Deterministic Finite State Automata
Part 1

Adam Wyner
CS3518, Spring 2017
University of Aberdeen

Recap

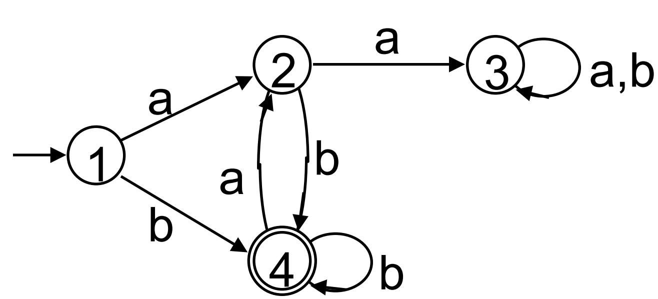
- Discussed of what was and what was not enumerable (countable) – can we find a bijective function between the counting numbers and some other set of objects.
- If yes, countable/enumerable; if not, uncountable/not enumerable.
- This is related to the issue of being to compute from some input to the output.
- In turn, this is related to defining and recognising languages (today).
- It extends to the issue of defining and recognising a logical language (last weeks of the course).

Recognising Languages

- We will tackle the problem of defining languages by considering how we could recognise them.
- Problem: Is there a method of recognising infinite languages? That is, given a language description and a string, is there an algorithm which will answer `yes' or `no' correctly?
- We will define an abstract machine which takes a candidate string and produces the answer `yes' or `no'.
- The abstract machine will be the specification of the language.

Finite State Automata

- A finite state automaton is an abstract model of a simple machine (or computer).
- The machine can be in a finite number of states. It receives symbols as input, and the result of receiving a particular input in a particular state moves the machine to a specified new state. Certain states are finishing states; if the machine is in one of those states when the input ends, it has ended successfully (or has accepted the input).
- Example A_1:



Formal Definition of FSAs

- We present here the special case of a Deterministic FSA (DFSA).
- It is `deterministic’ in that given an input, it leads to one and only one output (functional).
- As proven in CS2013, DFSAs can recognise the same set of languages as Nondeterministic FSAs (NDFSAs)

DFSA: Formal Definition

- A DFSA is a 5-tuple (Q, I, F, T, E) where:
  - Q = states, where Q is a finite set;
  - I = the initial state, where I is an element of Q;
  - F = final states, where F is a subset of Q;
  - T = an alphabet;
  - E = edges, the state changes, where E is a partial function from $Q \times T$ to Q (given as triples in the following).
- DFSA can be represented by a labelled, directed graph:
  - set of nodes (some final; one initial); and
  - directed arcs (arrows) between nodes; and
  - each arc has a label from the alphabet.

DFSA: Formal Definition (alternative)

- A DFSA is a 5-tuple ($\mathcal{Q}$, Σ, δ, q_0, $\mathcal{F}$) where:
- $\mathcal{Q}$ = a finite set of states;
- Σ = the alphabet;
- δ: $\mathcal{Q} \times \Sigma \rightarrow \mathcal{Q}$ is the transition function.
- $q_0 \in \mathcal{Q}$ = the initial state;
- $\mathcal{F} \subseteq \mathcal{Q}$ = the set of final (accepted) states;

Formal Definition of Example A_1

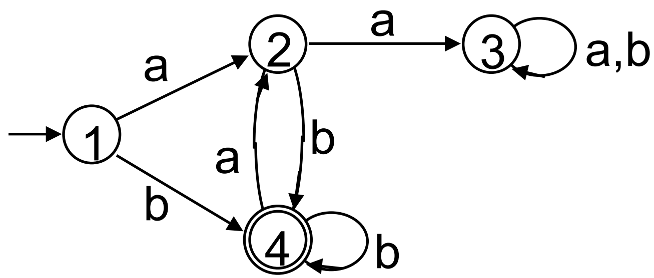
$Q = \{1, 2, 3, 4\}$

$I = \{1\}$

$F = \{4\}$

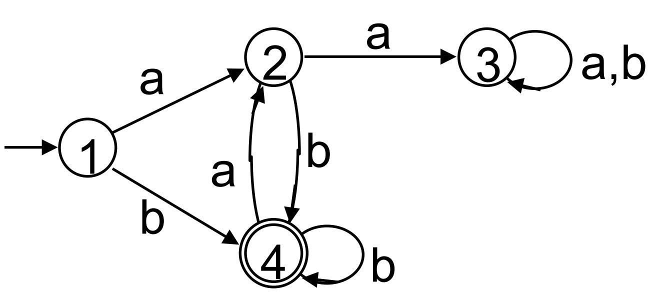
$T = \{a, b\}$

$E = \{(1,a,2), (1,b,4), (2,a,3), (2,b,4), (3,a,3), (3,b,3), (4,a,2), (4,b,4)\}$



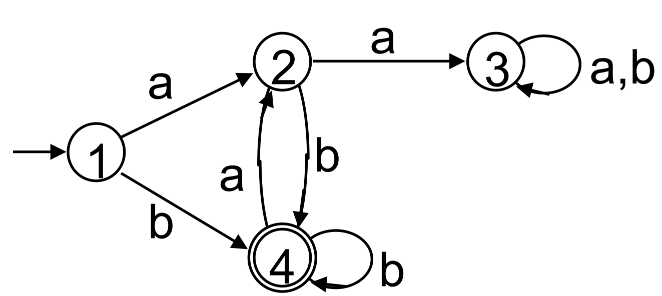
What is it to Accept a String/Language?

- For one state change: if (x,a,y) is an edge, x is its start state and y is its end state; a is read in to make the state change from x to y.
- For a series of state changes: a path is a sequence of edges such that the end state of one edge is the start state of the next edge.
- path p_1 = (2,b,4), (4,a,2), (2,a,3).



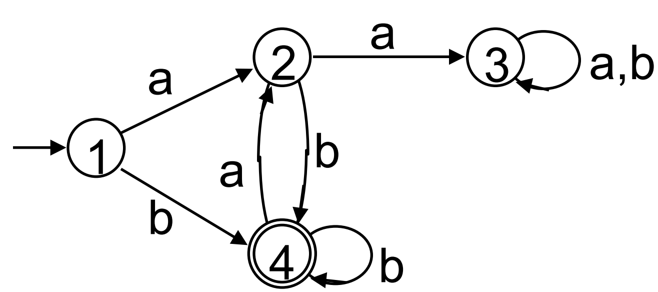
What is it to Accept a String/Language?

- A path is successful if the start state of the first edge is an initial state, and the end state of the last is a final state.
- path p_2 = (1,b,4),(4,a,2),(2,b,4),(4,b,4).
- The label of a path is the sequence of edge labels.
- path p_1 = (2,b,4), (4,a,2), (2,a,3).
- label(p_1) = baa.
- label(p_2) = babb.



What is it to Accept a String/Language?

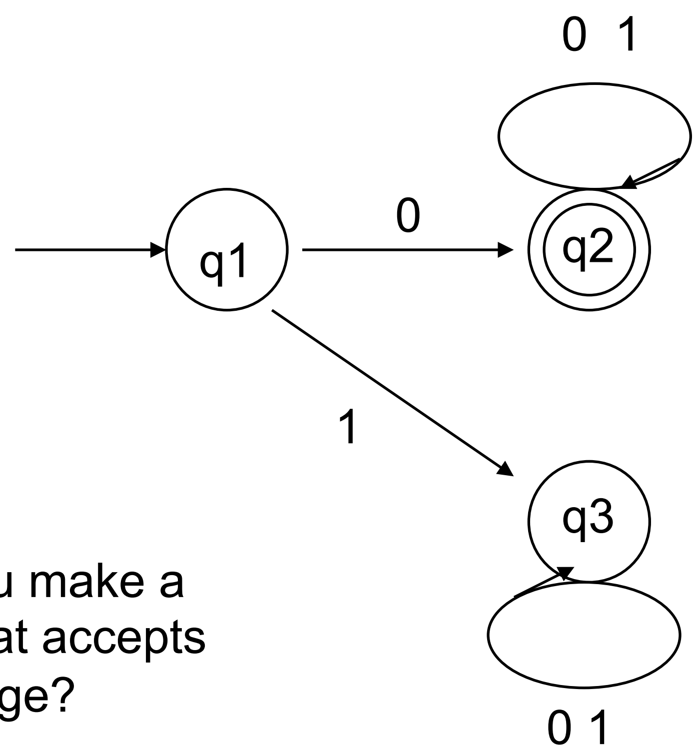
- A string is accepted by a DFSA if it is the label of a successful path.
- *babb* = label(p_2) is accepted by A_1.
- Let A be a DFSA. The language accepted by A is the set of strings accepted by A, denoted L(A).
- The language accepted by A_1 is the set of strings of a's and b's which end in b, and in which no two a's are adjacent.



Some Simple Examples (assuming determinism)

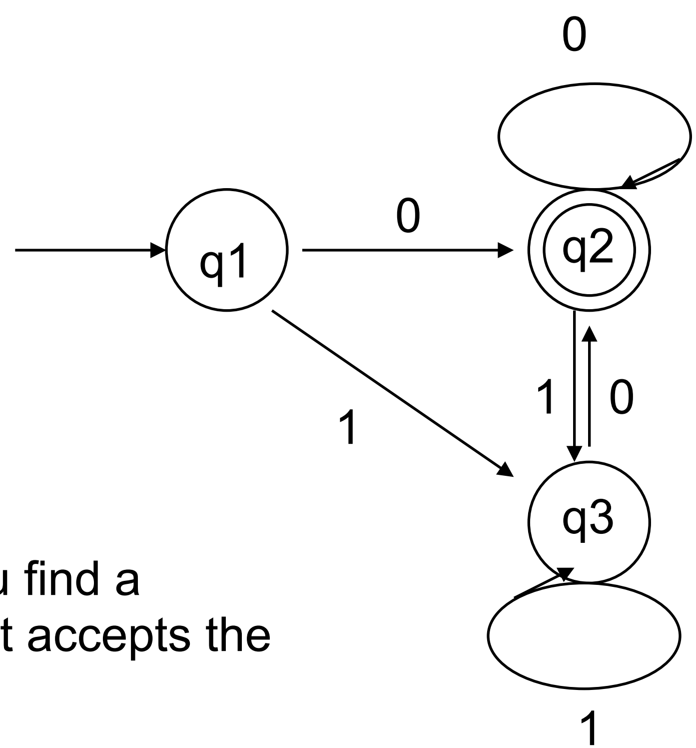
1. Draw a DFSA to accept the set of bitstrings starting with 0.
2. Draw a DFSA to accept the set of bitstrings ending with 0.
3. Draw a DFSA to accept the set of bitstrings containing sequence 00.
4. Draw a DFSA to accept the set of bitstrings containing both 1 and 0.
5. Can you draw a DFSA to accept the set of bitstrings that contain an equal number of 0 and 1?

L1: Bitstrings Starting with 0



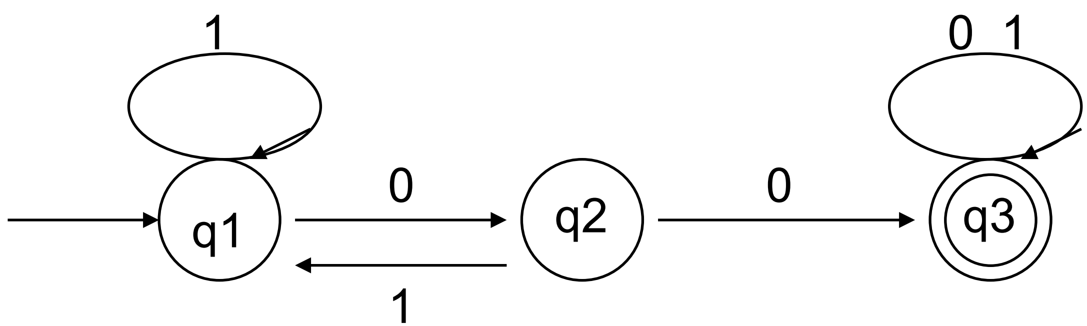
At home: can you make a smaller DFSA that accepts the same language?

L2: Bitstrings Ending with 0

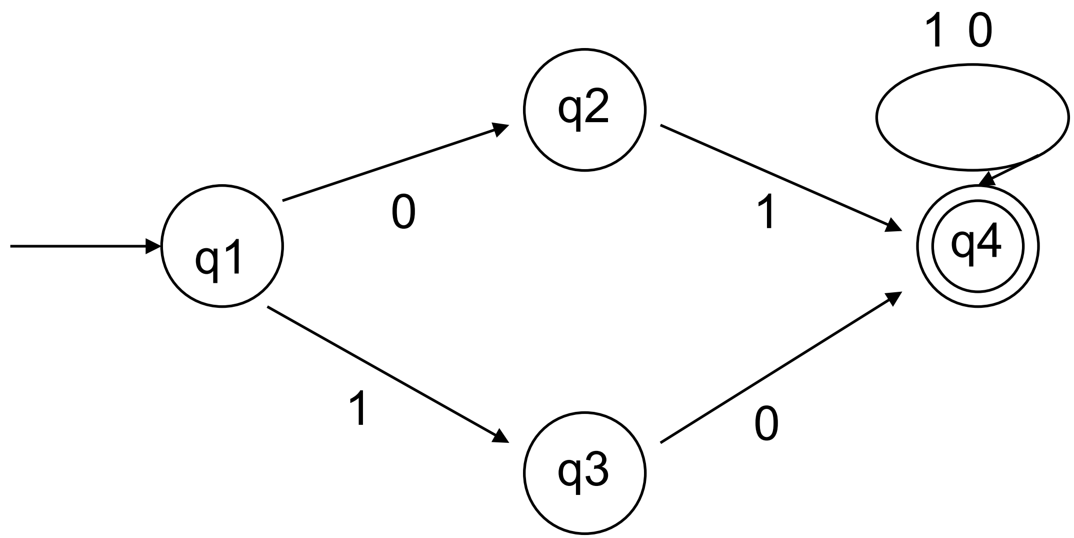


At home: Can you find a smaller DFSA that accepts the same language?

L3: Bitstrings Containing 00



L4: Bitstrings Containing Both 1 and 0



L5: Bitstrings Containing Equal 1 and 0

- This cannot be done.
- DFSAs are not powerful enough.
- See more in a couple of slides.
- Later we shall meet automata that can do it.
- And later, we will see problems that cannot be solved by any automaton.

Recognition Algorithm

- Problem: Given a DFSA, $A = (Q,I,F,T,E)$, and a string w, determine whether $w \in L(A)$.
- Note: denote the current state by q, and the current input symbol by t. Since A is deterministic, $\delta(q,t)$ will always be a singleton set or will be undefined. If it is undefined, denote it by $\perp$ ($\notin Q$).

- Reminder - $\delta(q,t)$ is the state change function, so leads to a q.

Recognition Algorithm

Add symbol # to end of *w (referred to as w*#).

q := initial state
t := first symbol of *w*#.

```
while (t ≠ # and q ≠ ⊥)
      begin
            q := δ(q,t)
            t := next symbol of w#
      end
return ((t == #) & (q ∈ F))
```

Minimum Size of DFSAs

- Let A = (Q, I, F, T, E). Definition:

  For any two strings x and y in T*, *x* and *y* are distinguishable with respect to A if there is a string z $\in$ T* such that exactly one of xz or yz are in L(A).

  z distinguishes x and y with respect to A.

- This implies (as we see) that with x and y as input, A must end in different states - A has to distinguish x and y in order to give the right results for xz and yz.
- This is used to prove the next theorem.

Minimum Size of DFSAs

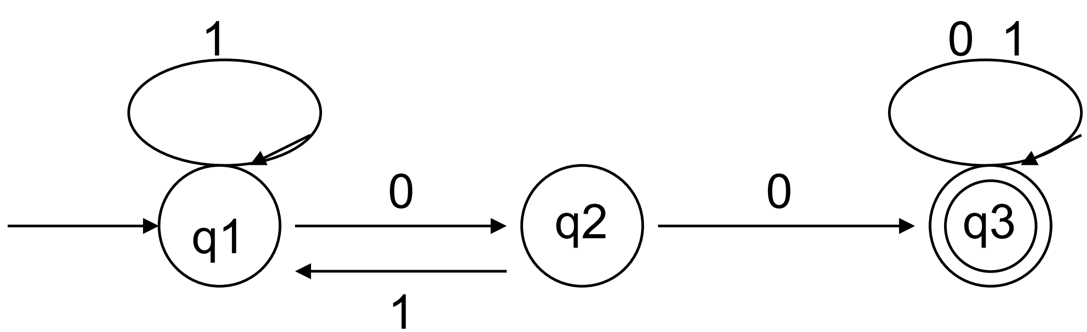
- Theorem: (proof omitted)

  Let $L \subseteq T^*$. If there is a set of n elements of T^* such that any two of its elements are distinguishable with respect to A, then any DFSA that recognises L must have at least n states.

- In other words, there are at least as many states as there are distinguishable elements of L.

Applying the Theorem (1)

- Consider L3 with the set of bitstrings containing 00.
- The set of elements {11,10,00} are distinguishable:
    - {11,10}: 100 is in L, 110 is out.
    - {11,00}: 001 is in L, 111 is out.
    - {10,00}: 001 is in L, 101 is out.
- {11,10,00} has 3 elements.
- Hence, the DFA for L3 requires at least 3 states.



Applying the Theorem (2)

- Consider L5 with equal numbers of 0 and 1. Try z = 01.
  - n=2: {01,001} → need 2 states
  - n=3: {01,001,0001} → need 3 states
  - n=4: {01,001,0001,00001} → need 4 states
  - …
- For any finite n, there's a set of n elements that are distinguishable.
- Yet, there are infinite strings of (potentially) equal 0 and 1; it is easy to see how to extend this example.
- Therefore, the DFSA for L5 would need more than finitely many states, (which is not permitted)!

A Taste of the Theory of Formal Languages

- This theorem tells you something about the kind of automaton (in terms of its number of states) that's required given a particular kind of problem (i.e., a particular kind of language)
- It also tells you that certain languages cannot be accepted by any DFSA.
- In other words, it indicates that there 'problems' that cannot be 'solved' by any DFSA.
- It addresses this by seeing 'problems' as strings of a language and 'solutions' as ways to (not) accept them.

Automata with Output

- We now have ways to formally define languages, and ways to automatically test whether a given string is a member of a language.
- We also have a simple model of a computer, which is done by adding output to string recognition.
- We extend DFSAs, so that instead of simply replying "yes" or "no" when presented with input, our abstract machine writes some output from an alphabet, determined by the input string.