of the language composed of a single x:

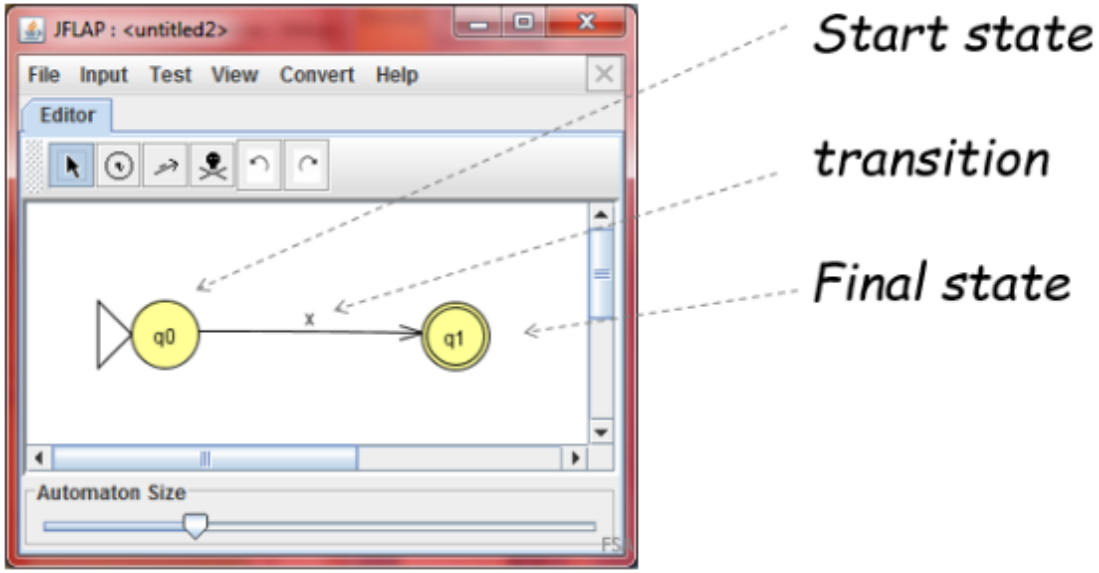


JFLAP

Let us step through this example:

- We can see that we start in state q0
- We follow the transition to state q1 if there is an x on the input
- Once we are in state q1 we can accept the string

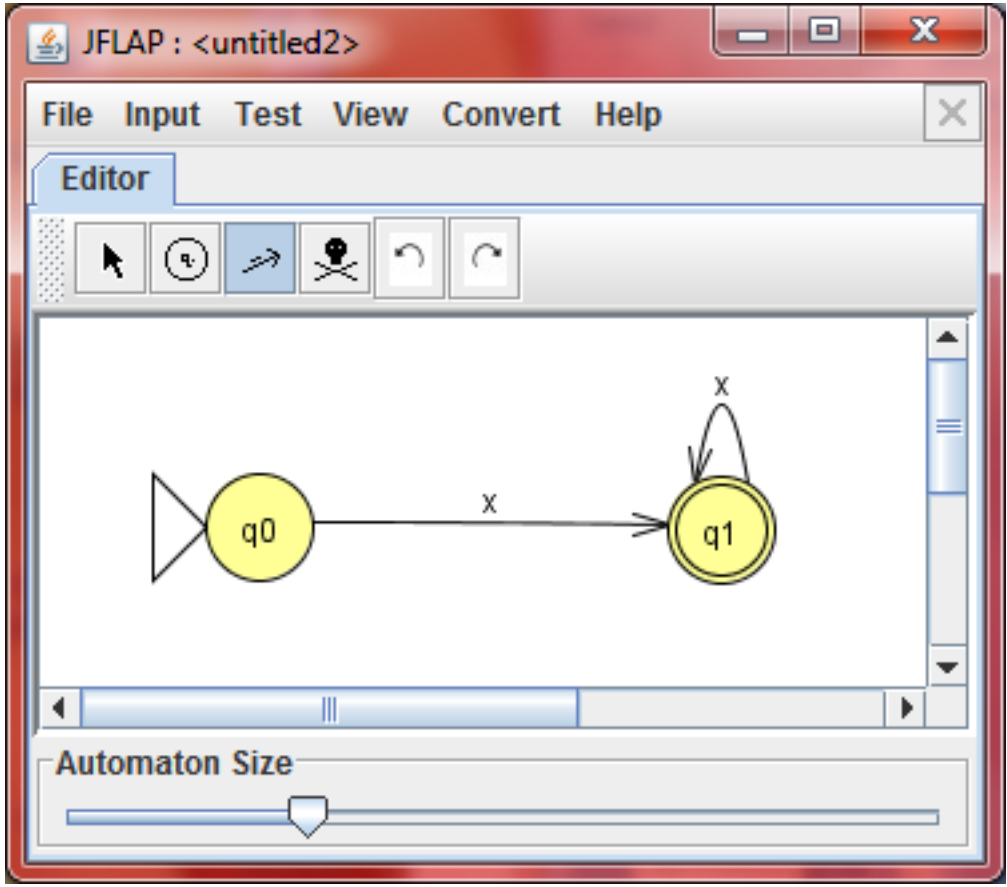
Here is the language composed of one or more x's represented in a FSM in JFlap:



JFLAP

Let us step through this example:

- Start in state q0
- Follow the transition to state q1 if there is an x on the input
- If there are more x's transition to stay in q1
- Only state q1 is an accepting state and once in state q1 accept the string



JFLAP

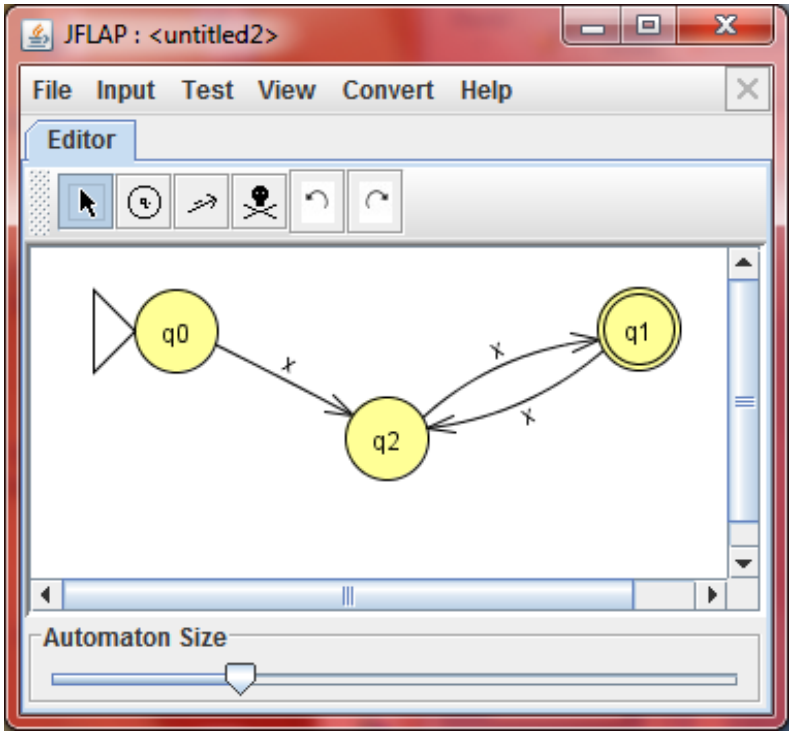
As a next example let us construct the FSM for the language composed of an even number of x's. It is useful to first think of some examples of correct words in the language:

xx

xxxx

xxxxxx

Let us now construct the FSM that will accept all even length strings containing x.



JFLAP

Analysing the FSM you should see that only an even amount of x's will be accepted. We start in state q0 and transition to state q2 when we get an input of x. If we get another x we transition to state q1. q1 is an accepting state – is the current word a valid word in the language? Yes – because the word is composed of two x's – namely xx.

What happens if we get another x or another two x's, or more x's?

q1 is the accepting state and can only be reached by having an even number of x's. Try out some of examples to ensure you understand yourself with JFLAP.

JFLAP - Exercise

To do:

Construct a FSM that will accept all words comprised of one or more occurrences of the string xy concatenated together from the language Σ={x,y}.

Valid words are: xy, xyxy, xyxyxy, xyxyxyxy,…

Deterministic Finite State Automata(DFA)

A deterministic finite automata (DFA) is a type of FSM that has only one transition for each label in a state. In the following example we can see an example of this:

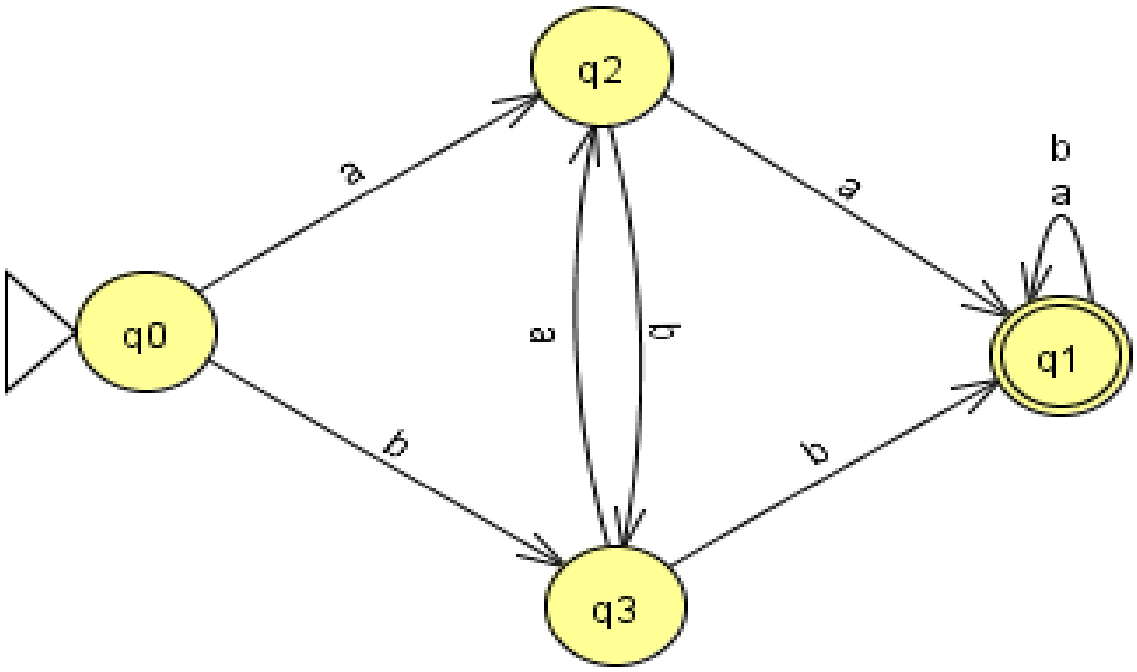
- The set of states is {q0,q1,q2,q3}

  - Start state – q0

  - Final state – q1

- The alphabet is {a , b}
- The transitions are represented by the arrows with the alphabet symbol on it.

Deterministic Finite State Automata



Notice that each state has only one transition for one alphabet symbol leaving the state.

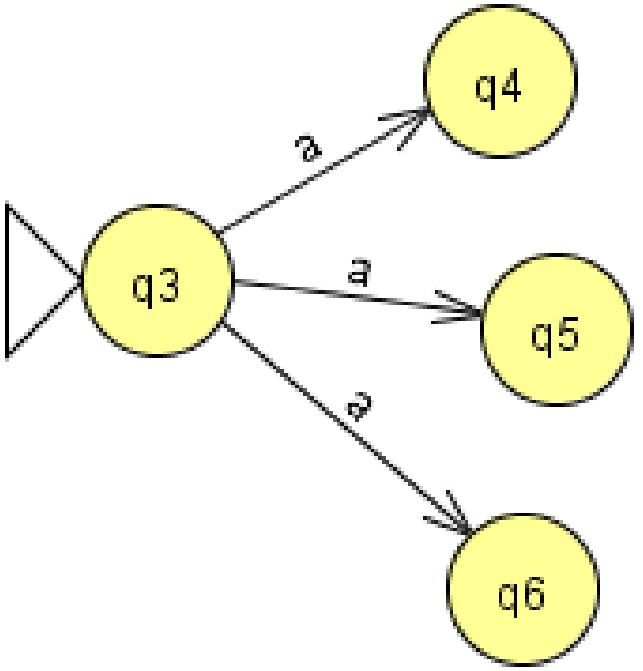
Deterministic Finite State Automata

If we construct the transition table for the DFA we can see that each state will have two transitions – one for each alphabet symbol.

		Symbols	
		a	B
States	q0	q2	q3
	q1	q1	q1
	q2	q1	q3
	q3	q2	q1

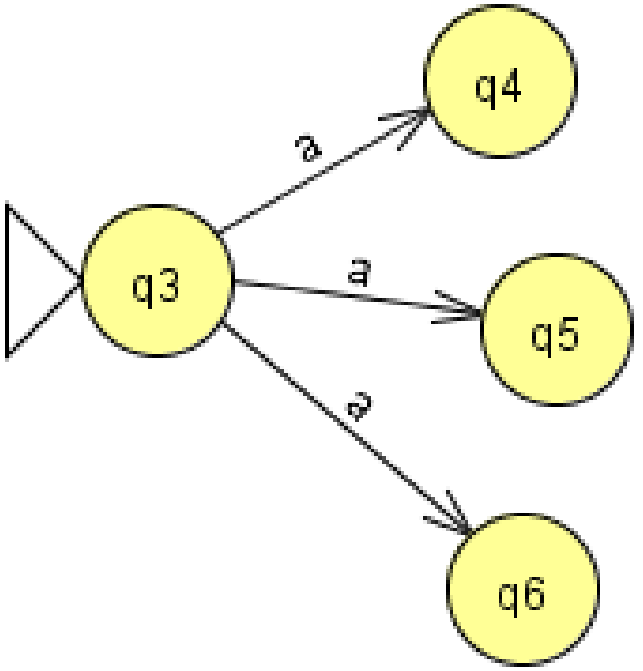
Non-deterministic Finite Automata

- Non-deterministic finite automata (NFA) allow for multiple transitions with the same label for many states.
- This corresponds to two (or more) entries in a cell in the state transition table.
- It is not deterministic because the next state could be any one of a number of possible states.



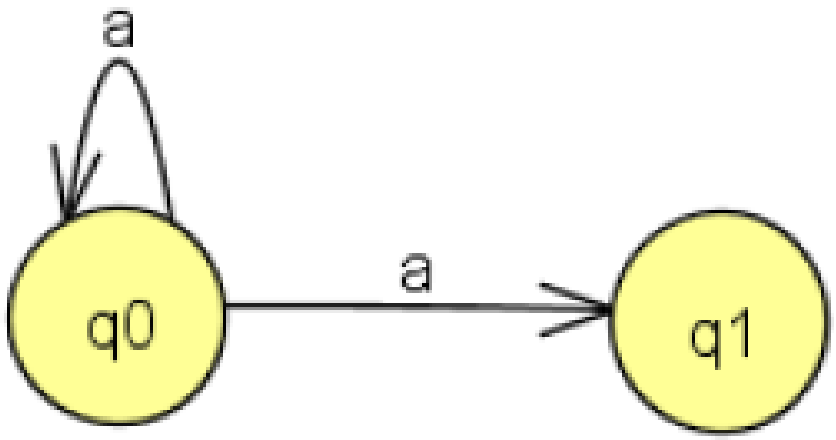
Non-deterministic Finite Automata

- If we are in state q3 and we get an input of a which arrow do we follow?
- We would follow all possible arrows, even if only one of them turns out to be correct.
- So we have to go to q4, q5 and q6 and see what the next character will be.



Non-deterministic Finite Automata

Consider the following NFA. In state q0, if we get an input of a, which state do we transition to?



We can either stay in q0 or move to q1.

Non-deterministic Finite Automata

- An NFA has more than one entry in at least one cell of the State Transition Table.

- The transition table would be:

	A
q0	q1, q0
q1	

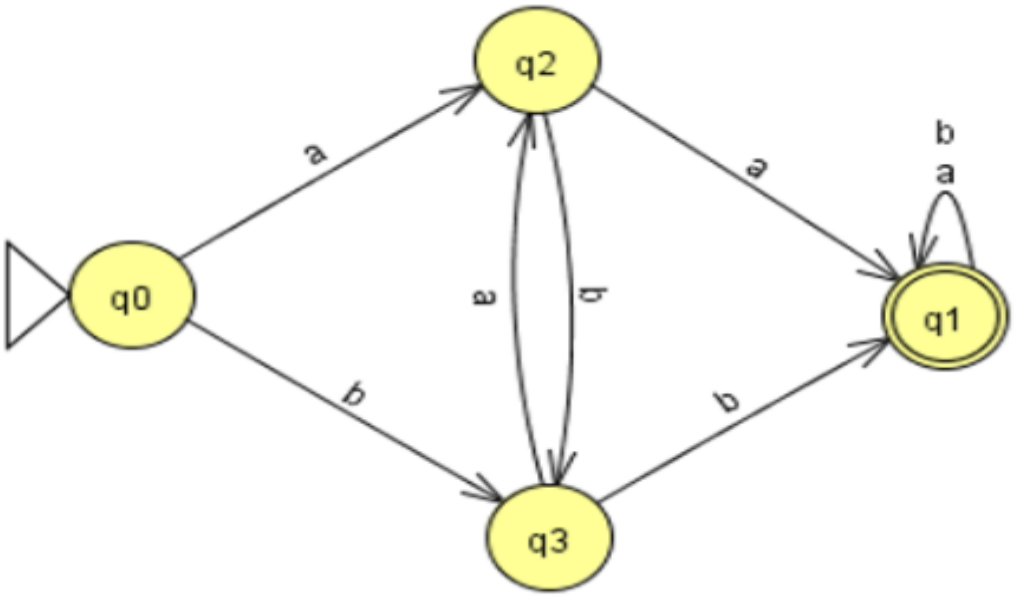
Non-deterministic Finite Automata

- Although DFA and NFA have distinct definitions, it can be shown that they are equivalent.
- That is, for any NFA we can construct an equivalent DFA and vice versa.
- We can even convert between them but it's not straight forward.
- Both NFA and DFA only recognise regular languages.

DFA - Exercises

To do:

Can you determine what language the following machine accepts is?



DFA - Exercises

The approach to determining the language is:

- Start at the start state and follow through the graph until you reach an accepting state

- List the strings accepted in this state in Lexicographic order.

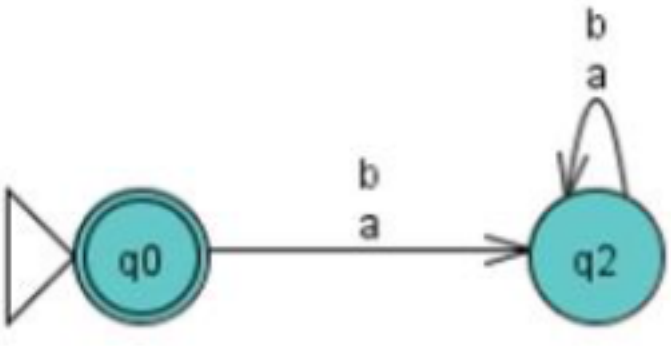
For this example some of the strings are:

aa, bb, aaa, aab, abb, baa, bba,… , aaaa, aaab, … ,abba, … baab, … aaaaa, aaaab, …

- This machine defines strings over {a , b} with a double letter in them.

DFA - Exercises

Can you determine what language the following machine accepts?



Summary

- FSM's are defined by the 5-tuple $(Q, \Sigma, \delta, q0, F)$ where:
  - Q: a finite set of states
    - The states describe the current state of the machine.
  - Σ: a finite alphabet set
    - The alphabet describes the accepted inputs.
  - $\delta: Q \times \Sigma \rightarrow Q$: transition function - a set of maps from states and inputs into states
    - The transition function describes how to transition from one state to another.
  - q0: an initial state ($q0 \in Q$)
    - There is always a start state.
  - F: a set of final/accepting states ($F \subseteq Q$)
    - There is always an initial state and at least one final state.

- A deterministic finite machine is similar to a FSM only this time there is only one transition for each label in a state.

- Non-deterministic finite automata allow for multiple arrows with the same label to exist for many states.

Advantages of Finite State Machine

The advantages of Finite State Machine include the following.

- Finite state machines are flexible
- Easy to move from a significant abstract to a code execution
- Low processor overhead
- Easy determination of reachability of a state

Disadvantages of Finite State Machine

The disadvantages of the finite state machine include the following

- The expected character of deterministic finite state machines can be not needed in some areas like computer games
- The implementation of huge systems using FSM is hard for managing without any idea of design.
- Not applicable for all domains
- The orders of state conversions are inflexible.