Copyright Notice

Curtin University of Technology

Theoretical Foundations of Computer Science 300

Lecture 2

Non-Deterministic Finite Automata

Curtin
University of Technology

Outline

- Non-Determinisism
  - ➢ Definition and characteristics of a Non-Deterministic Finite Automaton (NFA)
  - ➢ Computation with NFA
  - ➢ Equivalence of NFA to DFA

Curtin University of Technology

Unit Learning Outcomes

- Synthesize FA, PDA, CFG, and TMs with specific properties, and to relate and convert from one form to another.

Curtin University of Technology

Assessment Criteria

- **Model** a specification expressed in English or Mathematics as a NFA.
- **Explain** the operation of a machine on an input string.
- **Prove** that a string belongs to a language.
- **Convert** one formally specified NFA into an equivalent specification DFA.

Why NFAs?

- Learning about NFAs doesn't extend the problems you can model, since we'll show that their power is equivalent to that of DFAs.

- So why bother?

- Two reasons:
  1. It is often easier to design a NFA than a DFA, which gives you a short-cut.
  2. The concept of non-determinism is needed later, so it's best introduced at the simplest level.

Curtin University of Technology

Introducing the Non-Deterministic Finite Automaton

Background

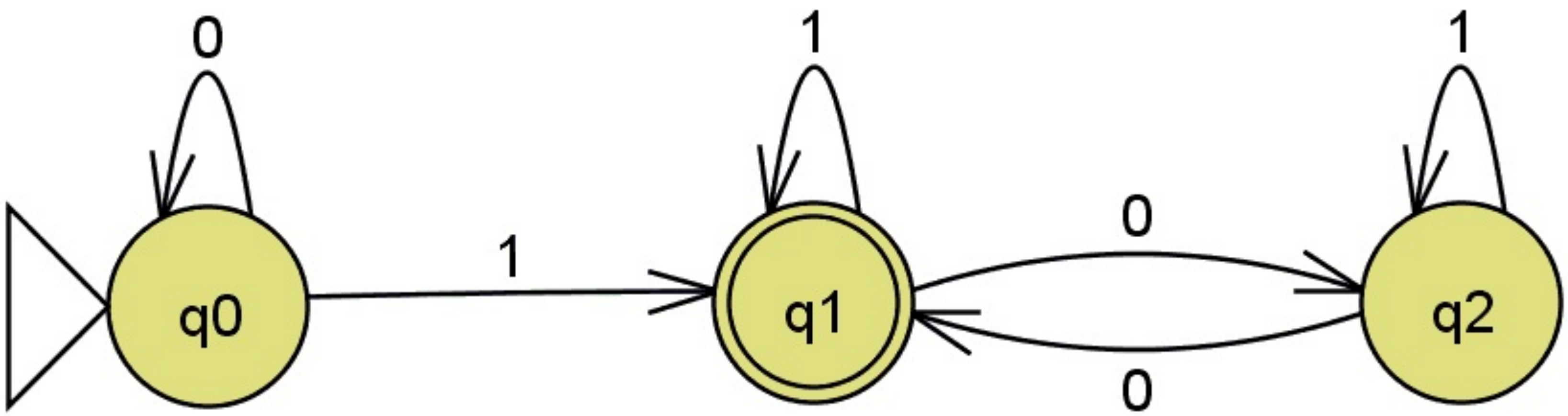
NFA Example

Formal Definition of DFA

Generate NFA

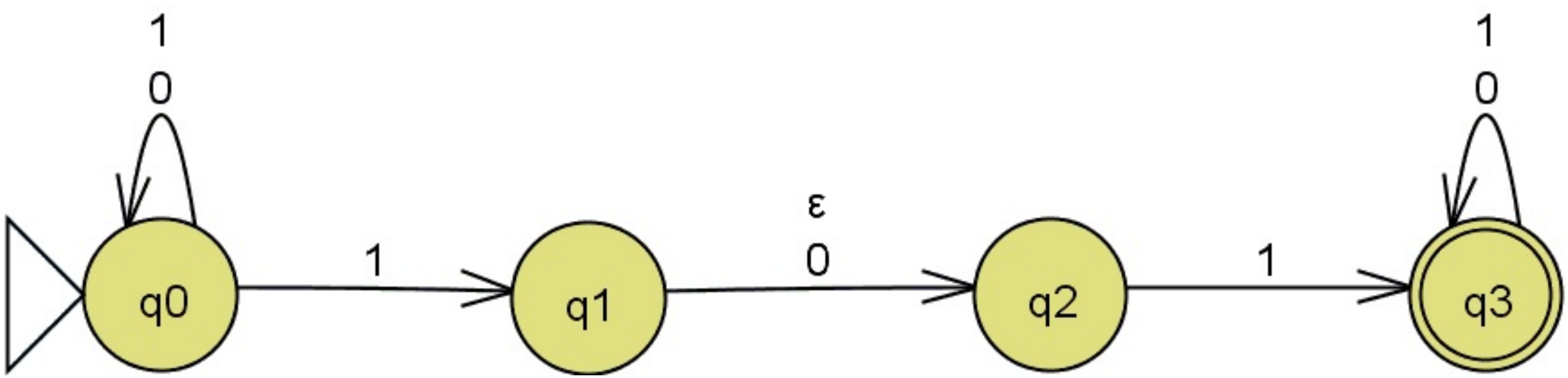
Curtin University of Technology

Deterministic FA



- Deterministic computation
  - ➢ When a machine in a given state reads the next input symbol, the next state is unique.

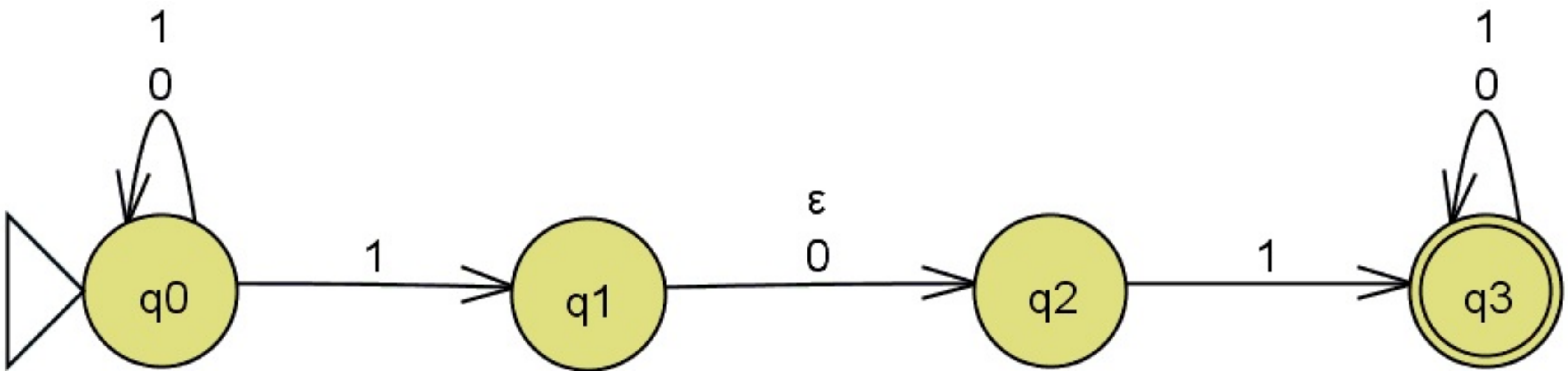
Non-deterministic FA



- Non-deterministic machine
  - ➢ Several choices may exist for the next state at any point
  - ➢ A generalization of deterministic machines

Curtin University of Technology

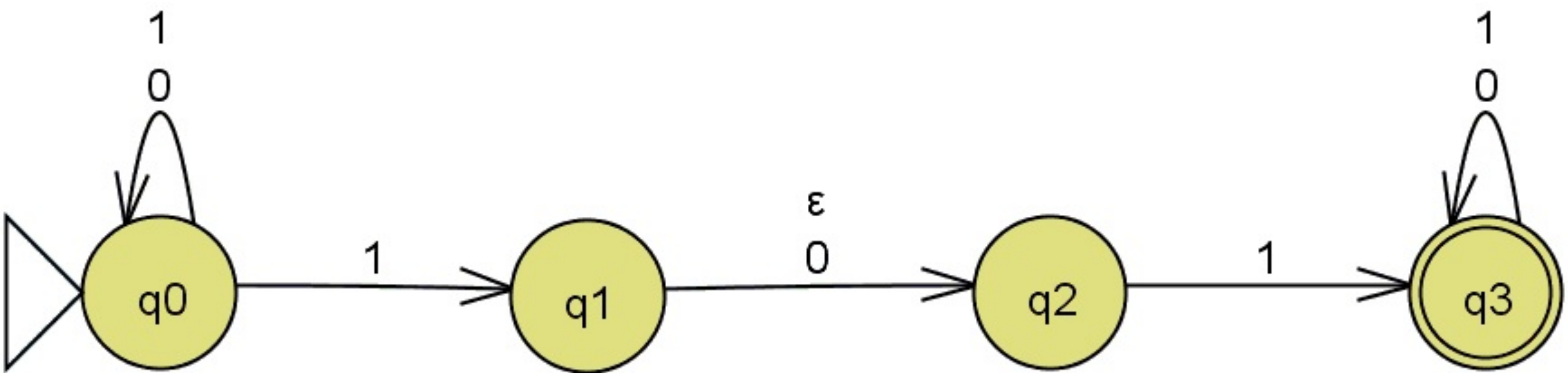
Sample NFA



- Non-deterministic finite automation (NFA)
  - ➢ A state may have zero, one or many exiting arrows for each alphabet symbol.
  - ➢ Arrows may be labeled with members of the alphabet or ε.
  - ➢ A DFA state has exactly one arrow for each input symbol and the symbols are only from the alphabet.

COMPUTATION WITH AN NFA

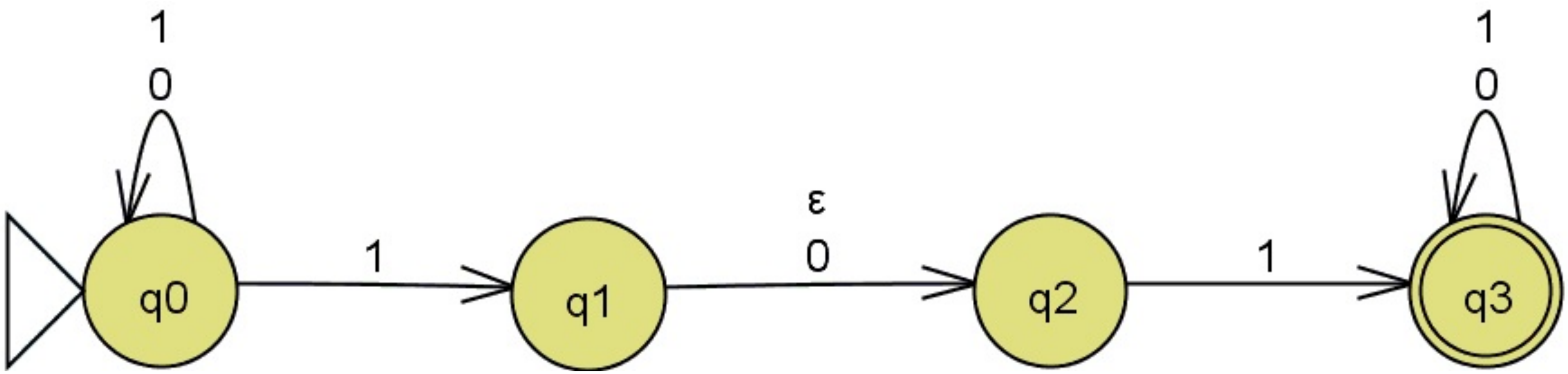
Computing Using an NFA



- A state with multiple paths for an input symbol
  - ➢ Before reading this symbol, the machine splits into multiple copies of itself and follows all the possibilities in parallel (*e.g.*, state q_0 with input symbol 1)
  - ➢ Each copy takes one of the possible ways to proceed
  - ➢ If there are subsequent choices the machine splits again
  - ➢ If a copy cannot accept the next input symbol, it dies
  - ➢ If any copy accepts the string, NFA accepts the string

Curtin University of Technology

Computing Using an NFA



- On reaching a state with an ε in an exiting arrow
  - ➢ Without reading any input, the machine splits into multiple copies one following each of the ε labeled arrows and one staying at the current state
  - ➢ Then the machine proceeds as before
- This NFA accepts all strings that contain either 101 or 11

Non-deterministic Computation

- Non-determinism is a kind of parallel computation
  - ➢ Process forking into several children
  - ➢ Each child process proceeds separately
  - ➢ If one of them accepts, the entire computation accepts

- For the NFA, each ‘process’ is instead a new machine.

- Practically speaking, we say that one machine can be in multiple states at the same time.

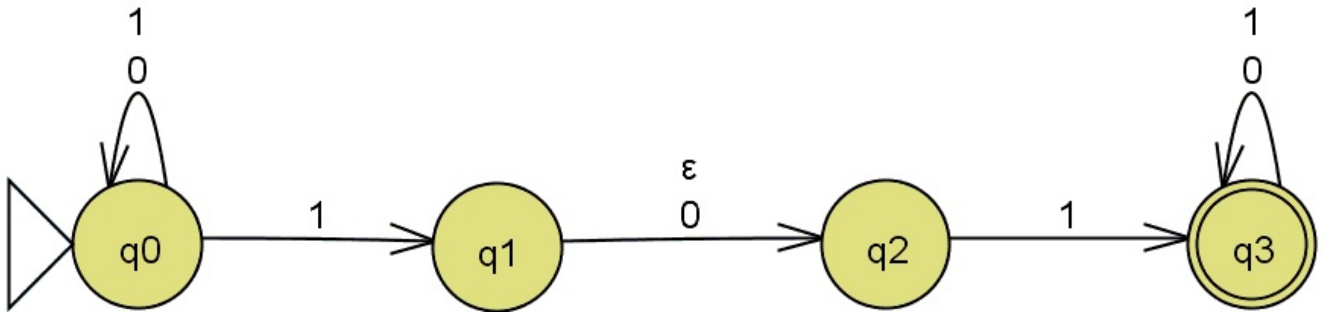
Curtin
University of Technology

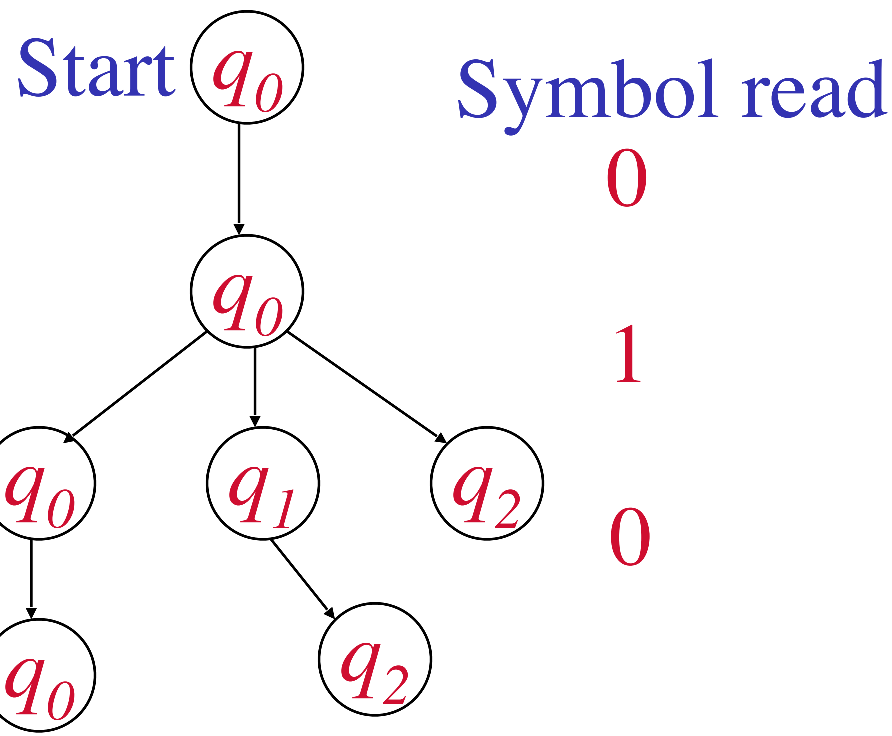
Non-deterministic Computation

- Another view: a tree of possibilities
  - ➢ Root of the tree corresponds to start of computation
  - ➢ A branch corresponds to a point with multiple choices
  - ➢ Machine accepts if at least one of the branches ends in an accept state

Example of NFA computation

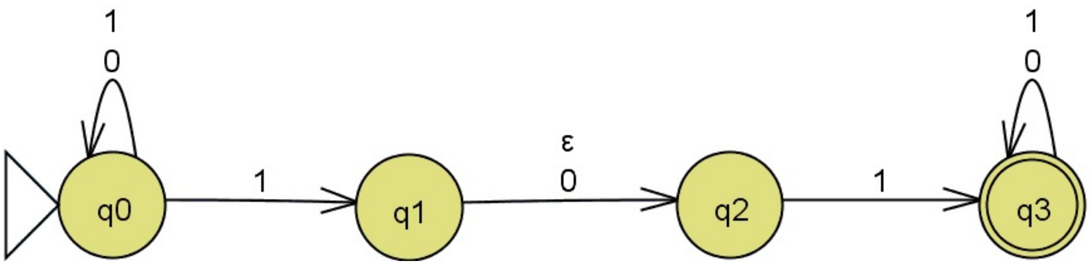
- Computation of *M* on input 010

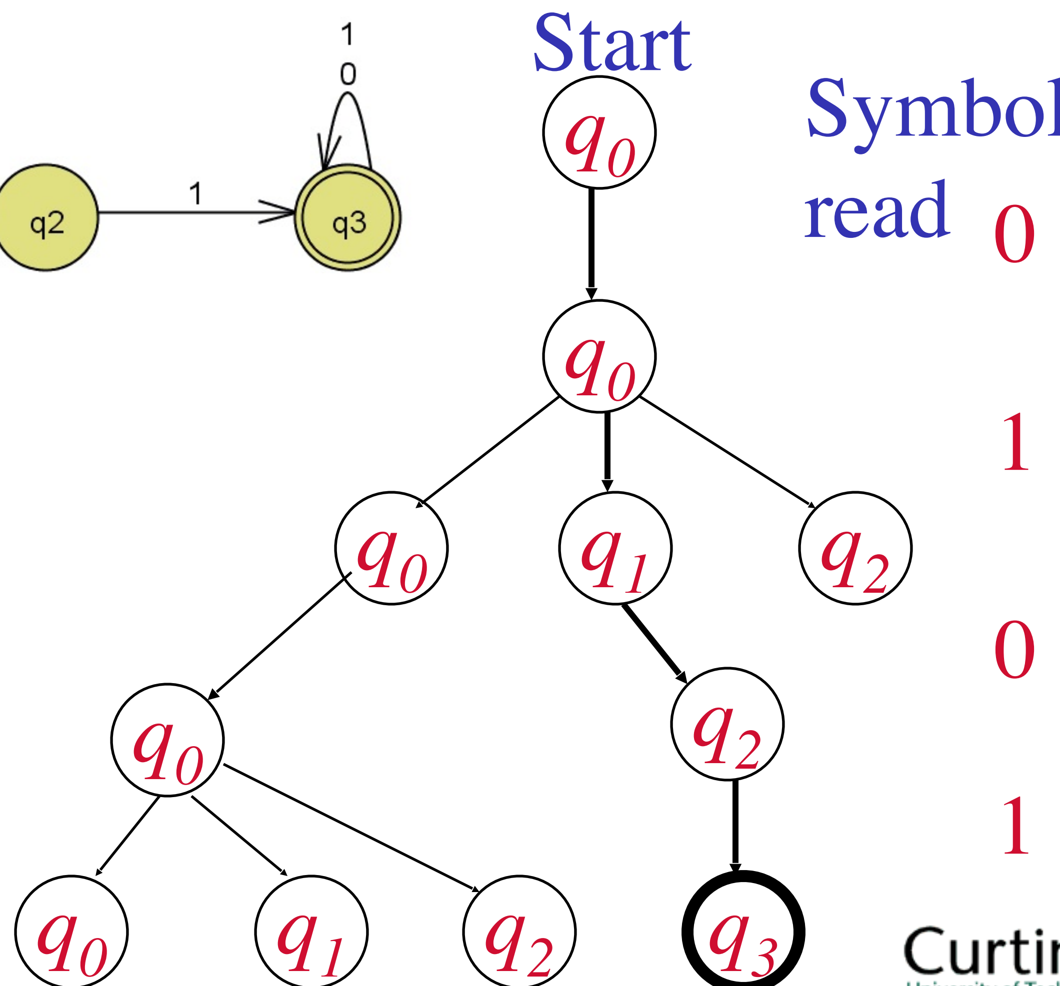




Example of NFA computation

- Computation of *M* on input 0101

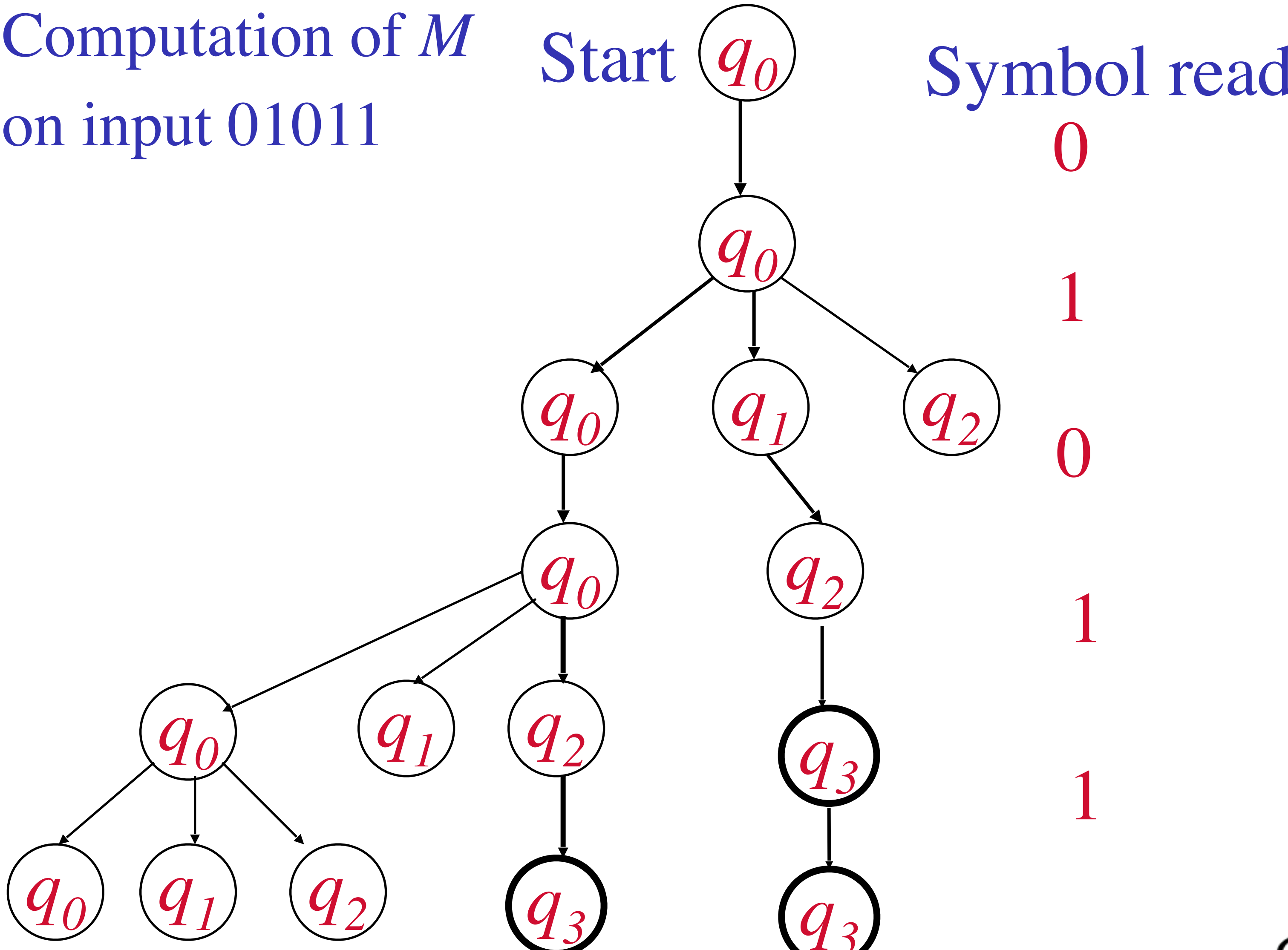




Curtin University of Technology

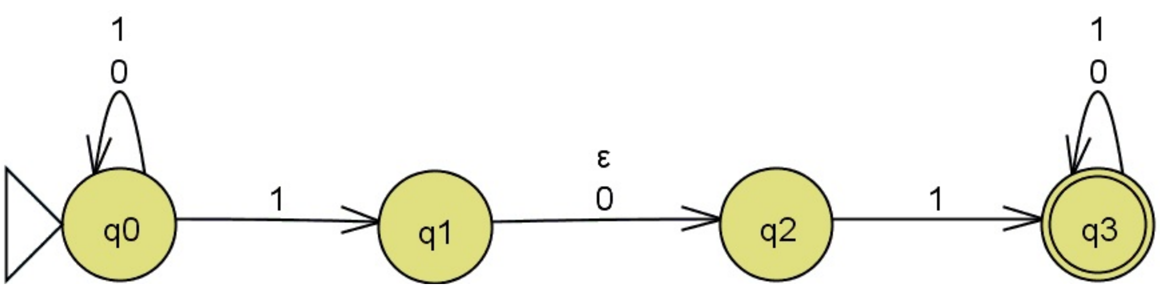
Example of NFA computation

- Computation of M on input 01011



Formal Definition of an NFA

- Similar to DFA except for transition function
  - In a DFA, the transition function takes a state and an input symbol and produces the next state.
  - In an NFA, the inputs are a state and either an input symbol or the empty string, and it produces a set of possible next states instead of a single next state.
  - The set of possible next states can be empty (ϕ). In this case the machine 'dies off'.



	0	1	ε
q_0	$\{q_0\}$	$\{q_0, q_1\}$	ϕ
q_1	$\{q_2\}$	ϕ	$\{q_2\}$
q_2	ϕ	$\{q_3\}$	ϕ
q_3	$\{q_3\}$	$\{q_3\}$	ϕ

Curtin University of Technology

Formal Definition of an NFA

- For any set Q, we write $P(Q)$ for the collection of all subsets of Q.
  - $P(Q)$ is the *power set* of Q.

- For any alphabet Σ, we write Σ_ε to be $\Sigma \cup \{\varepsilon\}$.

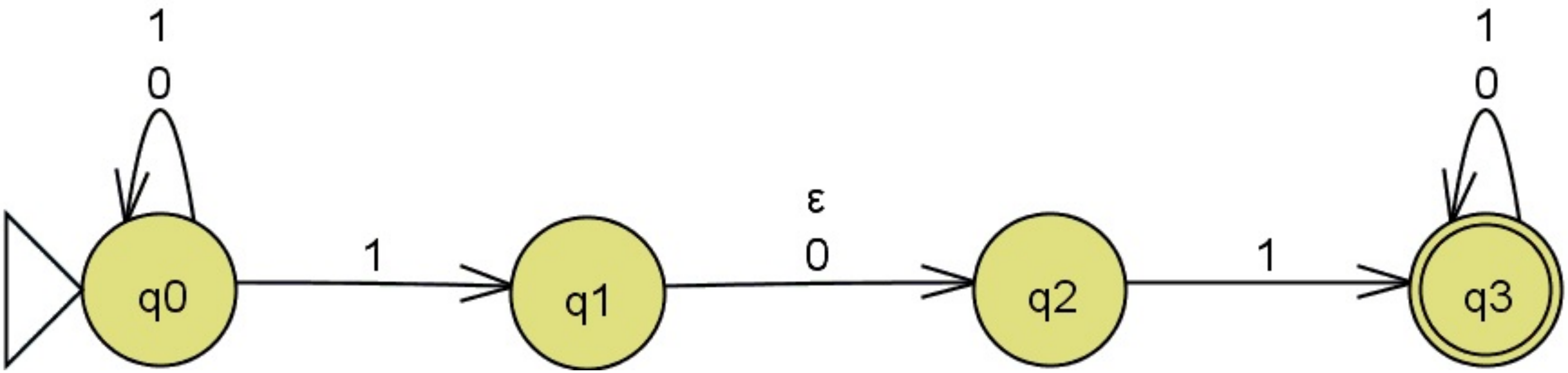
- $\delta : Q \times \Sigma_\varepsilon \rightarrow P(Q)$ is the type of *transition function*

Formal Definition of an NFA

- A non-deterministic finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where
  - Q is a finite set of *states*
  - Σ is a finite *alphabet*
  - $\delta : Q \times \Sigma_\varepsilon \rightarrow P(Q)$ is the *transition function*
  - $q_0 \in Q$ is the *start state*, and
  - $F \subseteq Q$ is the *set of accept states*.

Curtin University of Technology

Formal Definition of an NFA



- $M = (Q, \Sigma, \delta, q_0, F)$, where
- $Q = \{q_0, q_1, q_2, q_3\}$
- $\Sigma = \{0,1\}$
- δ is given as
- q_0 is the start state
- $F = \{q_3\}$

	0	**1**	**ε**
$\mathbf{q_0}$	$\{q_0\}$	$\{q_0, q_1\}$	ϕ
$\mathbf{q_1}$	$\{q_2\}$	ϕ	$\{q_2\}$
$\mathbf{q_2}$	ϕ	$\{q_3\}$	ϕ
$\mathbf{q_3}$	$\{q_3\}$	$\{q_3\}$	ϕ

Curtin University of Technology

DFA – NFA EQUIVALENCE

Properties of the NFA

- ***Assert***: Every NFA can be converted into an equivalent DFA

  - An NFA may be much smaller in number of nodes than its corresponding DFA

  - The functioning of an NFA may be easier to understand

Curtin University of Technology

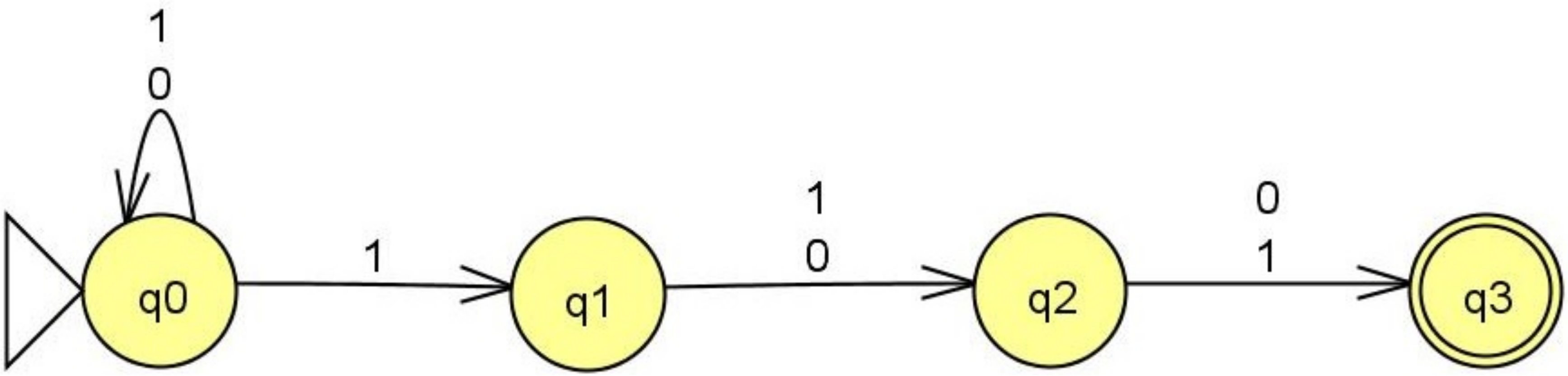
Theorem

- Example
  - ➢ Is there a similar construction for any DFA?
  - ➢ How do we know the two parse the same language

- Theorem: Every NFA has an equivalent DFA.
  - ➢ Two machines are equivalent if they recognize the same language

- Note an DFA is a NFA
  - ➢ Only have to show NFA to DFA

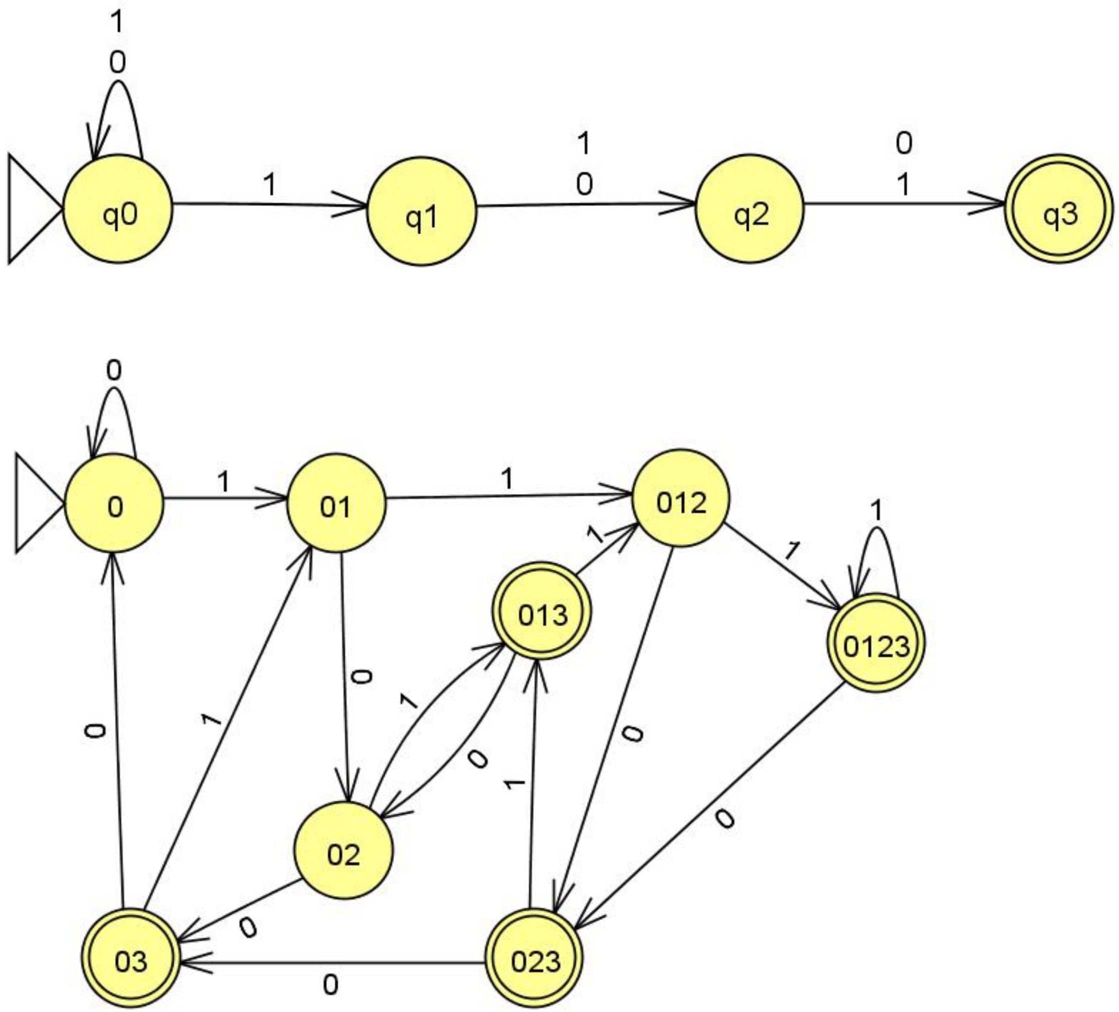
NFA Example



- Let *A* be a language of all strings over {0,1} containing a 1 in the third position from the end
- A good way to view the computation of this NFA:
  - ➢ It stays in state q0 until it "guesses" that it is 3 places from the end
  - ➢ Then if the input symbol is 1, it branches to q1 and uses q2 and q3 to "check" whether its guess was correct

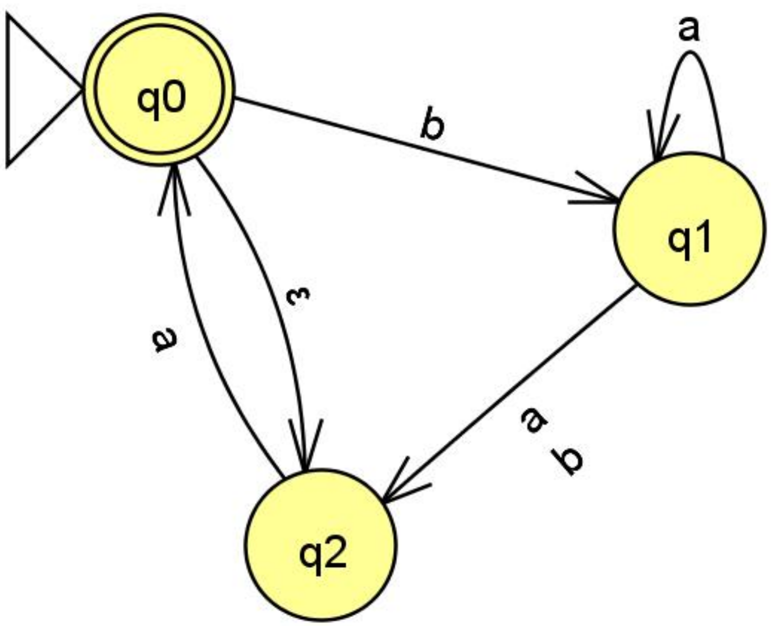
Equivalent DFA Example

- DFA has 8 states
  - ➢ each corresponds to the set of states of the NFA
  - ➢ Note that the NFA is always in state q0, so 0 is included in all DFA states.
  - ➢ The NFA accepts if in state q3, so any DFA state including 3 accepts.
  - ➢ The NFA starts in state q0, so the DFA starts in state 0.



Example: NFA N

- This example will be used to illustrate the formal proof
- Note, N
  - ➢ Accepts strings: ε, a, baba, baa, …
  - ➢ Does not accept strings: b, bb, babba, …
  - ➢ Next week look at characterizing the language



Curtin University of Technology

Proof idea

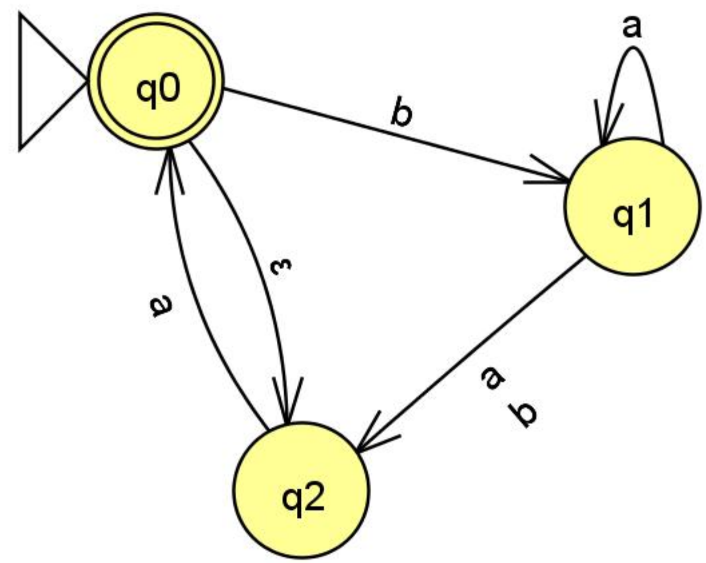
- Convert the NFA N into an equivalent DFA D that simulates the NFA.

- How to simulate the NFA by pretending to be a DFA ?
  - ➢ What do we need to keep track of when processing input string ? (The set of current states.)
  - ➢ Determine D's set of states. For k states of N, possible states are 2^k.
  - ➢ Need to decide the start and accept states.
  - ➢ Determine the transition function.

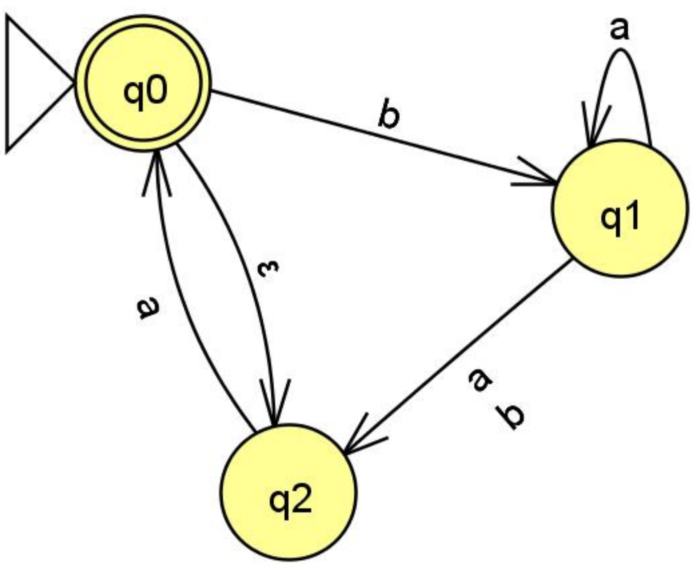
Curtin University of Technology

Example: NFA To DFA

- To construct a DFA *D* equivalent to *N*
- D's states:
  - {ϕ, {0}, {1}, {2}, {0,1}, {0,2}, {1,2}, {0,1,2}}
- Start state:
  - {0,2}
  - start state q0 of NFA plus the states reachable from q0 by ε transitions



Example: NFA To DFA



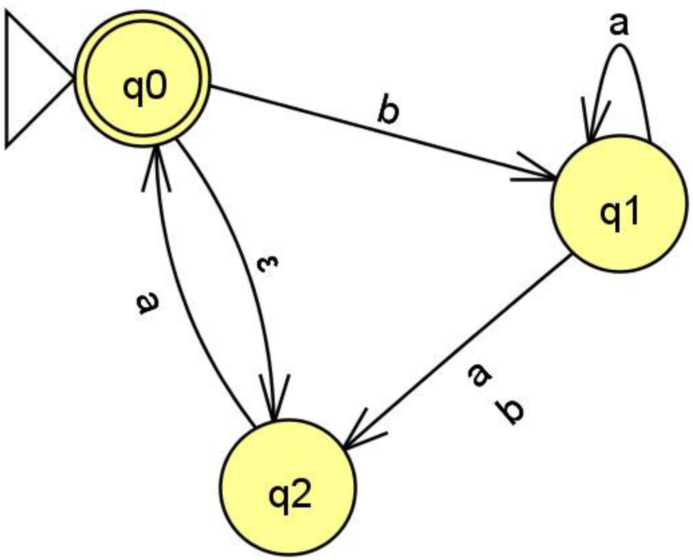
D's transition function:

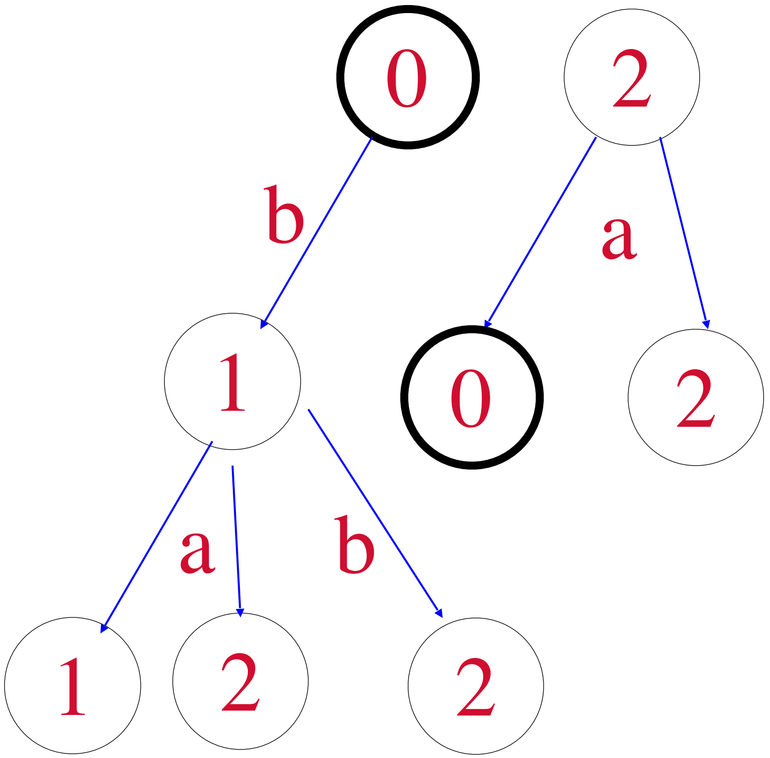
- From start state {0,2}, goes to {0,2} itself on input a; goes to {1} on b.
- From {1}, goes to {1,2} on a; goes to {2} on b.
- From {1,2}, goes to {0,1,2} on *a*; goes to {2} on *b*.
- From {2}, goes to {0,2} on *a*; goes to ϕ on *b*.
- From {0,1,2} goes to {0,1,2} on *a*; goes to {1,2} on *b*.
- From ϕ goes to ϕ on *a*,*b*.

- Potential accept states:
  - { {0}, {0,1}, {0,2}, {0,1,2}}
  - States containing N's accept states

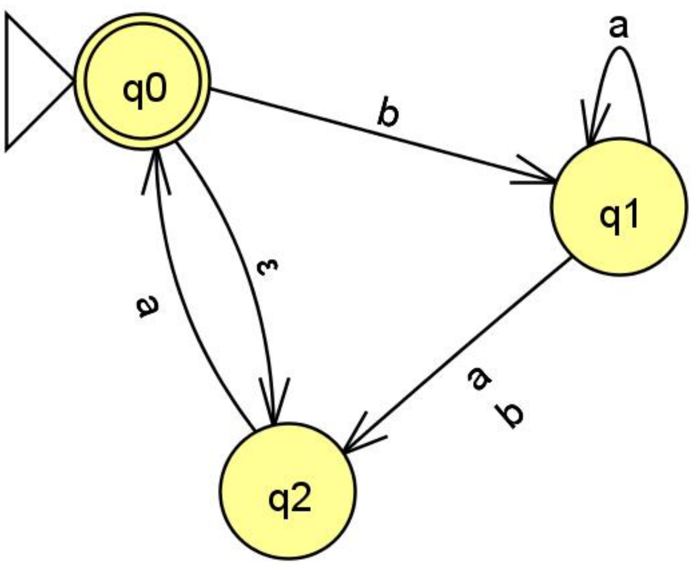
Curtin University of Technology

Example: NFA To DFA





Example: NFA To DFA



Result as a table

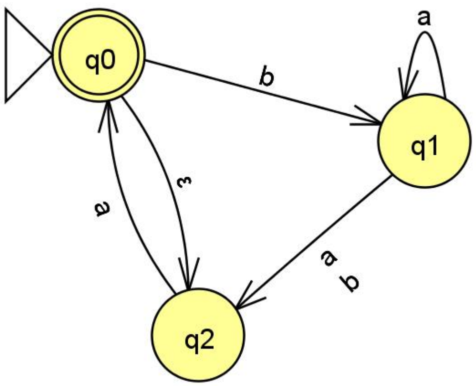
	a	b
{0,2}	{0,2}	{1}
{1}	{1,2}	{2}
{1,2}	{0,1,2}	{2}
{2}	{0,2}	ϕ
{0,1,2}	{0,1,2}	{1,2}
ϕ	ϕ	ϕ

Speed Up?

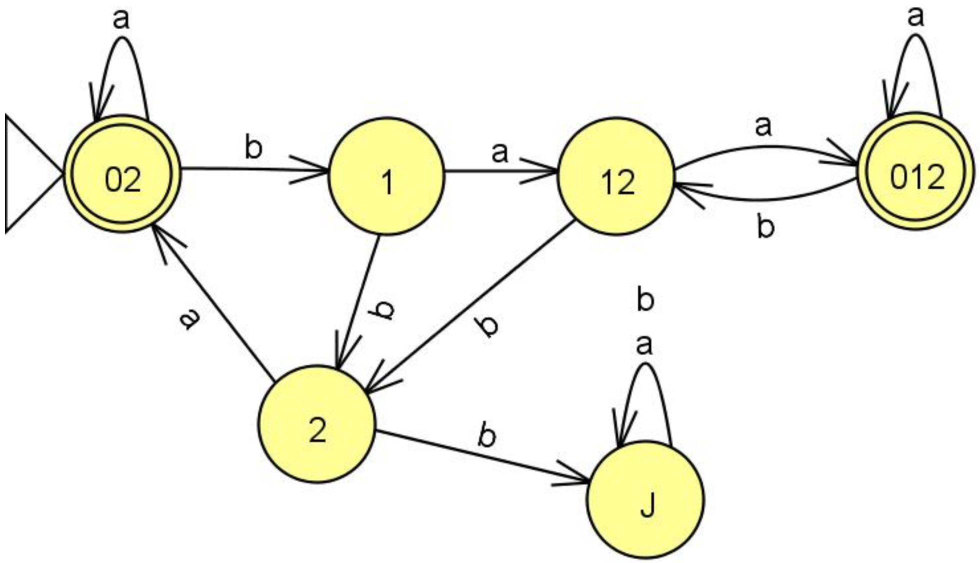
- Note: ϕ always gives ϕ
- Note
  - Systematic
  - Extra entries as needed
  - No need to list states that can never be entered
- The start state is the DFA start plus ε jumps
- Another method is to start with {1}, {2}, {3}.
  - Generate all other combinations using set union.
  - *e.g.*, the line for {1,2} is the union of the lines for {1} and {2}
  - Even using the first method, once both {1,2} and {2} are added to the table, it's faster to compute the line for {2} first.

	a	b
{0,2}	{0,2}	{1}
{1}	{1,2}	{2}
{1,2}	{0,1,2}	{2}
{2}	{0,2}	ϕ
{0,1,2}	{0,1,2}	{1,2}
ϕ	ϕ	ϕ

From NFA to DFA



- Actual accept states:
  - ➢ {{0,2}, {0,1,2}}



Equivalence of NFAs and DFAs

- Both recognize the same class of languages.
  - ➢ Surprising because NFAs seem more powerful

- NFAs useful
  - ➢ An NFA could be easier to describe than a DFA for a given language.

Curtin University of Technology

What You Need To Do

- Non-deterministism
  - ➢ Definition and characteristics of NFA
    - Learn definition
    - <ULO> Formal/Informal specification leads to State Diagram
    - <ULO> State Transition Function as alternative
- Computation
  - ➢ <ULO> Explain the operation of NFA on an input string.
  - ➢ <ULO> Prove that a string belongs to a language.
- Equivalence of NFA to DFA
  - ➢ <ULO> Convert one NFA into an equivalent DFA
  - ➢ Understand the consequences

Curtin University of Technology