Mathematics for Computing Science (CS2013)

Formal Languages and Finite State Automata

Finite State Automata (FSA)

- Very simple kind of computer
  - No software or hardware
  - Rudimentary functionality
    1. Read one input characters
    2. Update a state of computation
- We present a formal/mathematical definition (see next)

Formal definition of Finite State Automata (FSA)

A Finite State Automaton (FSA) A is a 5-tuple (Q, I, F, T, E) where:

- $Q = \{q_1, q_2, \ldots, q_n\}$ is the finite **set of states**
- $I \subseteq Q$, $I \neq \varnothing$, is the **set of initial states** (a nonempty subset of Q)
- $F \subseteq Q$, is the **set of final states** (a subset of Q)
- $T = \{a_1, a_2, \ldots, a_m\}$, is an **alphabet of symbols** a_i $1 \leq i \leq m$
- $E \subseteq Q \times (T \cup \{\lambda\}) \times Q$, is the set of edges (subset of Cartesian product)

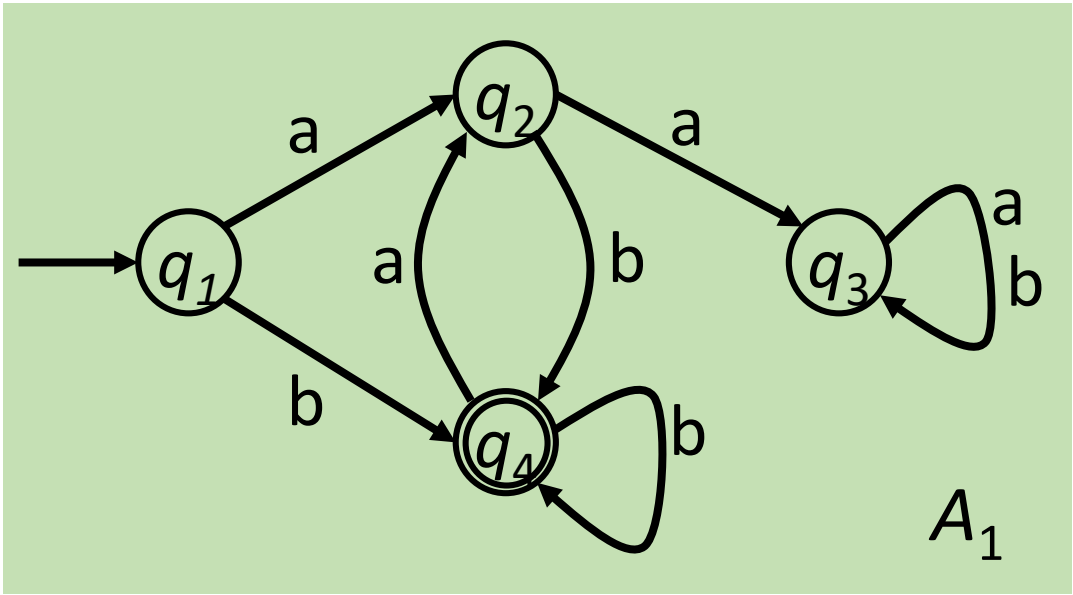
Formal definition of Finite State Automata (FSA)

Example FSA $A_1 = (Q, I, F, T, E)$ formally defined:

- $Q = \{q_1, q_2, q_3, q_4\}$
- $I = \{1\}$
- $F = \{4\}$
- $T = \{\text{a}, \text{b}\}$
- $E = \{(q_1, \text{a}, q_2), (q_1, \text{b}, q_4), (q_2, \text{a}, q_3), (q_2, \text{b}, q_4), (q_3, \text{a}, q_3), (q_3, \text{b}, q_3), (q_4, \text{a}, q_2), (q_4, \text{b}, q_4)\}$

Finite State Automata (FSA)

- Formal definition has a more intuitive visual/graphic representation
  - It resembles a graph
  - Each state is a node/vertex; states are connected via edges/arcs
  - Edges/arcs have a label
    - Label indicates the input character we need in order to follow the edge
  - Start states are indicated by incoming arrow; final states indicated by double circles
  - Previous example:



Paths in a Finite Automaton

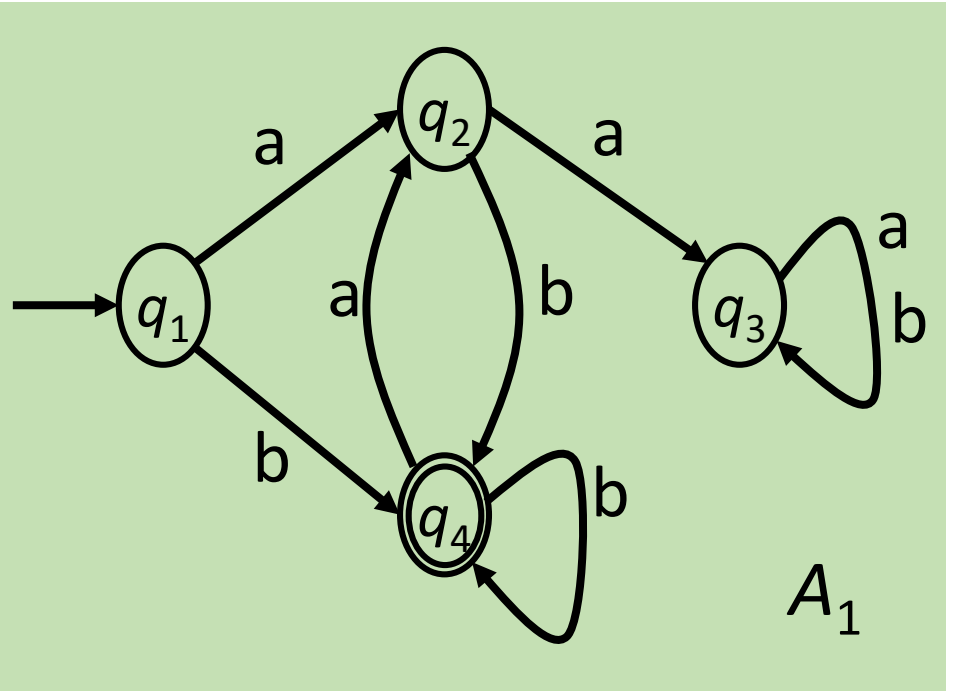
- If (x, a, y) is an edge, x is its **start** state and y is its **end** state
  - Notice: x, a, and y stand for **variables** (any values of states and labels)
- A **path** is a sequence of edges $\langle e_1, e_2, \ldots, e_n \rangle$, $e_i = (x_i, a_i, y_i)$, $1 \leq i \leq n$, such that $y_{j-1} = x_j$, $1 < j \leq n$, (end state of e_{j-1} is the start state of e_j)
  - Example: $p_1 = \langle (q_2, \text{b}, q_4), (q_4, \text{a}, q_2), (q_2, \text{a}, q_3) \rangle$
- A **successful path** $\langle e_1, e_2, \ldots, e_n \rangle$, $e_i = (x_i, a_i, y_i)$, $1 \leq i \leq n$, is one in which $x_1 \in I$ (start state of first edge is an initial state), and $y_n \in F$ (end state of last edge is a final state)
  - Example: $p_2 = \langle (q_1, \text{b}, q_4), (q_4, \text{a}, q_2), (q_2, \text{b}, q_4), (q_4, \text{b}, q_4) \rangle$
- The **label of a path** $p = \langle e_1, e_2, \ldots, e_n \rangle$, $e_i = (x_i, a_i, y_i)$, $1 \leq i \leq n$, $label(p)$, is the sequence of edge labels $\langle a_1, a_2, \ldots, a_n \rangle$
  - Example: $label(p_1) = \text{baa}$

Accepted strings and languages of FSAs

- A string is **accepted** by a FSA if, and only if, it is the label of a successful path
  - Example: babb is accepted by A_1, because label(p_2) = babb
- The **language accepted by** FSA *A* is the set of all strings accepted by A, represented as $L(A)$
- What's the language of A_1?

Answers (in increasing order of precision):

1. Stuff, like.
2. Strings of a's and b's
3. Strings of a's and b's which end in b, and in which no two a's are adjacent



Putting an FSA to work

- A computational mechanism to "run" an FSA
  - Inputs: an FSA definition A and a string w
  - Output: yes ($w \in L(A)$) or no ($w \notin L(A)$)

algorithm *RecogniseString*
input: an FSA $A = (Q, I, F, T, E)$ and a string $w = a_1, a_2, \ldots, a_n$

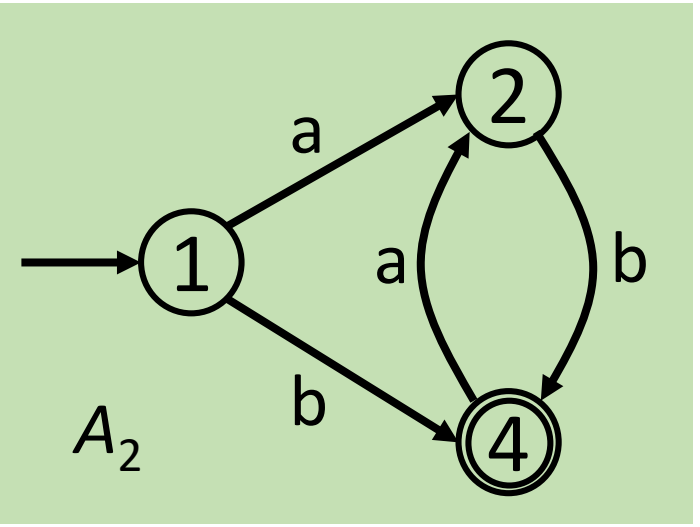
```
select q ∈ I                    % choose an initial state (non-determinism)
for i = 1 to n do               % for each symbol a_i from string w
    select (q, a_i, q′) ∈ E     % choose an edge (non-determinism)
    q ← q′                      % update state q to new state q′
if q ∈ F                        % if last state is a final state
    output yes                  % w ∈ L(A)
else                            % otherwise
    output no                   % w ∉ L(A)
```

Non-determinism and determinism

- Two steps in algorithm used “select” (choose one of many)
- This is **non-determinism**
  - It really means “there are choices” (no explanation on how to best choose)
  - Formal definition allows, for instance, (q_1, a, q_2) and (q_1, a, q_3)
- What if we make the wrong choice?
  - Solution: we must try all choices **exhaustively**
- Algorithm must be run repeatedly to try out
  - All initial states and every possible edge (and their combinations)
- Non-deterministic algorithms/solutions are easier to describe
  - They are inefficient, though, as they require exponential time/memory

Non-determinism and determinism (cont'd)

- Deterministic FSA (DFSA):
  - There is only one initial state, that is, $|I| = 1$ and
  - For all states $s \in Q$ there is at most one edge $(s, a, s') \in E$ for any symbol $a \in T$
  - *RecogniseString* algorithm works very efficiently (linear) if input A is a DFSA
- Important: we don't require a comprehensive set of edges
  - If there are "missing" edges (e.g., string aa in FSA below); we may get "stuck" in a state…
- Convention:
  - If FSA is in a state and there is no edge for the current input symbol then we **stop** and **reject the string**
  - This is also the case even if we are in a final state
  - In light of this convention, what is $L(A_2)$?



Non-determinism and determinism (cont'd)

- Notice that we allow transitions/edges with λ
  - $E \subseteq Q \times (T \cup \{\lambda\}) \times Q$
- What does a λ-transition mean?
  - We follow that transition without "consuming" symbols from string
  - In terms of our algorithm, the "for-loop" does not increment
- Try at home:
  - Adapt/extend algorithm to handle non-determinism
  - Adapt/extend algorithm to handle λ-transitions

What kind of questions with FSAs?

These are some of the questions in connection with FSAs:

- Given a language description, provide an FSA to accept the language
  - Only strings which should be accepted must be accepted;
  - Strings which should not be accepted must not be accepted
- Given an FSA what language does it recognise?
  - Language description must be as precise as possible
- Given a partial FSA and a language, complete missing parts
- Given two FSAs, what the intersection of their languages is
- And so on…

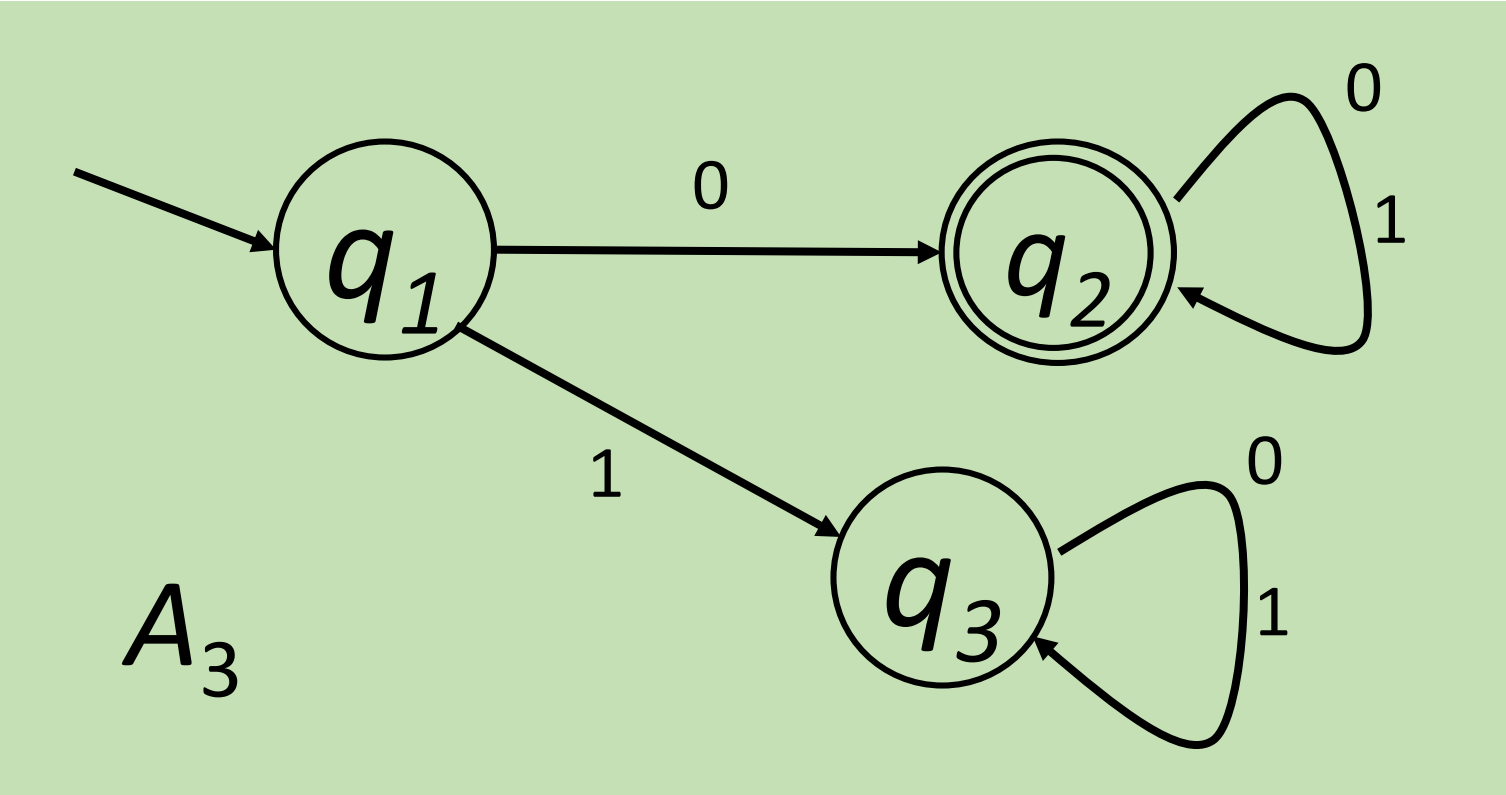
Let’s work together

Let’s draw **deterministic FSAs** for the following languages

1. All strings of 0s and 1s **starting with 0**
2. All strings of 0s and 1s **ending with 0**
3. All strings of 0s and 1s **containing a sequence 00**

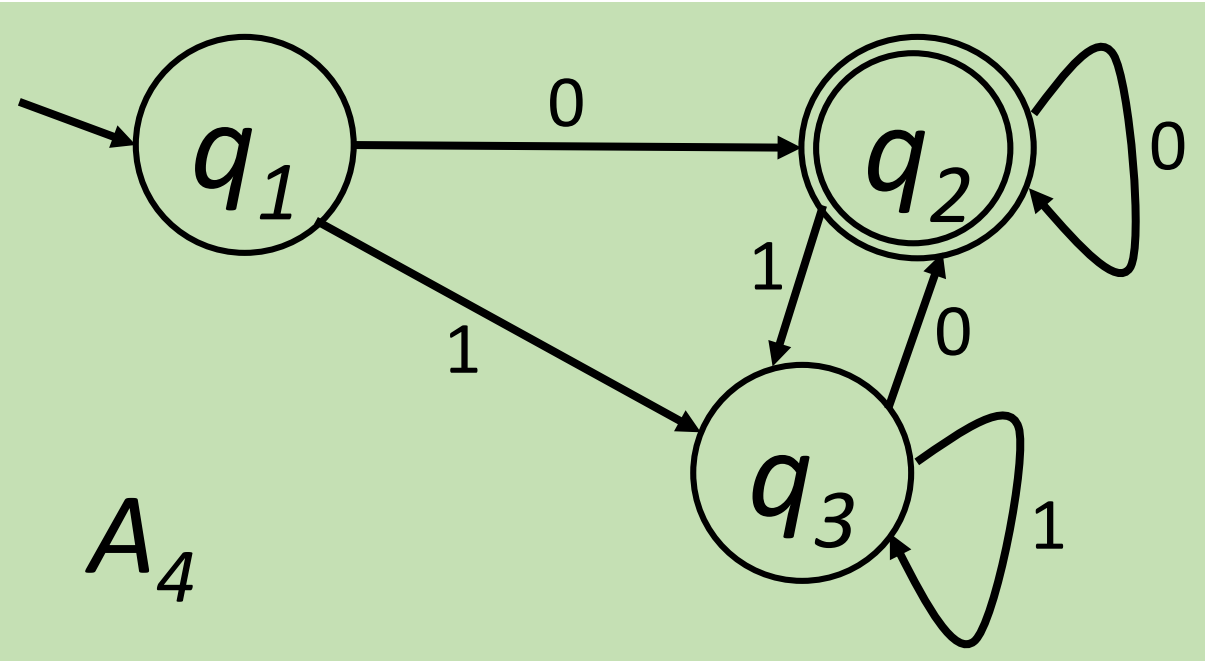
Is it possible to draw a deterministic FSA to accept all strings of 0s and 1s that contain an equal number of 0s and 1s (in any order)?

All strings of 0s and 1s starting with 0



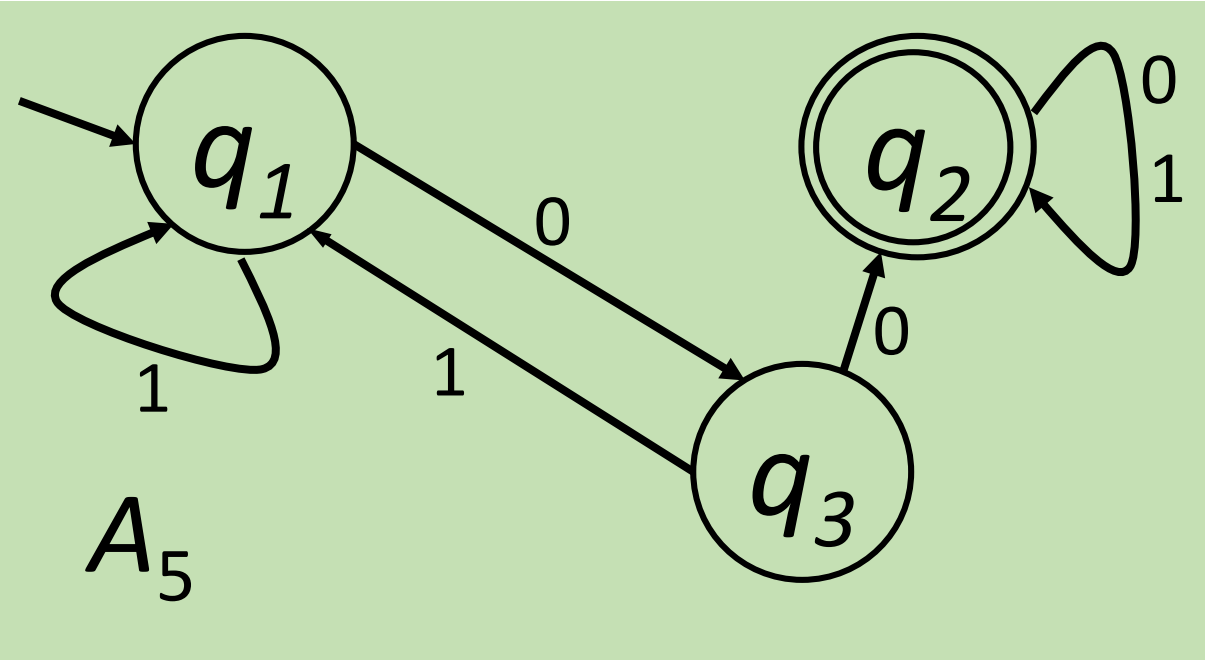
- Clear cases: strings which start with 0 and with 1; two states needed
- The strings we want to accept start with 0, so q_2 is a final state
- Provisions for any subsequent symbol (loops in q_2 and q_3)
- Is there a smaller FSA to accept the same language (i.e., with fewer states)?

All strings of 0s and 1s ending with 0



- Attempt to re-use previous solution…
- Splitting initial cases wasn't necessary
- Is there a smaller FSA to accept the same language?

All strings of 0s and 1s containing 00



Check-list

- Is there an initial state and at least one final state?
- Do all states have 2 edges (one for each symbol)?

Strings with equal numbers of 0 and 1

- These are strings like 01, 0011, 0101, 1010, 000111, 101010, etc.
  - Characters may appear in any order
  - We need to “count” the number of 0s and 1s
- FSAs cannot “count” – there is no FSA to recognise this language
- There are more expressive/powerful “devices” to handle this
  - We’ll cover these later in the course
- There are languages that cannot be recognised by any device
  - This is the limit of computation (or computability)
  - There is no point in waiting for the next generation of computers…

Minimum size of FSAs

- For any two strings x, $y \in T^*$, x and y are **distinguishable** with regards to FSA A if there is a string $z \in T^*$, such that exactly one of xz or yz are in $L(A)$
  - We say that z **distinguishes** x and y with respect to A
- This means 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 or yz
- Theorem: Let $L \subseteq T^*$. If there are n elements of T^* such that any two are distinguishable with respect to A, then any FSA that recognises L must have at least n states

Applying the theorem

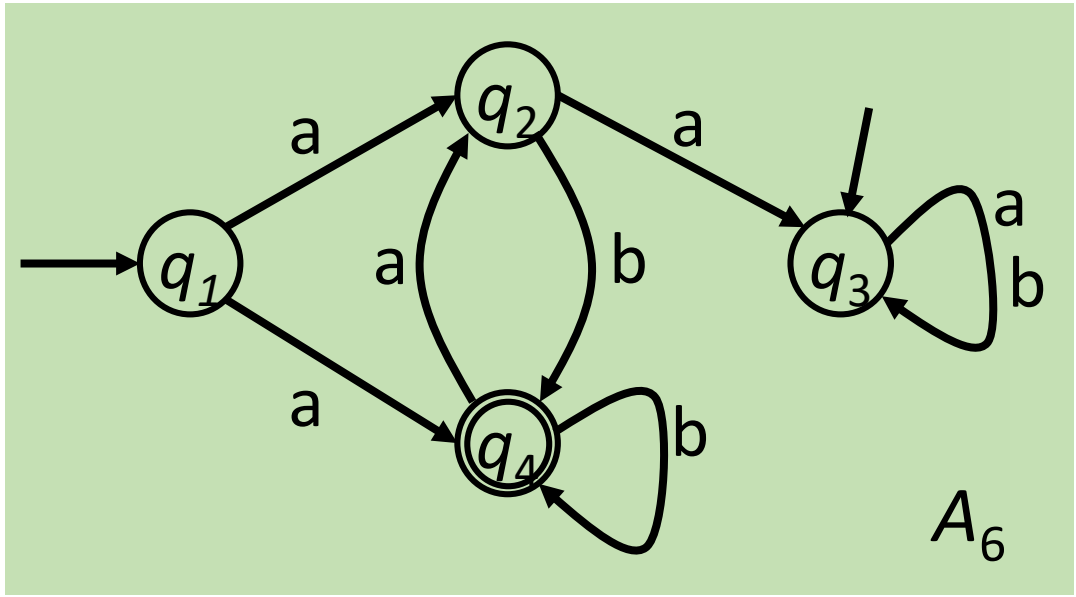
- Let L_5 be the set of strings of 0s and 1s containing 00 (from A_5 above)
- Distinguishable: all of {11,10,00}
  - {11,10}: 100 is in L, 110 is out
  - {11,00}: 001 is in L, 111 is out
  - {10,00}: 001 is in L, 101 is out.
- {11,10,00} has 3 elements, hence FSA for L_5 requires **at least 3 states**
- If you were not able to find an FSA with fewer states, blame this theorem…

Applying the theorem (Cont'd)

- Let L_5 be the set of strings of 0s & 1s with equal number of 0s & 1s
  - For n = 2, we have {01,001} and we need 2 states
  - For n = 3, we have {01,001,0001} and we need 3 states
  - For n = 4, we have {01,001,0001,00001} and we need 4 states
- For any finite n, there are n elements that are distinguishable
- Hence the FSA for L_5 would need an infinite number of states
  - This is not permitted by our definition of FSA

Exploiting non-determinism

- The definition of an FSA allows nondeterminism
  - Choice of initial state to start from
  - Choice of edge/transition to follow
  - Following λ-transitions (which do not depend on input string)
- Consider the following FSA below: abab $\in L(A_6)$?
  - No, if we follow $q_3q_3q_3q_3q_3$ (loop in q_3)
  - Yes, if we follow $q_1q_2q_4q_2q_4$
  - Yes, if we follow $q_1q_4q_4q_2q_4$



Equivalence of DFSAs and NDFSAs

- Theorem: A language *L* is accepted by an NDFSA if, and only if, *L* is accepted by a DFSA
  - NDFSA and DFSA have equal computational power
  - Choice of which one to use is for convenience
- The proof of the theorem is an algorithm to convert NDFSAs onto DFSA which preserve the language it recognises

Summary

- Formal definition of FSA
- How definition enable us to check if a string belongs to a language
- Graphical representation of FSA
- Non-determinism vs. determinism
- Important results (theorems)

Further reading

- Chapters 0 and 1 of “Introduction to the theory of computation”, by Michael Sipser (there are copies in the library)
- Chapters 1 and 2 of “An Introduction to Formal Languages and Automata”, by Peter Linz (PDF available on-line)