Copyright Warning

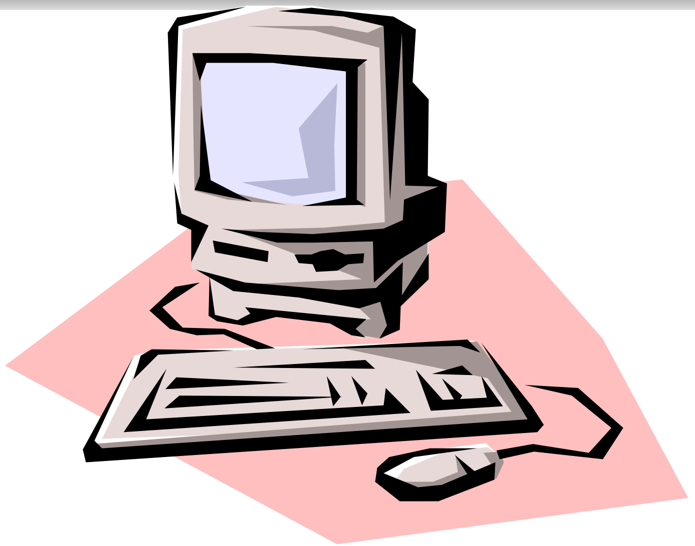
Curtin
University of Technology

Housekeeping

- Emergencies
- Consultation Hours
- Unit Outline
- Attendance (Lectures, Tutorials)
- Class Rules (Punctuality, Mobile Phones, *etc*).

Curtin University of Technology

THEORY OF COMPUTATION



Module 1: Automata and Languages

Lecture 1: Deterministic Finite Automata

Curtin University of Technology

Readings

- Sisper – Sections 0 & 1.1

  OR

- Savage – 1.1 to 1.3, 4.1

  (the rest of Ch 1 is also useful)

- Simulation – jFAST

  (http://sourceforge.net/projects/jfast-fsm-sim/)

- Simulation - JFLAP

  (http://www.jflap.org/)

Curtin University of Technology

Topics

- Deterministic Finite Automata (DFAs)
- Computation with DFAs
- Designing DFAs
- Regular Language

Unit Learning Outcomes

Outcome 1

Synthesize Finite Automata, Push-Down Automata, Context Free Grammars, and Turing Machines with specific properties, and relate one form to another.

Assessment Criteria

- You need to be able to:
  - ➢ **Model** a specification as a Deterministic Finite Automata (DFA).
  - ➢ **Explain** the operation of a machine on an input string.
  - ➢ **Classify** a problem as belonging to a class able to be modelled by a DFA.

BACKGROUND

Simple model of computing

Example

Representations

What is a computer ?

- The Theory of Computing begins with the question above.
  - ➢ Real computers are too complex for direct mathematical representation

- Use of a computational model
  - ➢ An idealized computer
    - Accurate in some ways but not in others
    - Use different models depending on the features we want to focus on
  - ➢ One simple model: a Finite Automaton (FA)
    - Note that Savage also discusses an even simpler model – basic electrical circuits.

Curtin University of Technology

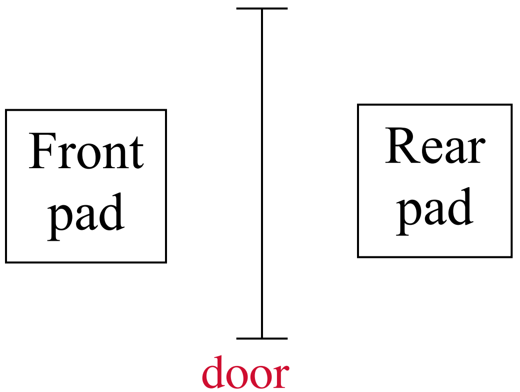
Finite Automata

- Finite Automata concept based on
  - ➢ Models of computers with an extremely limited amount of memory
    - What can such a computer do?
      - Many useful things!
    - We interact with such computers all the time
      - The heart of many electromechanical devices
    - An example:
      - Controller of an automatic door

Example: Automatic door control

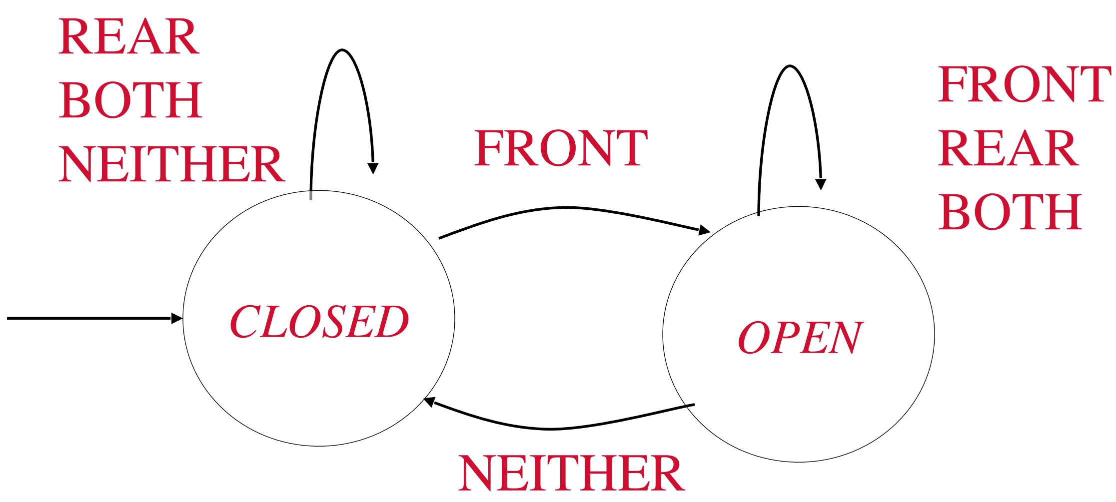
- Front pad to detect the presence of a person about to walk through
- Rear pad to hold the door long enough for the person to pass all the way through
- **States** of controller and door:
  - ➢ OPEN or CLOSED
- Four possible **input conditions**:
  - ➢ FRONT (person standing in front), REAR, BOTH, or NEITHER

- Top view of an automatic door:



Curtin University of Technology

State Diagram



- State diagram for automatic door controller
- Each **state** is represented by a circle. The **initial state** has an arrow entering on the left.
- Each state has an arrow leading from it for each possible **input condition**.

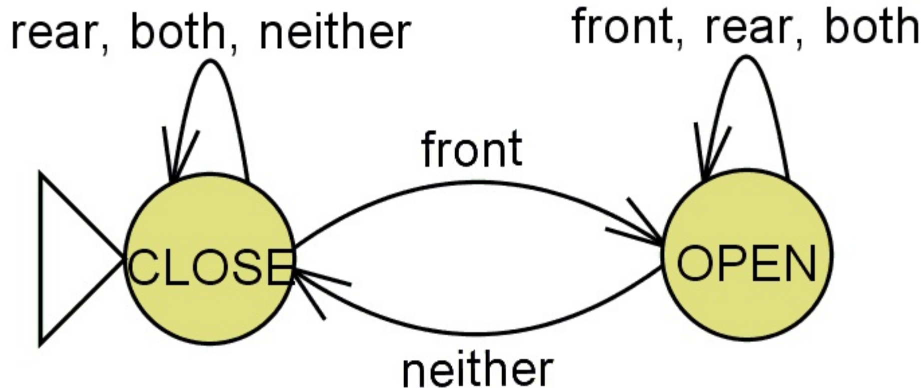
State transition table

- Input conditions and state transitions for automatic door controller can also be shown in a table.
- Note that there are a **finite** number of **states**, and movement between them is fully **deterministic**.

	Input signal			
State	**NEITHER**	**FRONT**	**REAR**	**BOTH**
CLOSED	CLOSED	OPEN	CLOSED	CLOSED
OPEN	CLOSED	OPEN	OPEN	OPEN

Simulation

- This door controller can be built using the JFLAP simulator.



- The .xml file containing the controlled is available on Blackboard. Feel free to experiment with it.

- JFLAP Note – This simulator doesn't like input conditions of more than one character. To run a simulation use jFAST instead.

Curtin University of Technology

Finite Automaton As Controller

- Provides a standard way of representation
  - ➢ A computer with just a single bit of memory
  - ➢ Records which of the two states the controller is in

- Controllers for other common devices
  - ➢ Have slightly larger memories
  - ➢ *e.g.*, elevator controller:
    - states represent floors
    - inputs are signals from buttons
    - several bits to keep track of the information

Curtin University of Technology

Other Applications

- Markov chains
  - ➢ Probabilistic counterpart of finite automata

- Finite Automata & Markov Chains useful for recognizing patterns in data
  - ➢ speech processing and optical character recognition
  - ➢ Markov Chains used for predicting price changes in financial markets

Curtin University of Technology

FORMAL DEFINITION OF A DFA

Formal Definition

Transition Function

Example

Formal Definition: Concept

- Need for formal definition
  - ➢ It is precise: resolves any uncertainties about what is allowed in a finite automaton
  - ➢ Provides notation: good notation helps to think and express thoughts clearly
- A finite automaton consists of
  - ➢ A set of states
  - ➢ An input alphabet (the allowed input symbols)
  - ➢ Rules for going from one state to another, based on the input symbol
  - ➢ A start state and a set of accept/final states

Transition Function

- Denoted by δ (delta)
  - ➢ Defines the rules for moving from one state to another
  - ➢ An arrow from state x to y with input symbol 1, indicated by $\delta(x, 1)=y$

- A mathematical shorthand
- Deterministic computation: δ is a function
  - ➢ When a machine in a given state reads the next input symbol, the next state is unique.

Curtin University of Technology

Definition of Finite Automaton

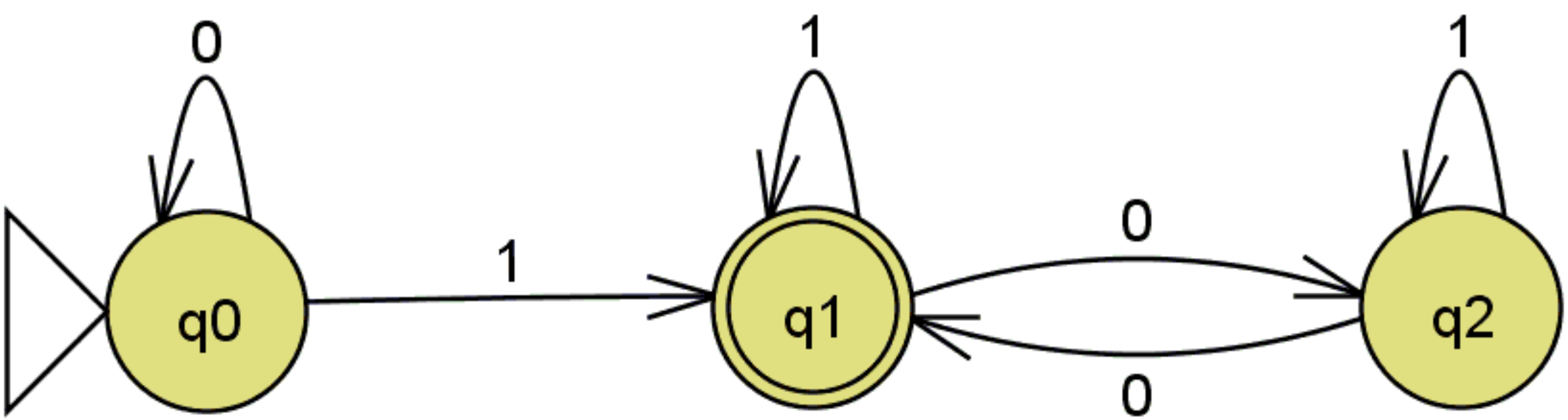
- A Finite Automaton is a 5-tuple (Q, Σ, δ, q_0, F), where:
  - ➢ Q is a finite set called the *states*.
  - ➢ Σ Is a finite set called the *alphabet*.
  - ➢ $\delta : Q\times\Sigma\rightarrow Q$ is the *transition function*.
  - ➢ $q_0 \in Q$ is the *start state*, and
  - ➢ $F \subseteq Q$ is the *set of accept states*.

- This precisely describes what is meant by a finite automaton.
- Precision at this level is important for mathematical proof.

Curtin University of Technology

Example Machine M

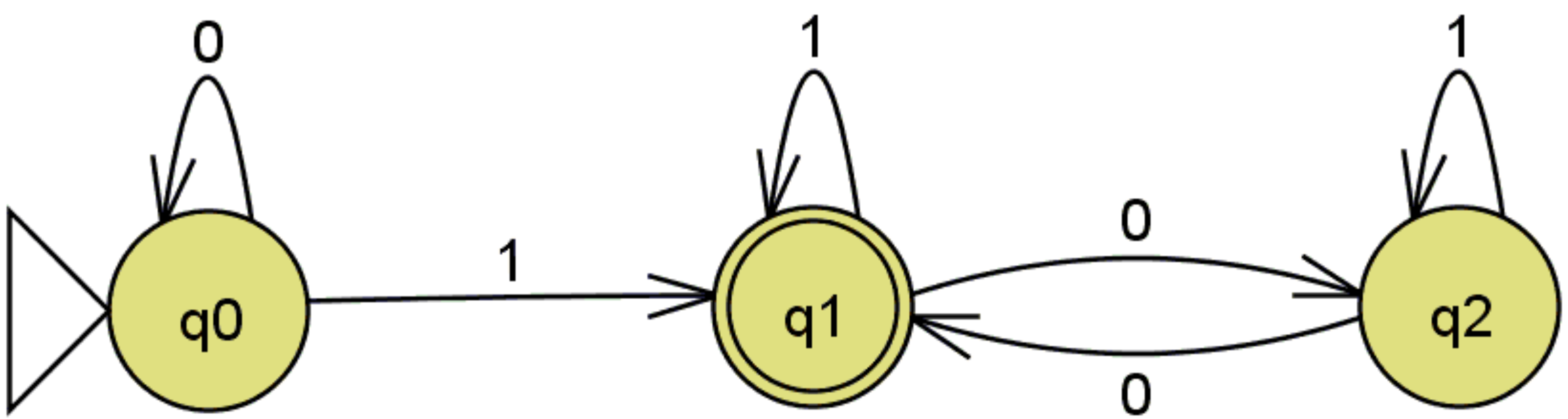
- To fully describe a machine, specify each of the five elements in the definition.



Curtin University of Technology

State Diagram of M_1

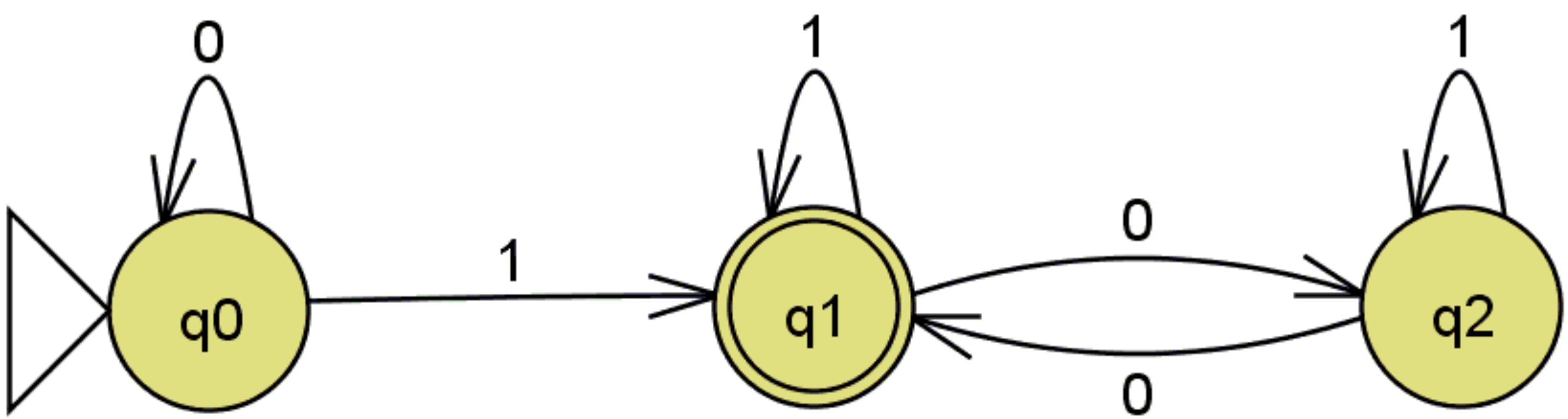
- A labeled directed graph
- 3 states: q_0, q_1, q_2
  - ➢ Start state q_0, accept state q_1
- Transitions
  - ➢ Arrows from one state to another



Specifying M

- M= (Q, Σ, δ, q_0, F), where
  - $Q=\{q_0,q_1,q_2\}$
  - $\Sigma = \{0,1\}$
  - $q_0=q_1$
  - $F=\{q_2\}$

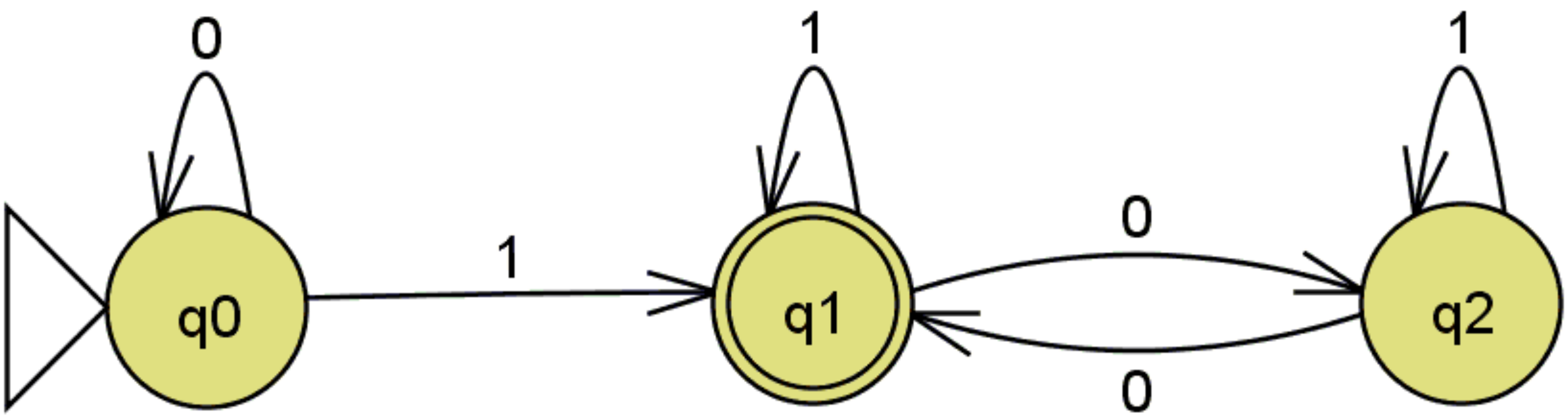
δ	0	1
q_0	q_0	q_1
q_1	q_2	q_1
q_2	q_1	q_1



Curtin University of Technology

Language of M

- *Let A* = {*w* | *w* contains at least one 1 and an even number of 0s after the first 1}
- We say *L*(*M*) = *A*, or *M* recognizes *A*.
- Note that the machine we've been looking at does exactly this.
- What level of task is this?



COMPUTATION WITH FINITE AUTOMATA

Formal Definition

Inputs and Outputs

Example

Language of a Machine

- If A is the set of all strings that machine M accepts, then we say that A is the *language of machine M.*
  - Written as $L(M) = A$
  - *M recognizes A* or *M accepts A*
  - To avoid confusion of machines accepting strings and accepting languages, we use the term *recognize* for languages
  - A machine may accept several strings, but recognizes only one language.

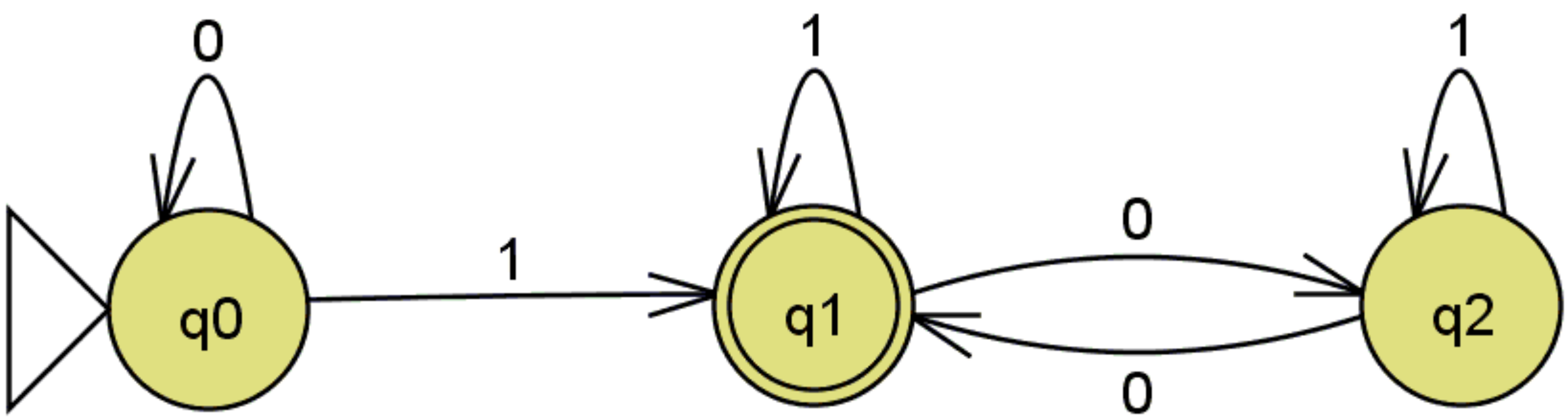
Curtin University of Technology

Definition of Computation

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and $w = w_1 w_2 \ldots w_n$ be a string over the alphabet Σ
- M <u>accepts</u> w if a sequence of states $r_0, r_1, \ldots, r_n$ exists in Q such that
  - ➢ $r_0 = q_0$,
  - ➢ $\delta(r_i, w_{i+1}) = r_{i+1}$ for $i = 0, \ldots, n\text{-}1$, and
  - ➢ $r_n \in F$.
- M <u>recognizes</u> language A if
  - ➢ $A = \{w \mid M \text{ accepts } w\}$

Curtin University of Technology

Inputs and Outputs



- Receives an input string (*e.g.*, 1101)
- Processes the string one symbol at a time
- Produces output after reading last symbol
  - ➢ *Accept* if M in accept state, otherwise *reject*

Strings Accepted by M

Check M on 11, 100, and 10100

1 1:	$q_0\ q_1\ q_1$	q_1 is an accept state
	1 1	11 accepted by M
100:	$q_0\ q_1\ q_2\ q_1$	q_1 is an accept state
	1 0 0	100 accepted by M
10100:	$q_0\ q_1\ q_2\ q_2\ q_1\ q_2$	q_2 is not an accept state
	1 0 1 0 0	10100 rejected by M

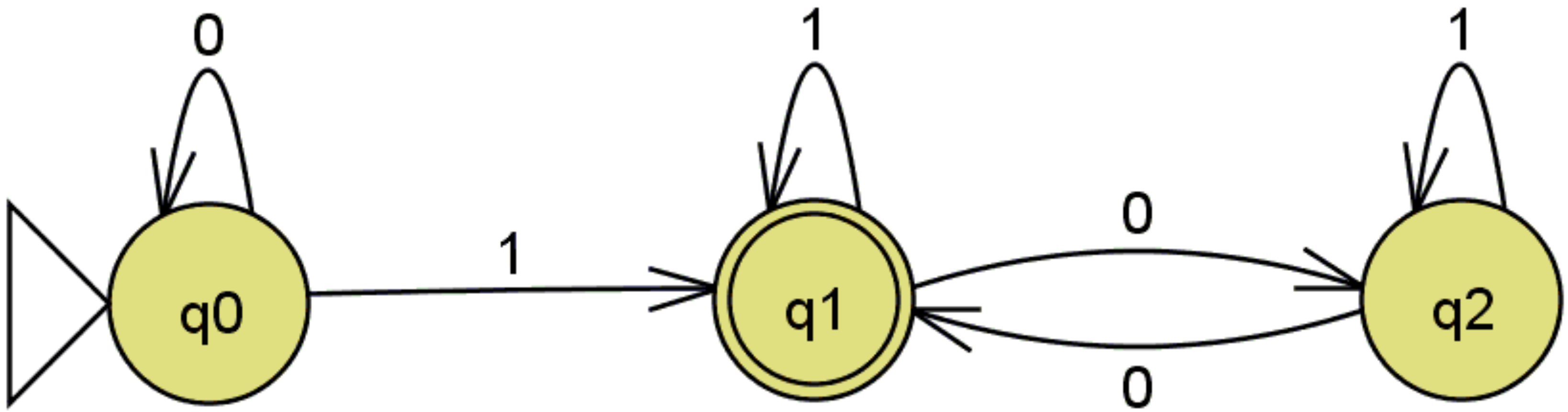
Strings Accepted by M

- We can see that these computation examples agree with our earlier interpretation of M.
- M_1 accepts any string in $\{w \mid w$ contains at least one 1 and an even number of 0s follow the first1$\}$
- Want to formalize this.

Language of a Machine

- If A is the set of all strings that machine M accepts, then we say that A is the *language of machine M.*
  - Written as $L(M) = A$
  - *M recognizes A* or *M accepts A*
  - To avoid confusion of machines accepting strings and accepting languages, we use the term *recognize* for languages
  - A machine may accept several strings, but recognizes only one language.

Language of M



- Let $A = \{w \mid w$ contains at least one 1 and an even number of 0s follow the first1$\}$
- L(M) = A, or M recognises A.
- This is a *regular* language. Discussed in Week 3.

Curtin University of Technology

DESIGNING A FINITE AUTOMATA

Concept

Example

How to Design Finite Automata?

- Pretend to be the automaton
  - ➢ You receive an input string
  - ➢ Need to determine whether it is a member of the language the automaton is to recognize
  - ➢ See the symbols one at a time
  - ➢ After each symbol, decide whether the string seen so far is in the language
    - the string may end at any point, so must be ready with an answer
    - if the string seen so far is in the language, the current state should be an accept state, otherwise it should not be an accept state

Curtin University of Technology

Designing Finite Automata

- What is to be remembered about the string you are reading ?
  - ➢ The Automaton has a finite set of states or a finite memory
  - ➢ For many languages, only need to remember the crucial information
  - ➢ What is crucial depends on the language being considered

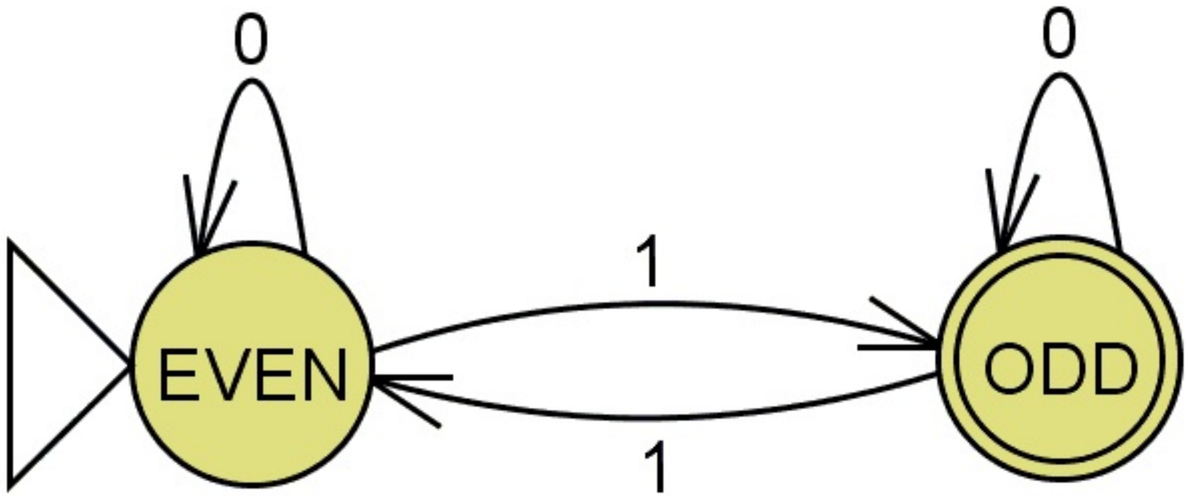
Curtin University of Technology

Design Example

- Alphabet $\Sigma = \{0,1\}$
- Language consists of all strings with an odd number of 1s
- Construct an automaton to recognize this language
- What is to be remembered ?
  - Whether the number of 1s seen so far is odd or even
  - If you read a 1 flip the answer, if you read a 0 leave the answer as is

Design Example

- A finite list of possibilities
  - ➢ Two states *even* and *odd*
- How to go from one possibility to another
  - ➢ Transitions for each input symbol
- Set state for 0 symbols seen so far (empty string ε)
  - ➢ start state is *even* (0 is even)
- Set accept states to possibilities where you want to accept the input string
  - ➢ *odd* (odd number of 1s are seen)



Curtin University of Technology

Deterministic Finite Automata Task Level

- If a problem can be solved using a DFA, how hard is it to write a program?

- Answer – very easy.

- It can probably be handled by a simple script or module.
  - ➢ There should be no need to undergo the full software design process.
  - ➢ You may consider PERL, PYTHON or a shell script for the task.

Curtin University of Technology

Deterministic Finite Automata Task Level

- You will probably never have to design a DFA in order to help you finish a task like this.

- However, designing such a DFA proves that you have correctly classified the task.

- It can be easy to be wrong.
  - Consider the problem "Accept any string that has the same number of 1s and 0s".
  - This problem looks similar to the problems we have seen so far.
  - This problem **can not** be solved with a DFA.

Curtin University of Technology

Summary

- Finite automata
  - ➢ Definition Examples
  - ➢ Be able to state the formal definition
- Designing FA
  - ➢ <ULO> Generate State Diagram based on informal (English) specification
  - ➢ <ULO> Translate between Transition Function and definition.
- Computation with FA
  - ➢ <ULO> Explain the operation of a FA
- Regular language
  - ➢ Introduction to a theme
  - ➢ <ULO> Generate State Diagram based on formal specification

Curtin University of Technology