Mathematical Concepts (G6012)

Computing Machines

Thomas Nowotny
Chichester I, Room CI-105
Office hours: Mondays 10:00-12:00
T.Nowotny@sussex.ac.uk

Last time: Regular languages

- A regular language can be defined like this (over an alphabet $\mathcal{S}$):
  - The empty language is regular
  - The singleton language $\{a\}$ is regular ($a \in \mathcal{S}$)
  - If $\mathcal{A}$ and $\mathcal{B}$ are regular languages, then $\mathcal{A} \cup \mathcal{B}$ (union) and $\mathcal{A} \circ \mathcal{B}$ (concatenation) and $\mathcal{A}*$ (Kleene star) are regular.
  - No other languages are regular

BB Example: Determine whether a language is regular

- Take the Alphabet $\mathcal{S} = \{a\}$ and language $\mathcal{L} = \{a, aa\}$
- Is $\mathcal{L}$ a regular language?
- Need to show that it can be constructed by legal operations ($\circ, \cup, *$) from (a) regular language(s)
- Start: Singleton language $\mathcal{A} = \{a\}$ is regular by definition
- The language $\mathcal{B} = \{aa\}$ can be generated as $\mathcal{B} = \mathcal{A} \circ \mathcal{A}$
- Finally, $\mathcal{L} = \mathcal{A} \cup \mathcal{B}$
- This proves that $\mathcal{L}$ is a regular language.

Regular expressions

- Regular expressions can be used to define regular languages
- A regular expression describes the legal word in a language by a matching operation:

Regular expressions

– ‘*a*’ matches the symbol ‘a’ in the alphabet
– The ‘|’ denotes alternatives (Boolean (x)or)
– Brackets ‘(‘ and ‘)’ are used for grouping

– ‘*’ matches zero or more of the preceding symbol
– ‘+’ matches one or more of the preceding symbol
– ‘?’ matches 0 or 1 of the preceding symbol

Precedence of operators

Precedence	Operator
Highest	( )
Medium	? * +
Lowest	|

FINITE STATE AUTOMATA (FSA)

Introduction

- FSA are examples of a model of computing or an (abstract) computing machine
- Models of computing are how computer scientists make sense of the world
- Many models of computing have been suggested
- FSA are in a sense the most simple ones

FSA: General Characteristics

- Discrete inputs & possibly outputs
- System in one of a finite number of internal configurations
- State encodes information about all past inputs needed to determine behaviour of system on subsequent inputs

Typical example 1

- Control mechanism of an elevator
- Input is requests for service
- State is current floor & direction of motion
- Does not record history of satisfied requests
- Unsatisfied input is unordered collection of requests

Typical Example 2

- Lexical Analysis:
  Transform a string of characters into a sequence of (legal) tokens:

"x+501 = foo"

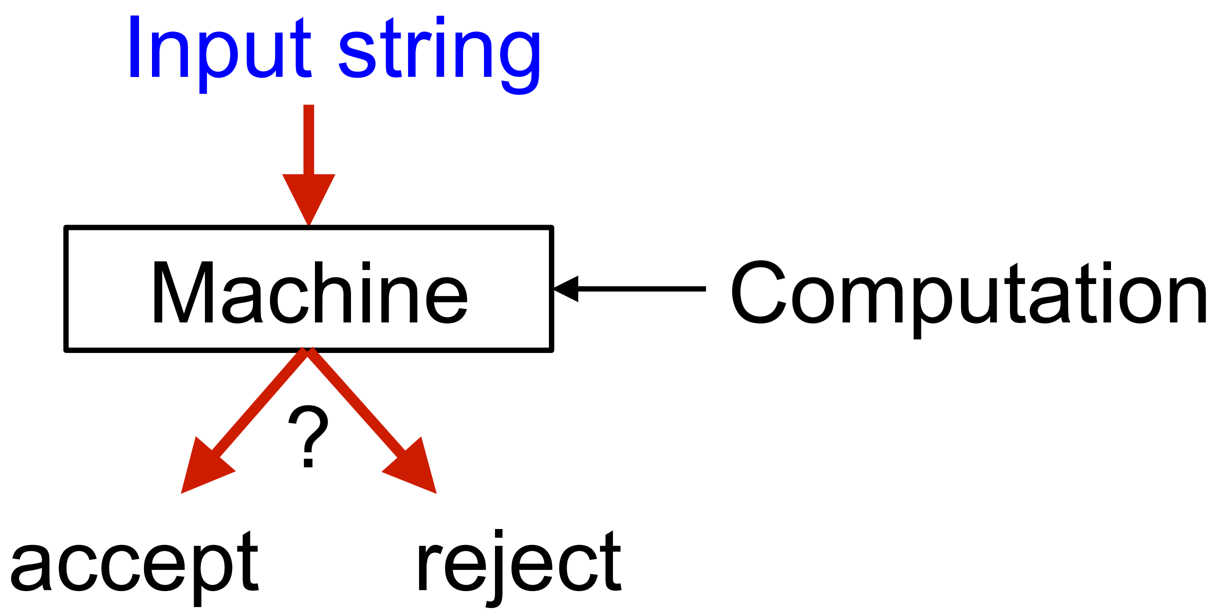
↓

(id(x),plus,num(501),equals,id(foo))

(This is something a compiler needs to do)

Focus on (language) parsing

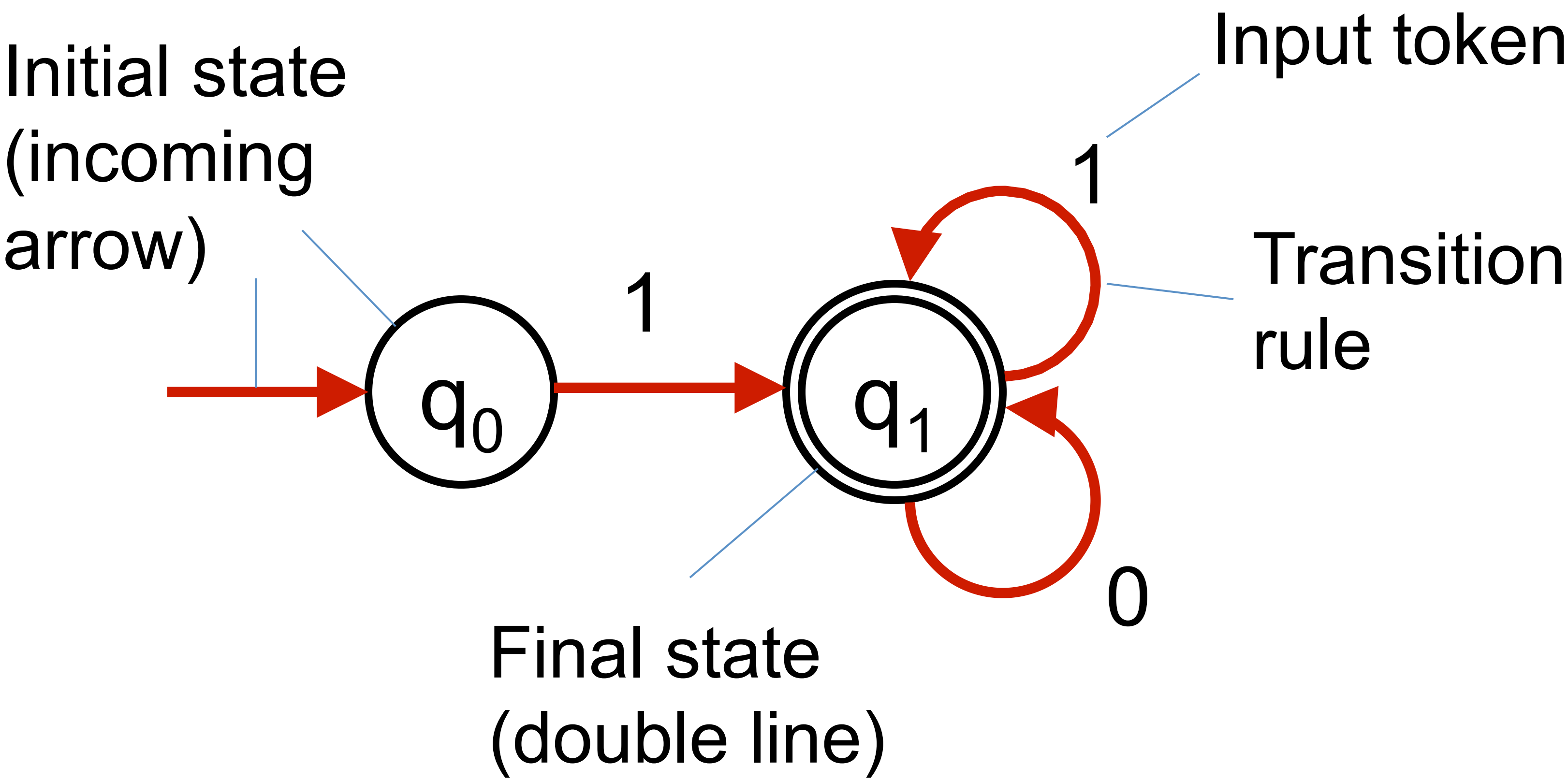
We are interested in the following type of machine (for now):



Definition of a (deterministic) Finite State Automaton (FSA)

- An FSA consists of a finite number of states q_0, q_1,q_2, …, q_n and an input “tape” with input symbols or tokens
- The FSA is in one state at a time, there is one initial state and at least one final state
- Symbols on the input tape are consumed one by one
- For each state there is a finite set of rules for input-dependent state transitions (these depend only on the current state and the current input)

Graphic Representation

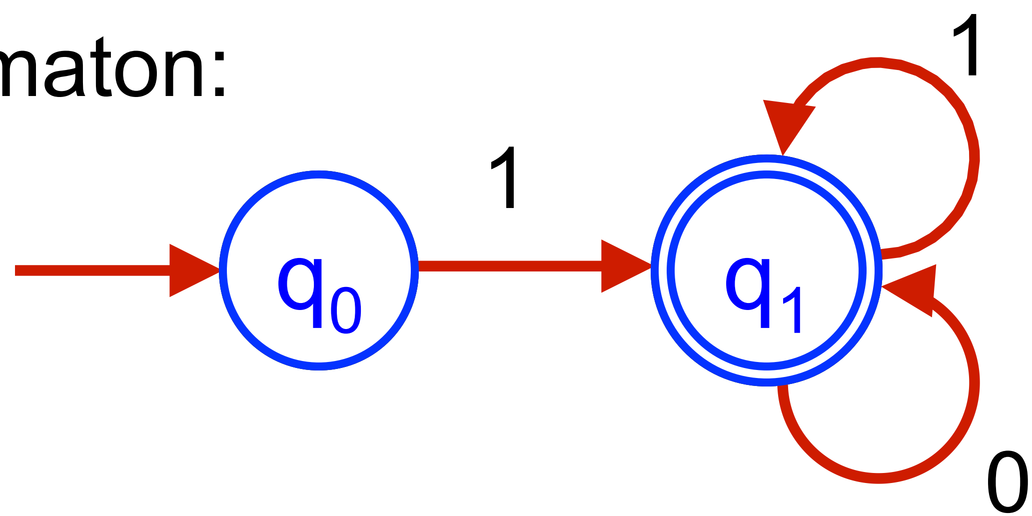


Idea of FSA

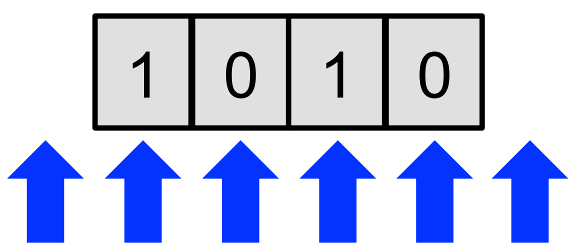
- Description of a decision process
- Is a string acceptable or not?
- All acceptable strings form a language (as we have discussed before)

How it works

Automaton:



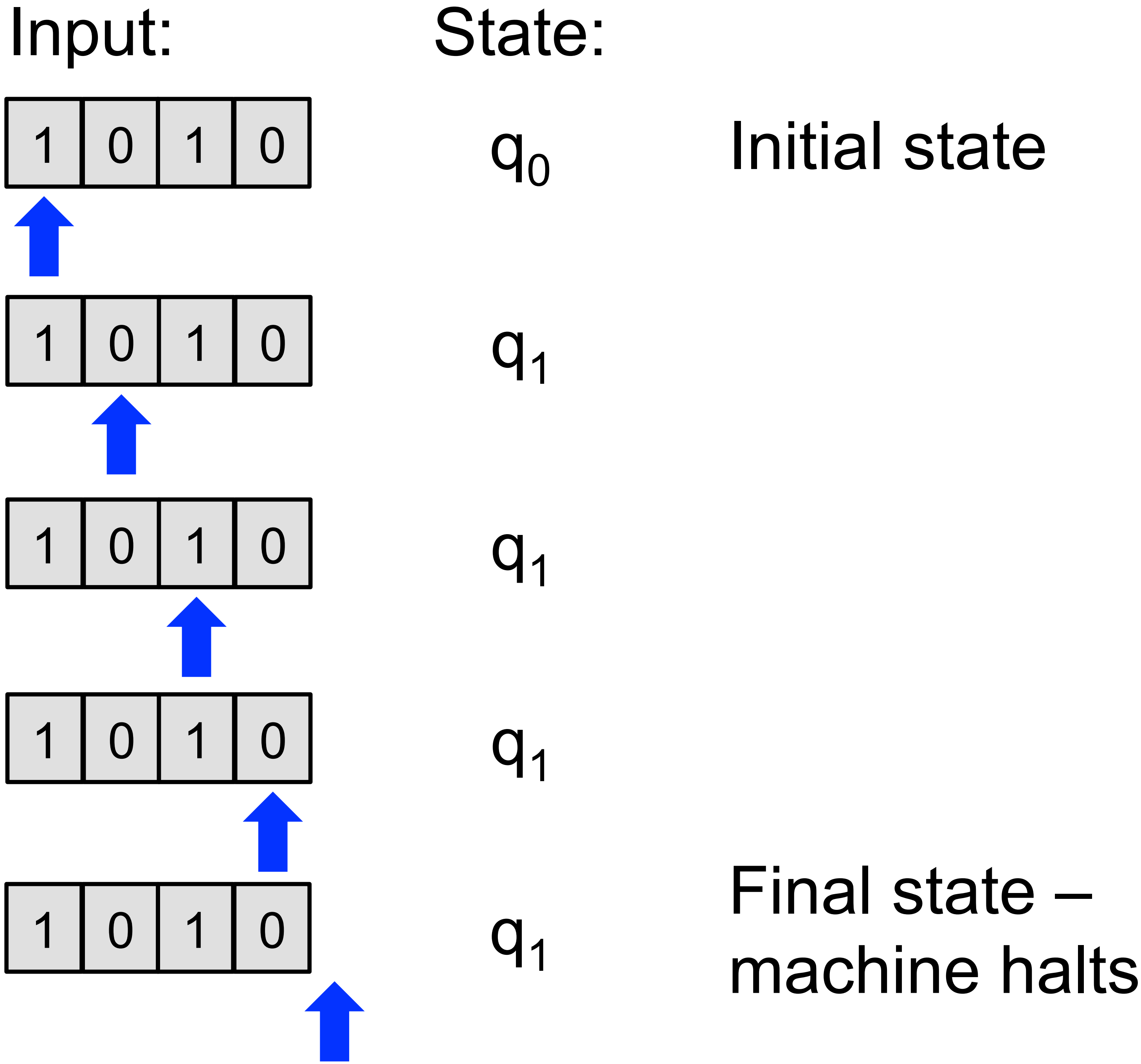
Input tape:



What words does this automaton accept?

Protocol of a computation

We can document the computation I just showed as a list of states and input positions:



Input:
State:
1 0 1 0
q0
Initial state
1 0 1 0
q1
1 0 1 0
q1
1 0 1 0
q1
1 0 1 0
q1
Final state –
machine halts

Outcomes of a FSA computation

- **Accepting computation**: Computation in which the machine reaches a final state and reads all the input.
- **Non-accepting computation**: Computation in which either the machine gets stuck before end of input or finishes in a non-final state.

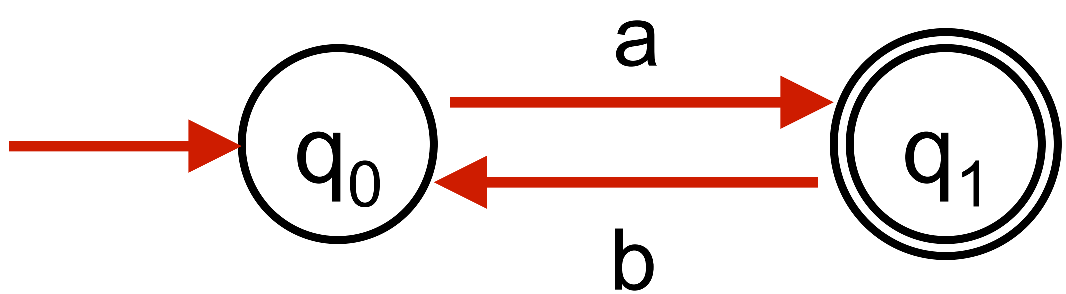
What's accepted -

- An automaton **defines a language**:
  Set of all strings which when given as input give rise to an accepting computation
- The family of languages accepted by **any FSA**:
  Collection of all languages which some finite state machine accepts. – Turns out to be the family of regular languages

Some comments

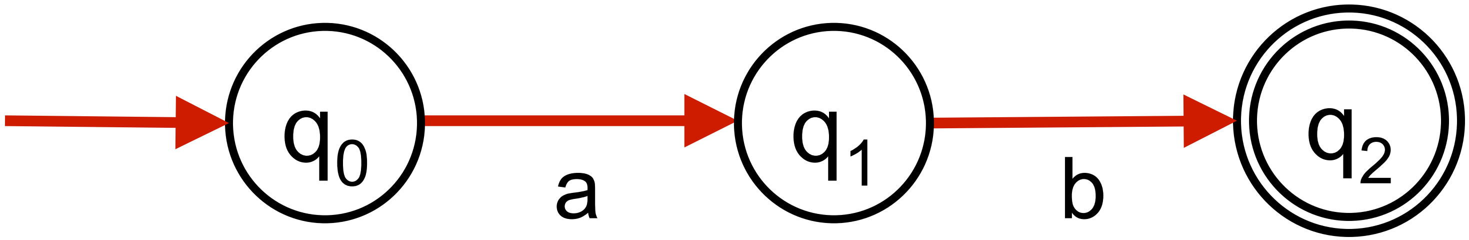
- Getting stuck:
  - no more input available or
  - no transition rule applies
- Input read:
  - Must read past end of the input before accepting a string
- Two choices only:
  - Every input is either rejected or accepted

More examples:



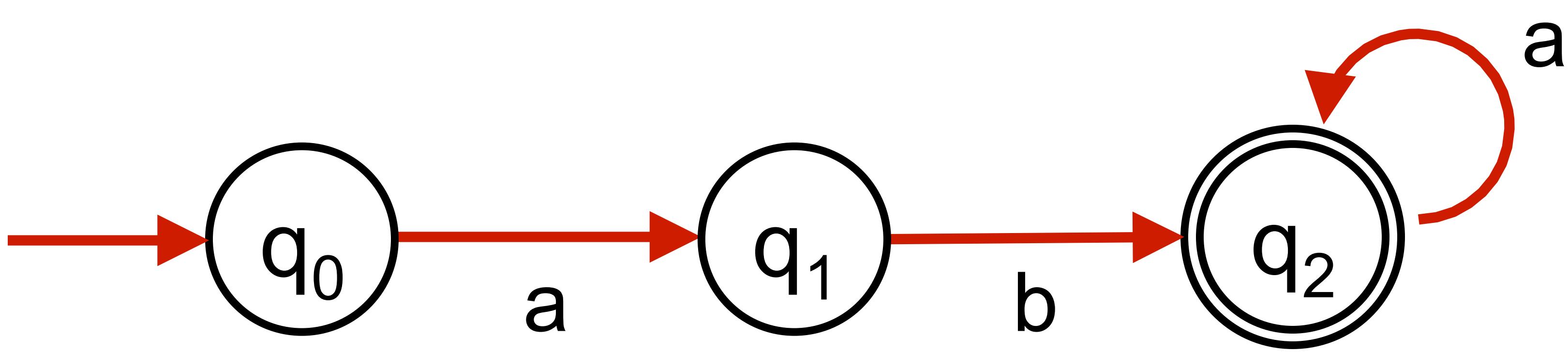
- Accepts any string that has alternating *a*'s and *b*'s that begins and ends with an *a*
- More precisely: $\{a(ba)^n : n \geq 0\}$
- Using **Regular Expression** notation: *a* (*ba*)*

More examples



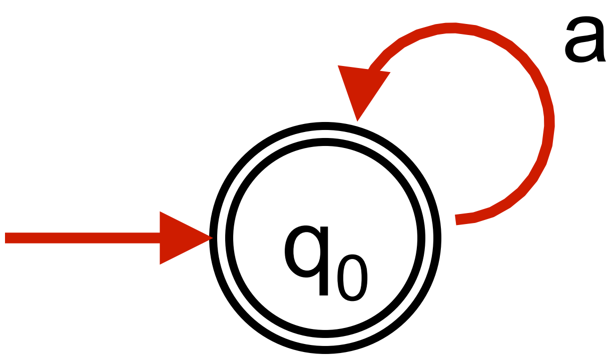
- Accepts only one string: *ab*
- More precisely: {*ab*}
- Regular expression: *ab*
- No cycles gives a finite language

Adding a cycle



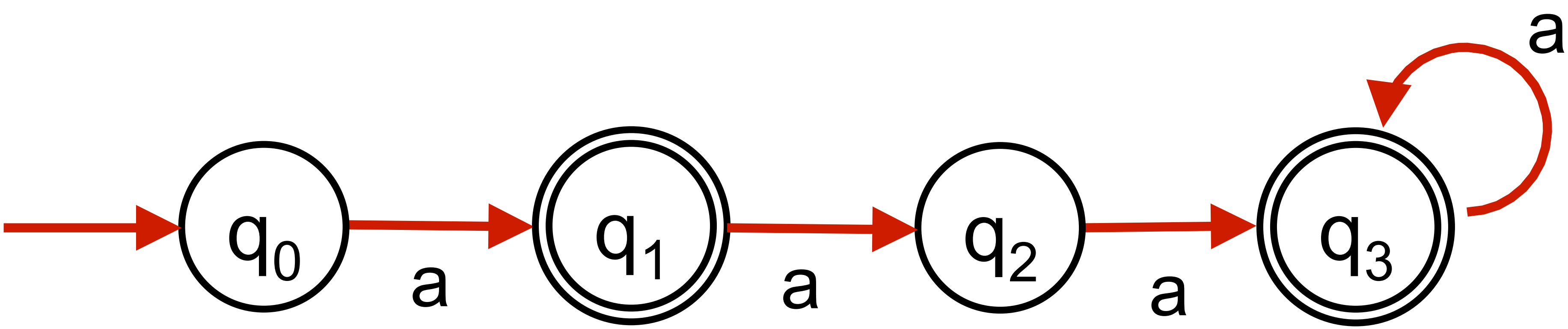
- Accepts ab followed by strings of *a*'s – 0 or more
- More precisely: $\{ab(a)^n : n \geq 0\}$
- Regular expression: *aba**
- Needs states to remember that the first *a and b* were found

Another example



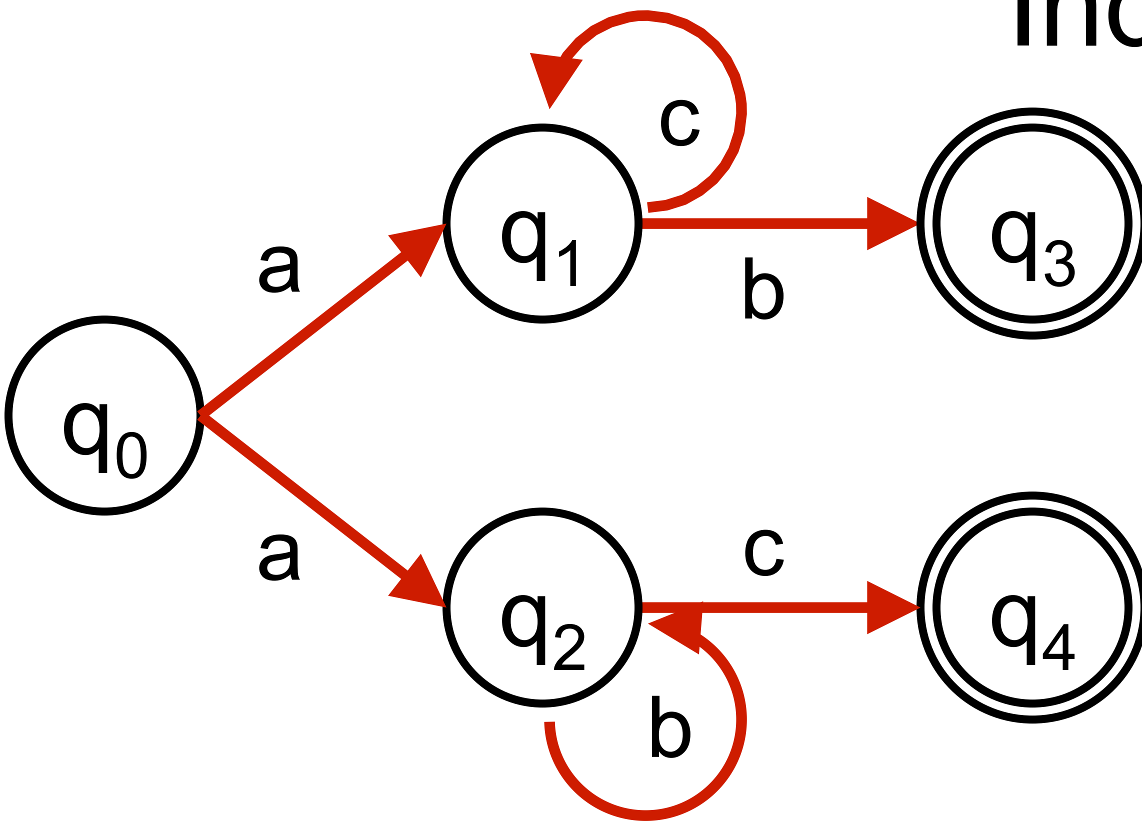
- Accepts any string of *a*'s
- More precisely: $\{a^n : n \geq 0\}$
- Regular expression: *a**
- Initial state can also be a final state

Several final states



- Accepts any string of *a*'s, except *aa*
- More precisely: $\{a^n : n \geq 1 \text{ and } n \neq 2\}$
- Regular Expression: (*a*)|(*aaa*+)
- There can be more than one final state

Indeterministic FSA



- Either *a* then *b*'s then *c*, or *a* then *c*'s then *b*.
- More precisely: $\{ab^nc : n \geq 0\} \cup \{ac^nb : n \geq 0\}$
- Regular expression: (*ab**c*)|(*ac**b*)
- Nondeterministic!

Non-determinism

- What does it mean?
  - Machine has a choice of more than one legal move
  - Machine is able to explore all options
- Significance
  - Important theoretical idea
  - Nondeterminism arises with many computational models

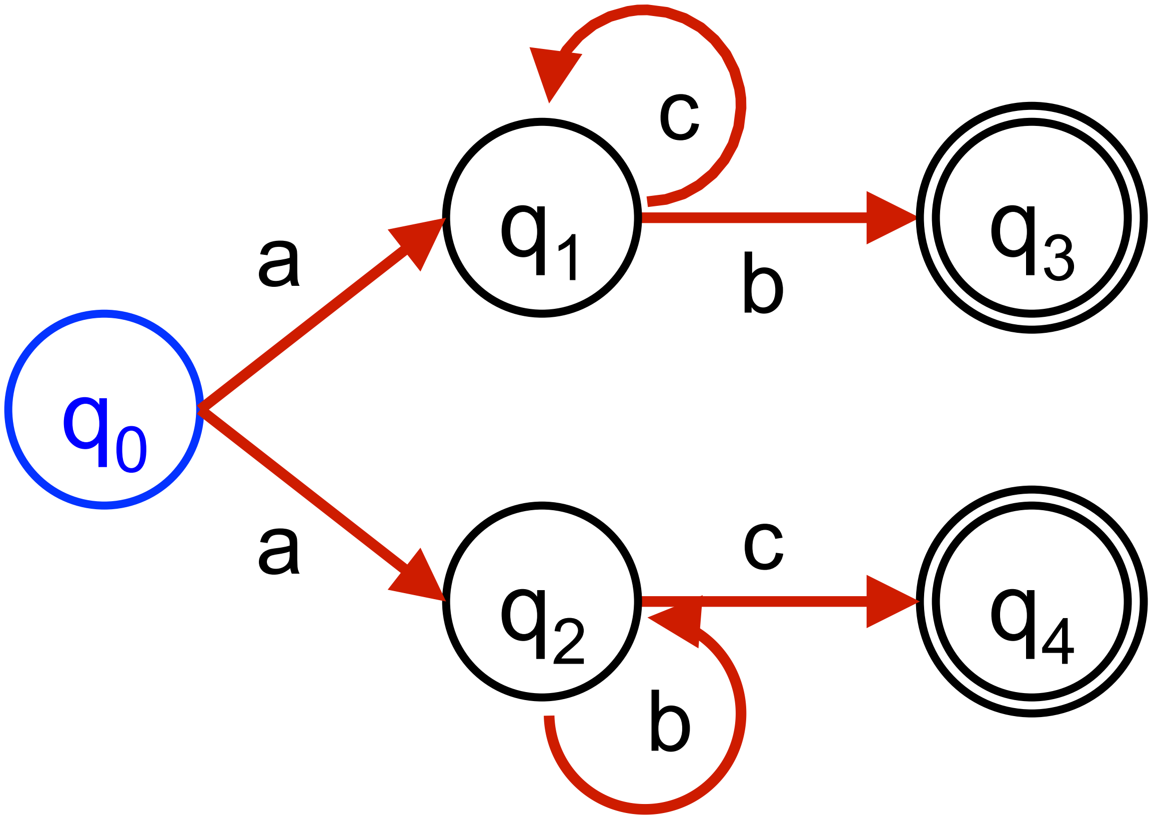
Deterministic versus nondeterministic FSA

- Deterministic FSA: There is never any choice in the computation
- Equivalence (!):
  - Nondeterministic FSA are equivalent to deterministic FSA, i.e. for every FSA there is an equivalent deterministic FSA
  - Prove by means of a construction:

Construction

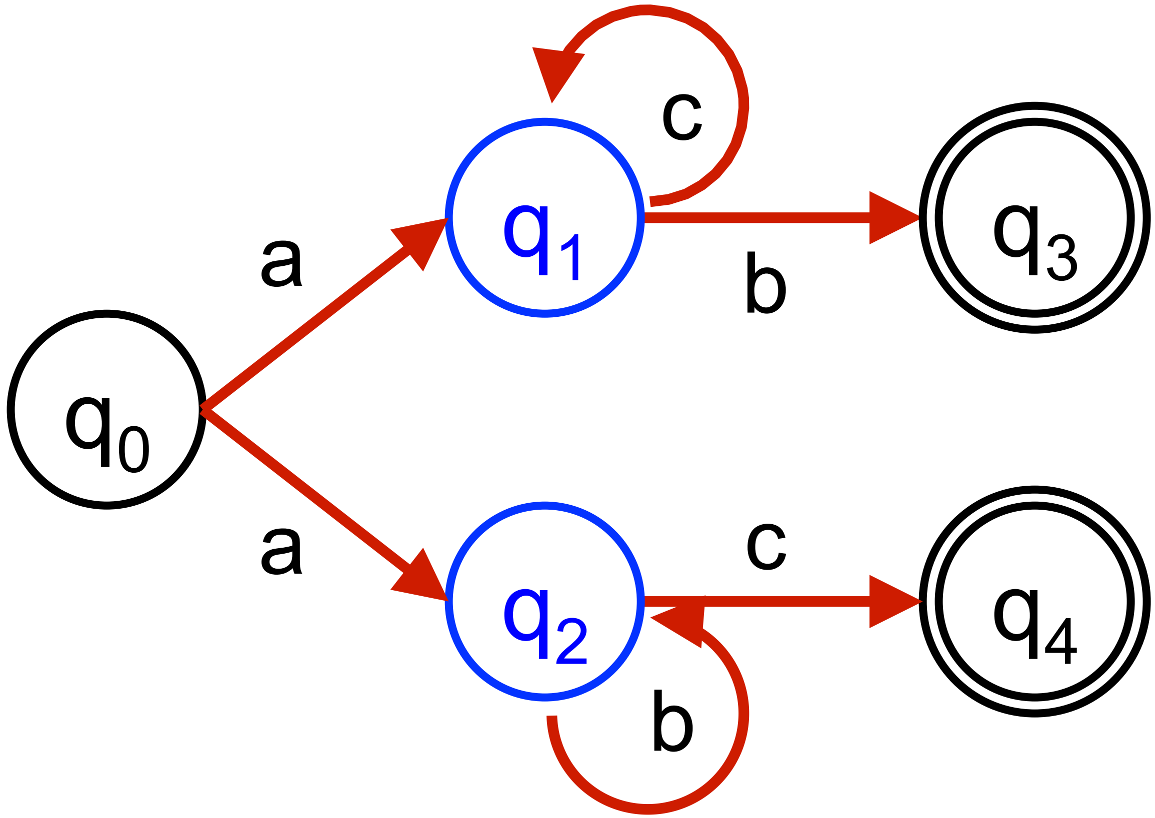
- What do we need to do?
  - Create deterministic machines that simulate nondeterministic machines
- Let's have a closer look at our nondeterministic example...

Example revisited



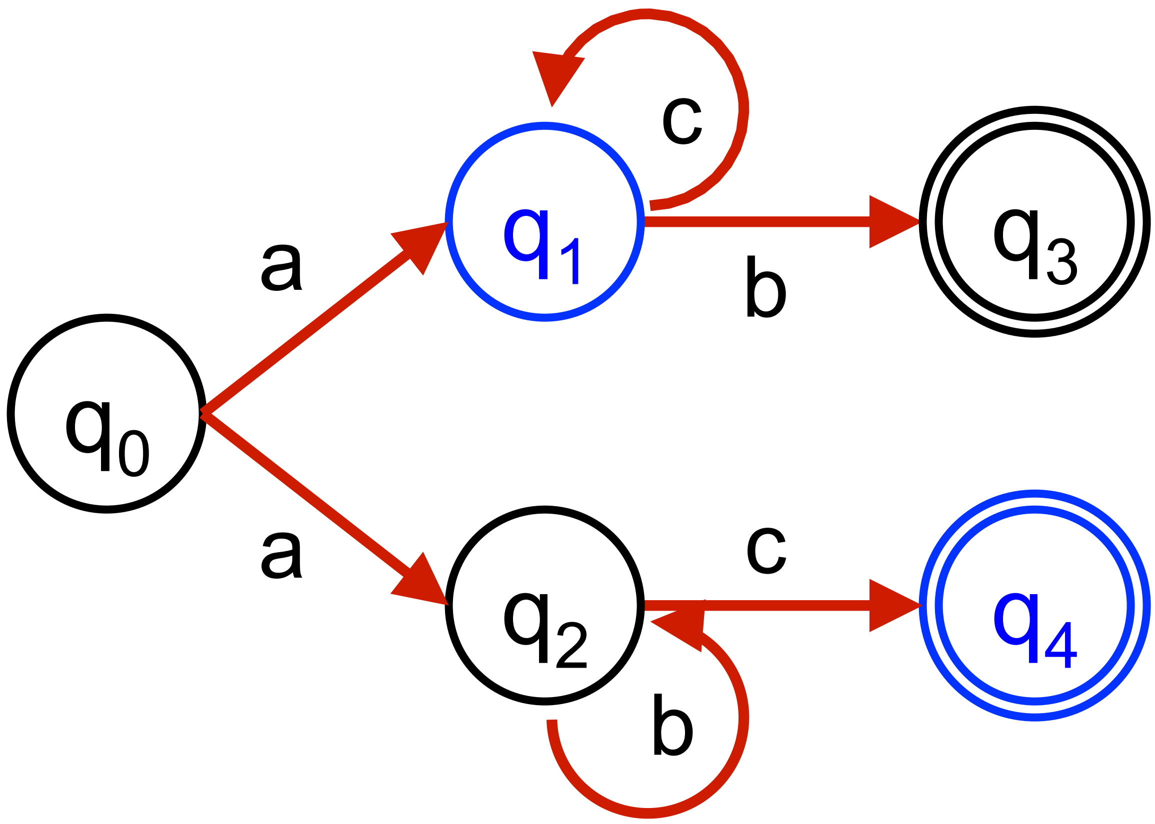
Suppose we see an *a* first

Example revisited



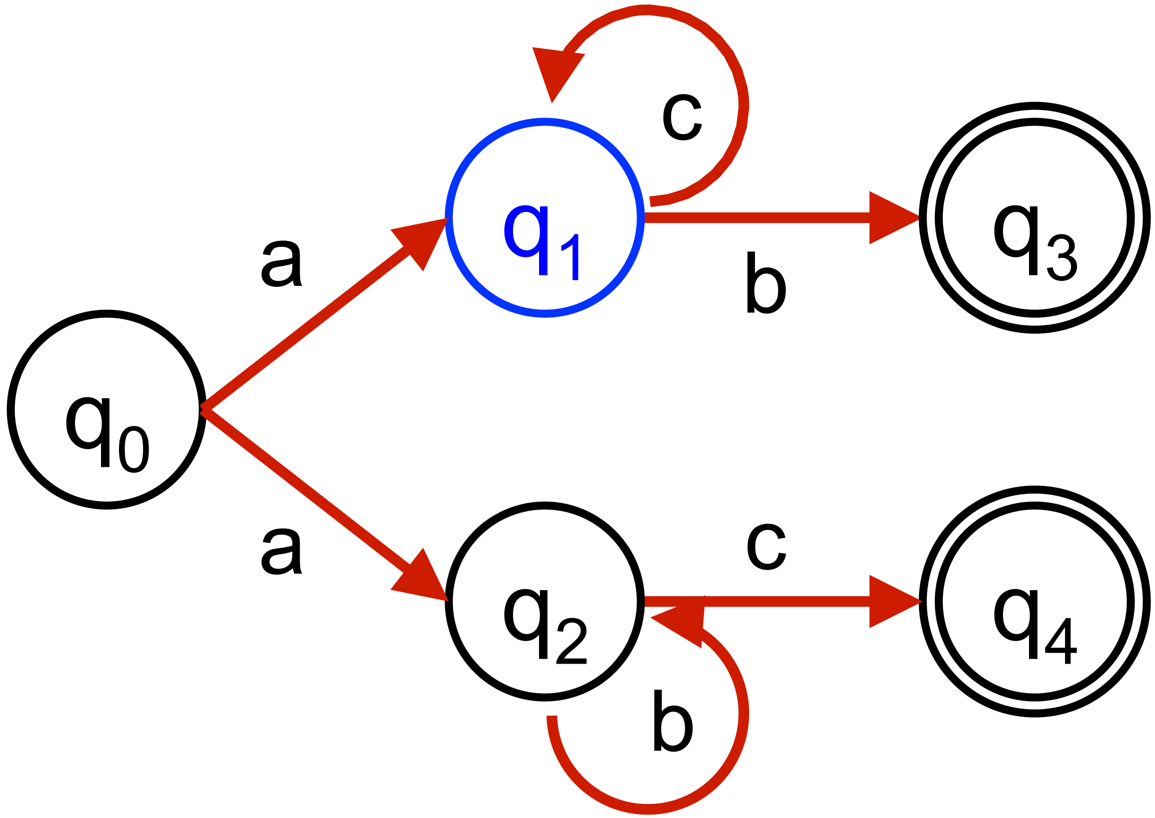
Suppose we see a c next

Example revisited



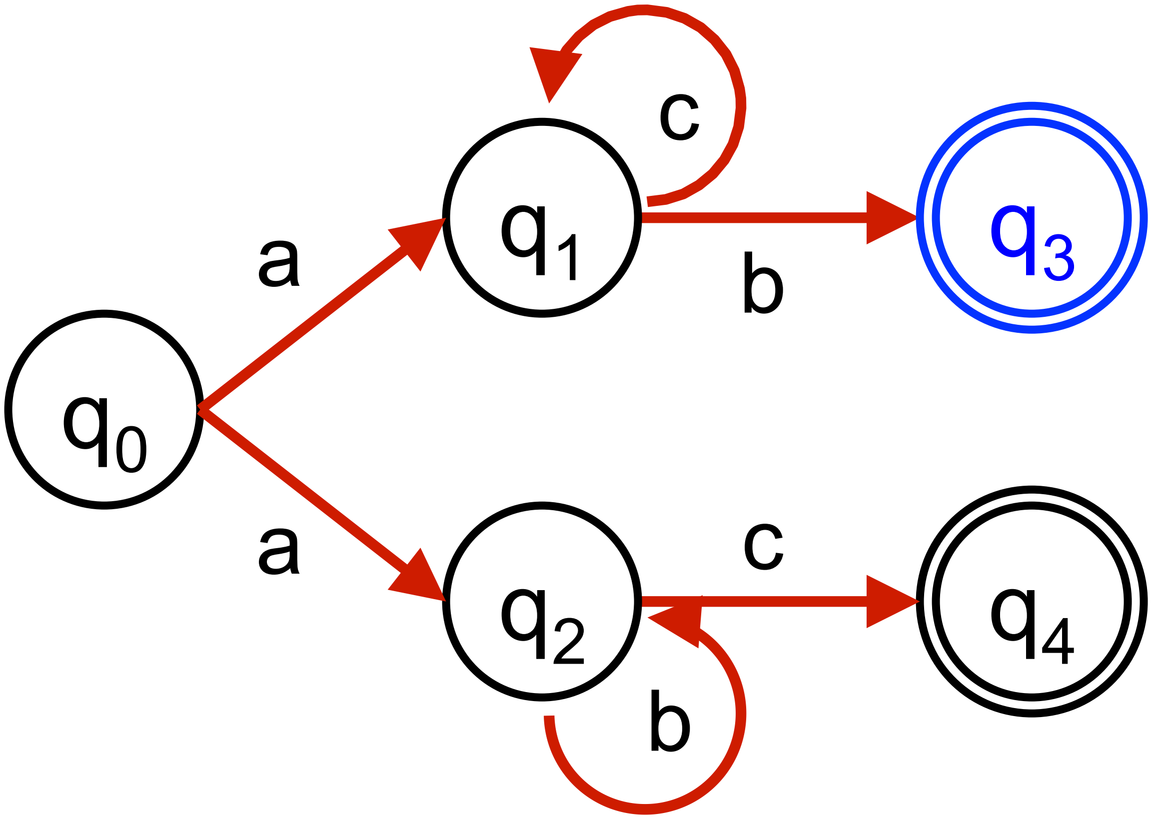
Suppose we see a c next

Example revisited



And finally we see a *b*

Example revisited



The input is consumed and we are in a final state

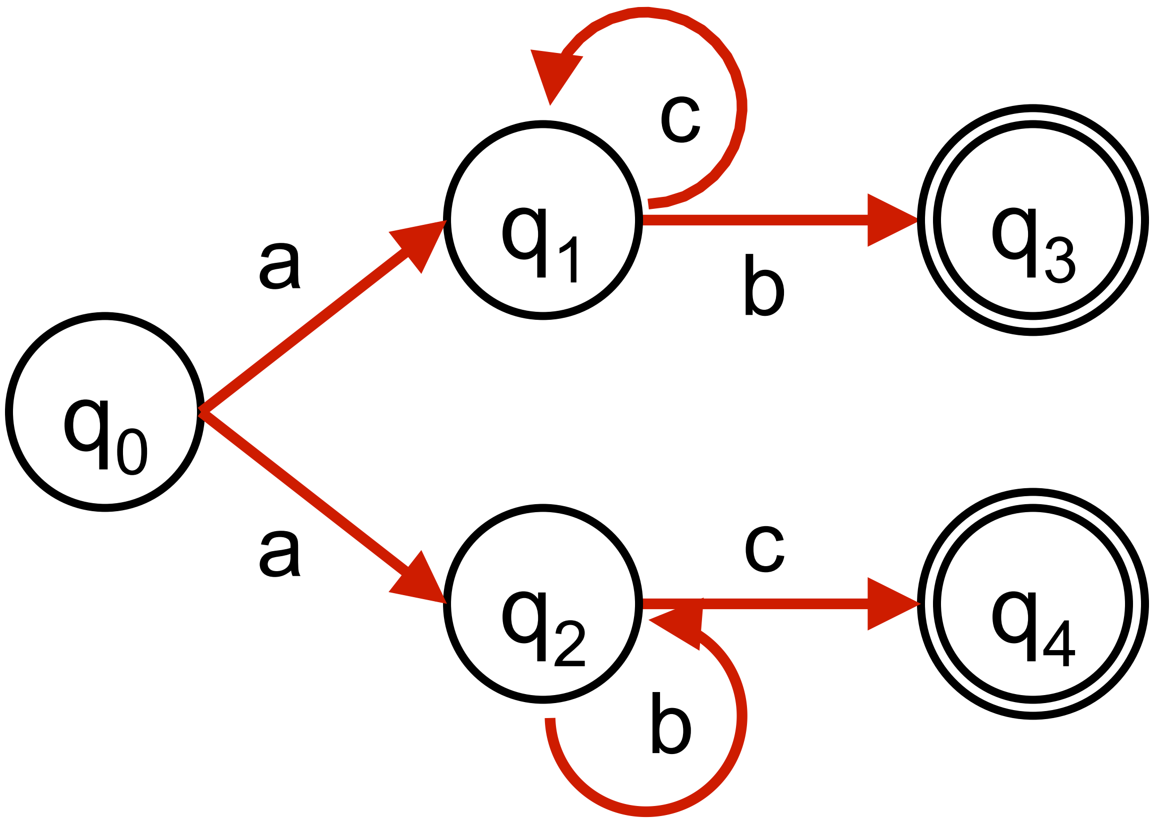
Simulating indeterminism

- Finiteness is crucial:
  - Finite number of states
  - Finite number of possible sets of states
  - 2^n possible subsets of n objects
  - Use subset to record all possible states that could be reached
  - Run all computations of a nondeterministic machine in parallel

Simulating indeterminism

- Build new deterministic machine
  - One state for every subset
  - New transitions based on original machine
  - Next state determined by what original machine would do

Our example



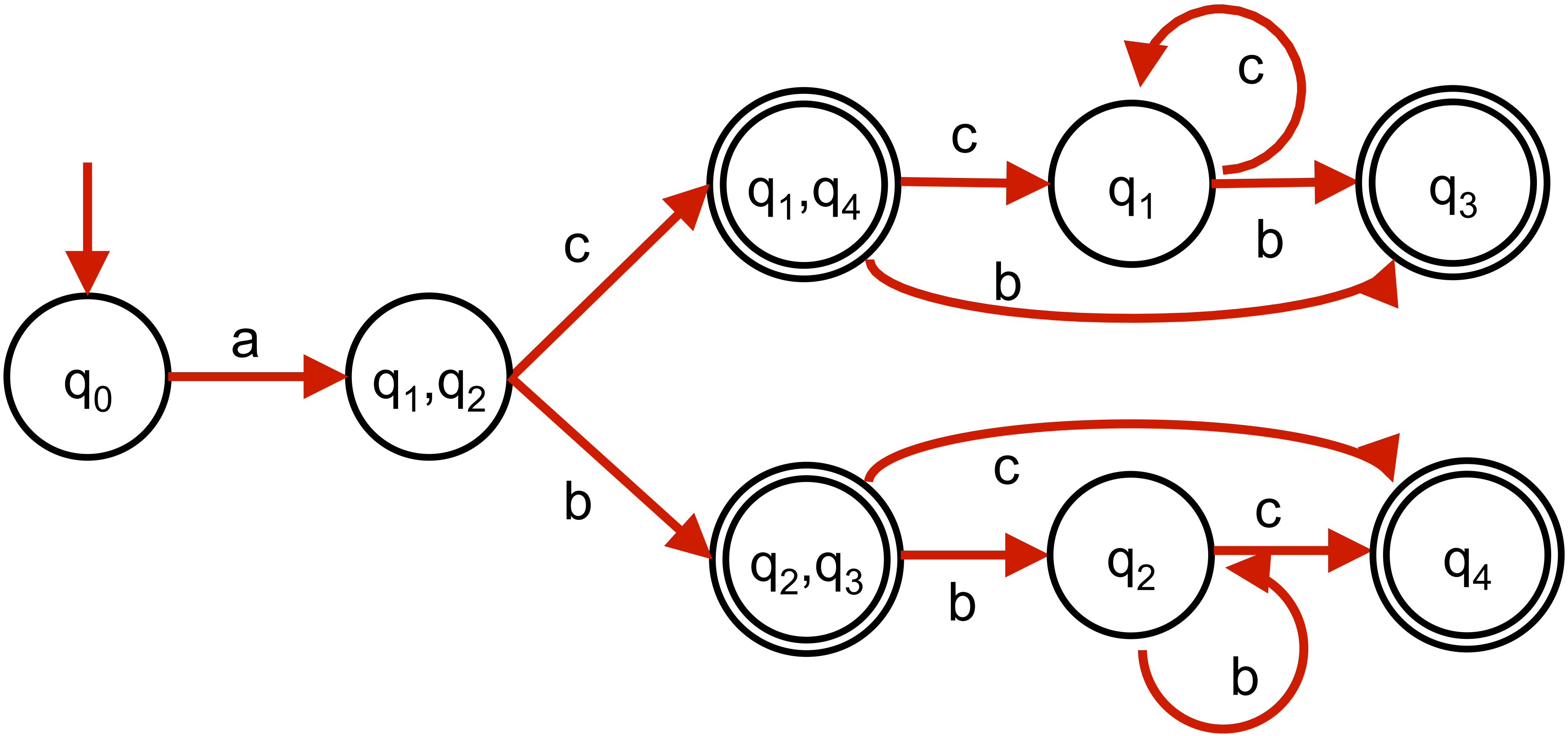
BB Constructing a deterministic machine

- States: $\{q_0\}$, $\{q_1,q_2\}$, $\{q_2,q_3\}$, $\{q_1,q_4\}$, $\{q_3\}$, $\{q_4\}$
- Transition from $\{q_0\}$ to $\{q_1, q_2\}$ on *a*
- Transition from $\{q_1,q_2\}$ to $\{q_1,q_4\}$ on *c*
- Transition from $\{q_1,q_4\}$ to $\{q_3\}$ on *b*
- Transition from $\{q_1,q_2\}$ to $\{q_2,q_3\}$ on *b*
- Transition from $\{q_2,q_3\}$ to $\{q_4\}$ on c

BB Constructing a deterministic machine

- Initial state is { q_0 }
- Any set containing q_3 or q_4 is final

Equivalent deterministic FSA



Stepping back …

- What did we just do?
  - We showed something very general
  - Two classes of machines are equivalent
  - Based on a general simulation
  - This is an important idea