Summary of enumerability

Two ways of looking at enumerability

- A set A is enumerable if it is the range of a function $f : N \to A$ from the natural numbers to A.
- Alternative description: a set A is enumerable if it has an encoding, i.e. if there is a total injective function $c : A \to N$ that sends every element x of A to its **code** $c(x)$.

Enumerability and non-enumerability

- Some important sets are enumerable: the natural numbers (trivially), the integers, pairs of integers, the rational numbers, strings, **computer programs**, etc.
- Some sets are not enumerable, as shown by Cantor's famous diagonal argument: the powerset $P(N)$ of the natural numbers, and—**most importantly**—the functions $N \rightarrow N$.

Significance of enumerability

- Enumerability is important, because enumerable sets can be represented on a computer, and non-enumerable sets cannot.
- Because computer programs are enumerable and functions $N \rightarrow N$ are not, some functions cannot be computable (i.e. represented by a computer program).

Automata

Automata in computer science

- In computer science, and **automaton** is an abstract computing machine.
- “Abstract” means here that it need not exist in physical form, but only as a precisely-described idea.

Automata in this lecture

- **Turing machines** (1937) and **abacus machines** (1960s): have all capabilities of today's computers. Used to study the boundary between **computable** and **uncomputable**.
- **Finite automata** (also called **finite state machines**, emerged during the 1940's and 1950's): originally introduced to model brain functions, but they turned out to have important applications in computer science.

Finite automata

We shall study finite automata first, because they can be seen as a first step towards Turing machines and abacus machines.

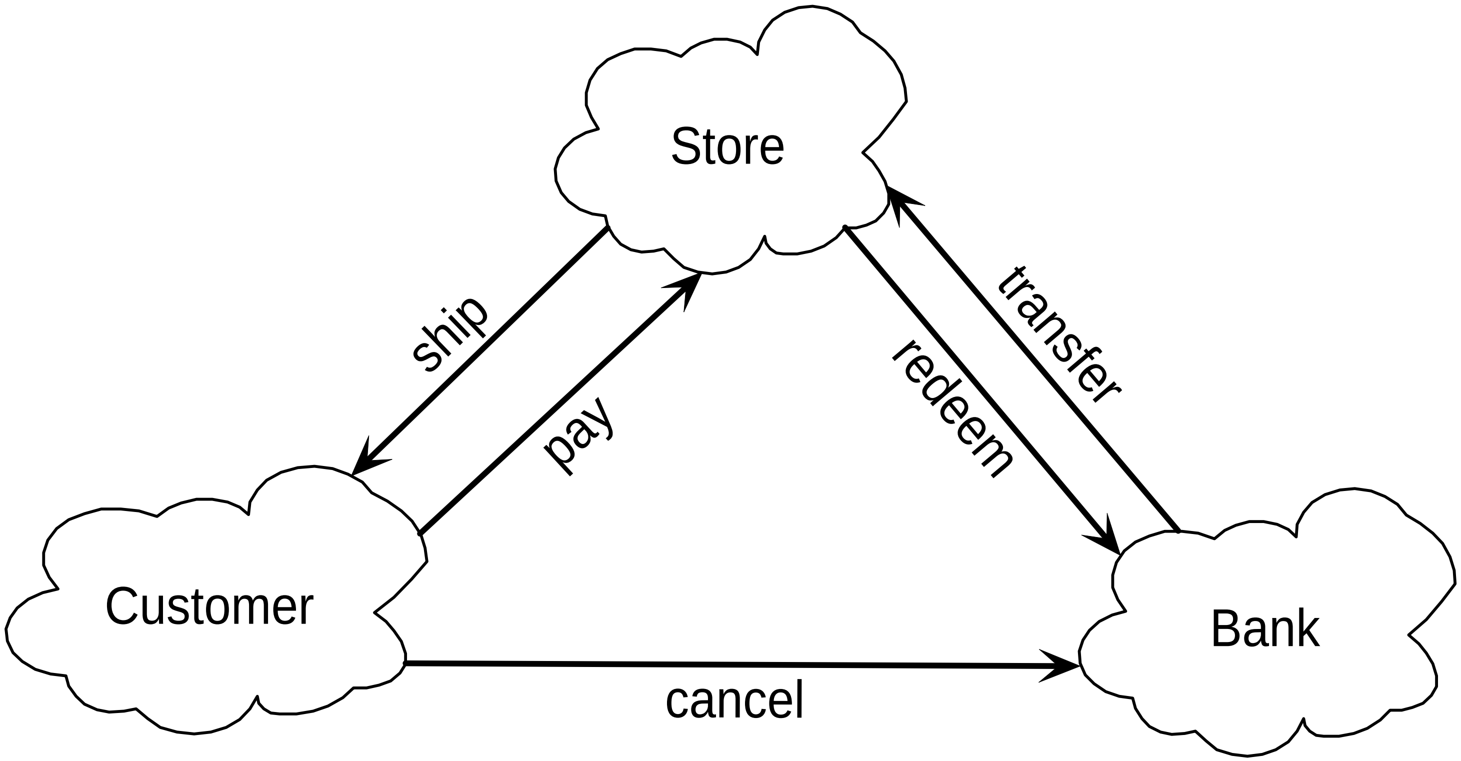
Uses of finite automata

- Used in software for verifying all kinds of systems with a finite number of states, such as communication protocols
- Used in software for scanning text, to find certain patterns
- Used in “Lexical analyzers” of compilers (to turn program text into “tokens”, e.g. identifiers, keywords, brackets, punctuation)
- Part of Turing machines and abacus machines

Motivating example for finite automata

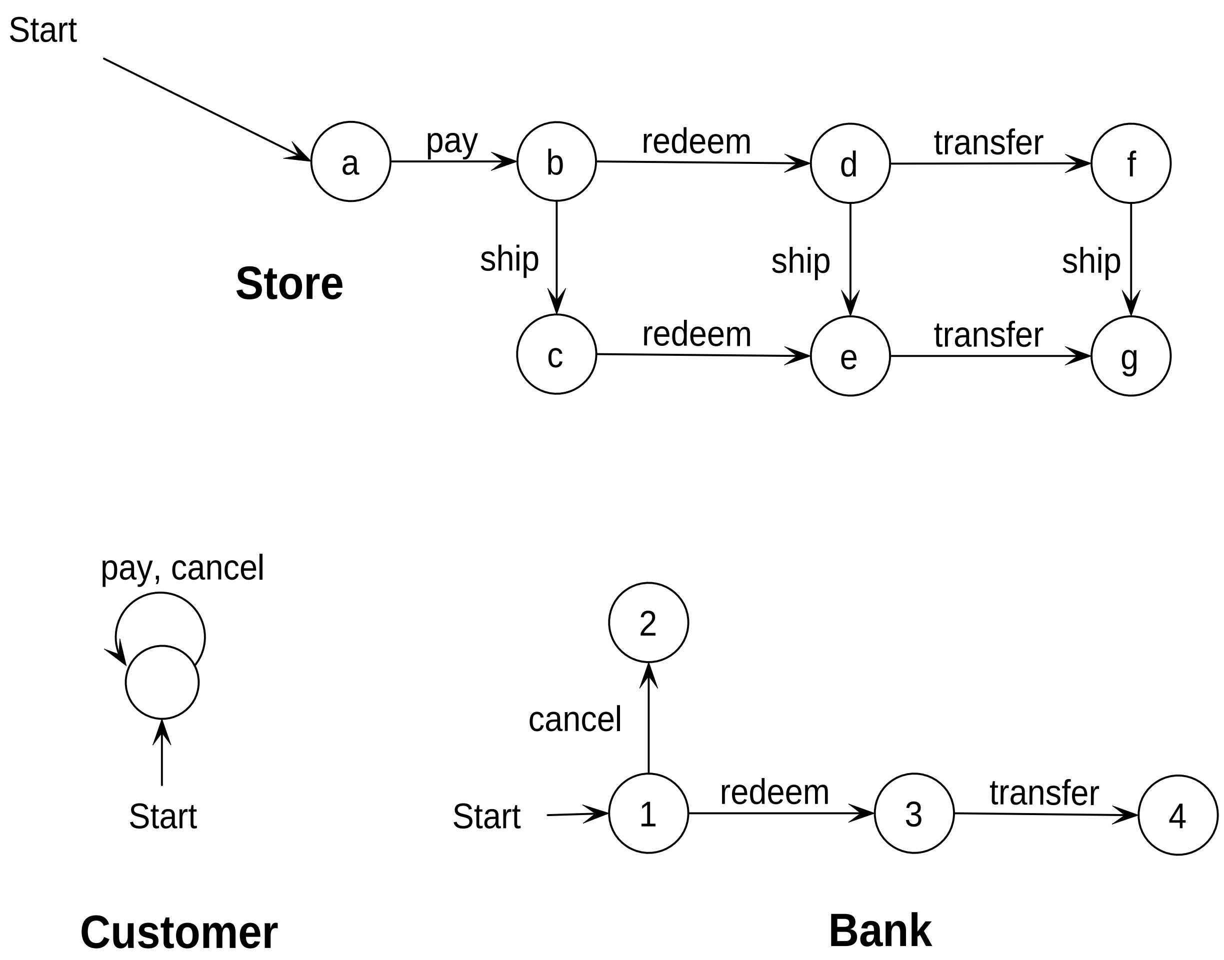
- Next, we shall see how finite automata can be used to model real-life systems, e.g. the interactions between a Customer, a Store, and a Bank. (This example is from the book by Hopcroft/Motwani/Ullman.)
- The automata describe the rules of interaction, also called the **communication protocol**.
- They allow to answer questions about the system that are hard or impossible to obtain otherwise.

Communication protocol



Customer, Store, and Bank will be finite automata.

A close look at the participants



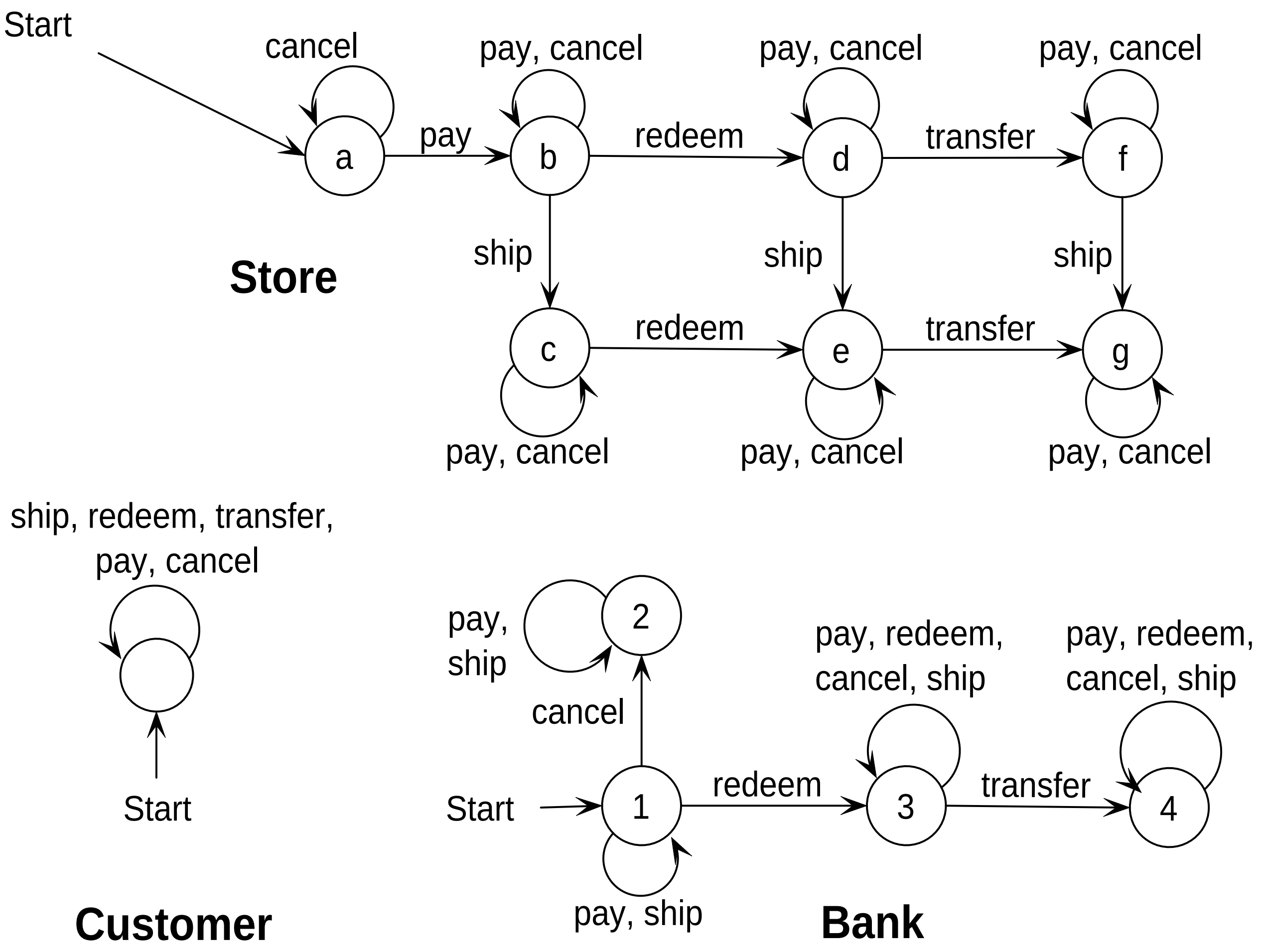
Simulating the whole system

- Idea: running Customer, Store, and Bank “in parallel”.
- Initially, each automaton is in its start position.
- The system can move on for every action that is possible in **each** of the three automata.

The missing irrelevant actions

- Problem: Bank gets stuck during the pay action, although paying is only between Customer and Store.
- Solution: we need to add a loop labeled “pay” to state 1 of Bank.
- More generally, we need loops for all such “irrelevant” actions.
- But illegal actions should remain impossible. E.g. Bank should not allow “redeem” after “cancel”.

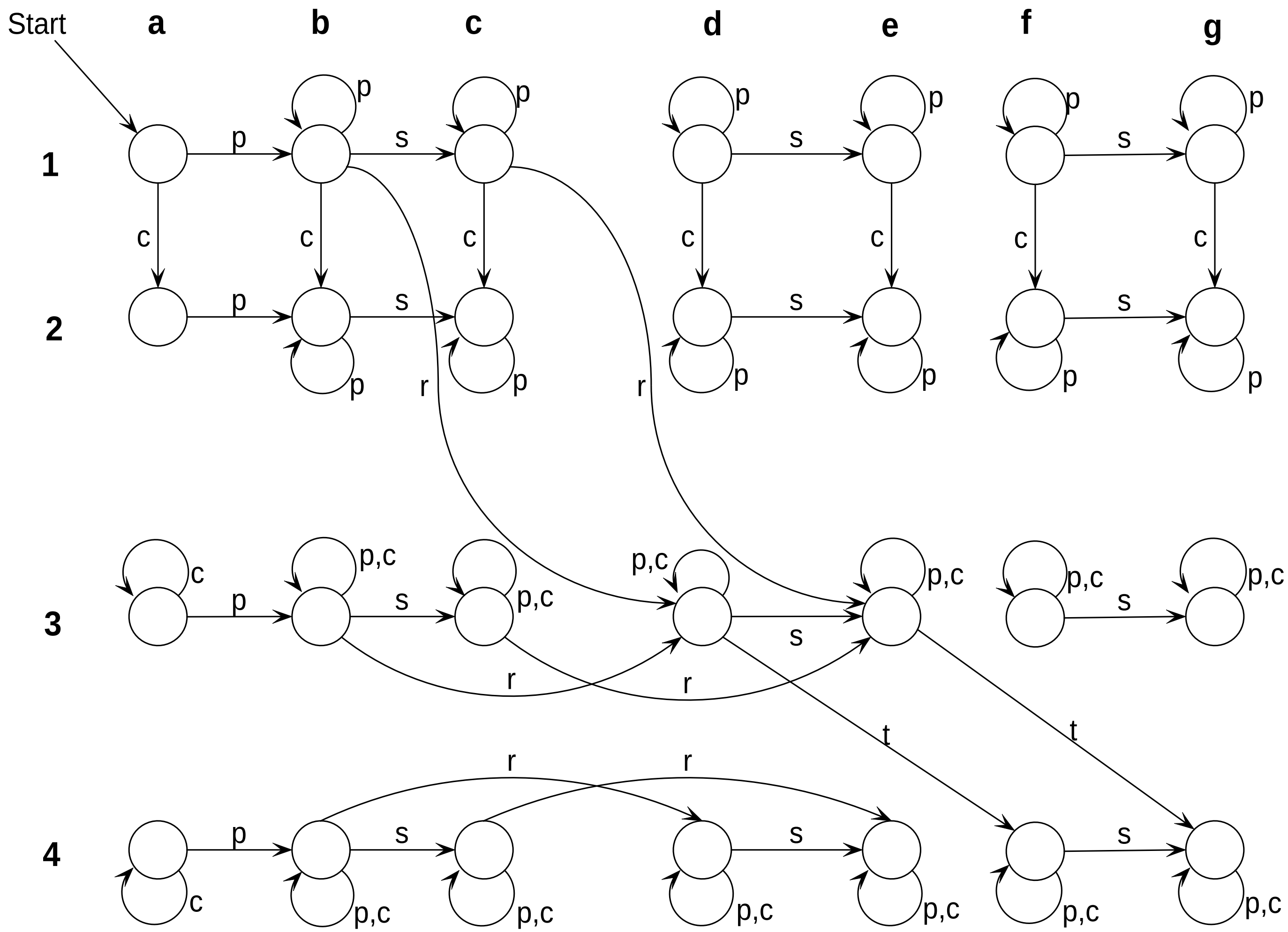
Adding irrelevant actions



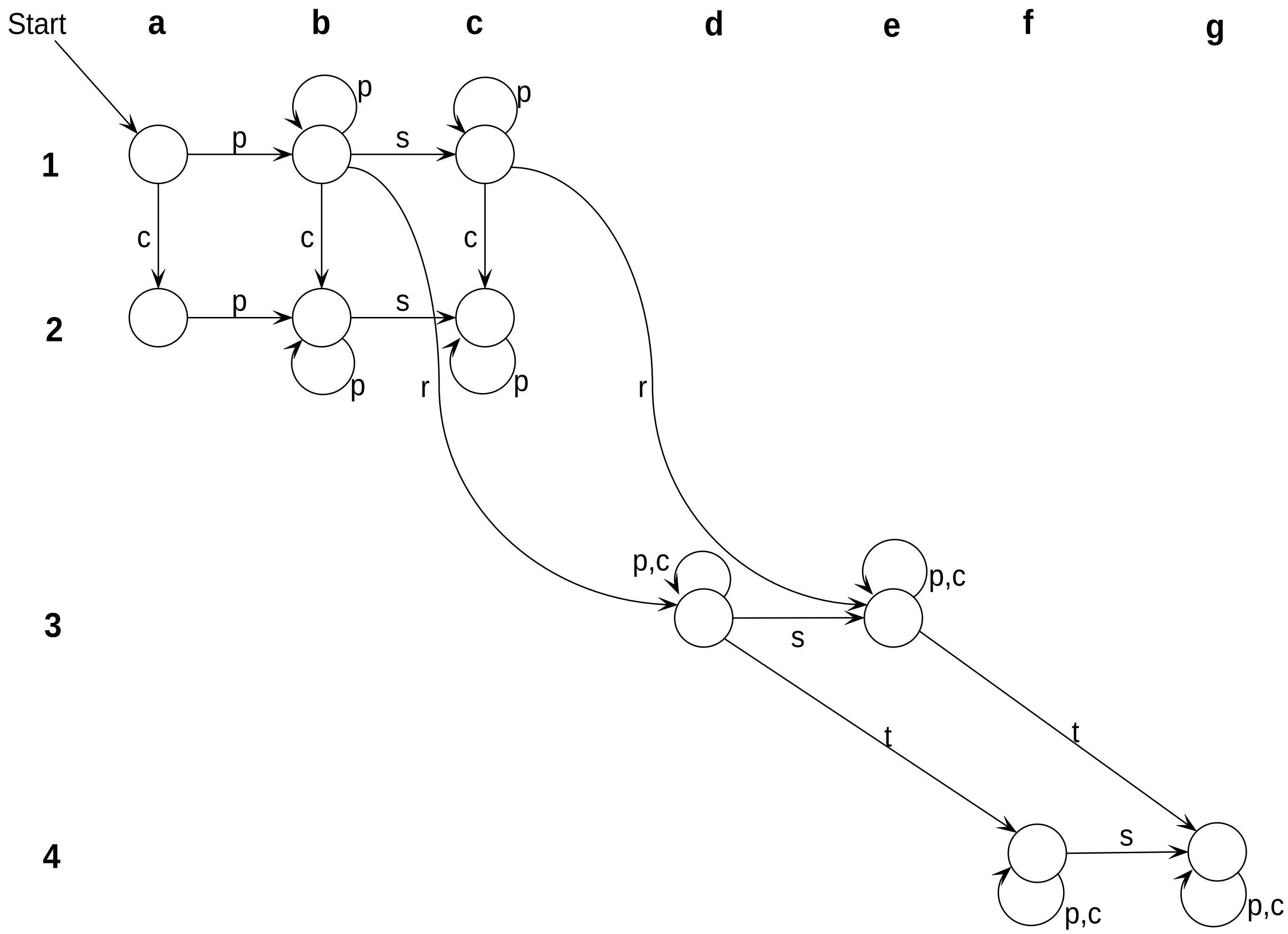
Simulating the whole system

- Simulation by **product automaton**.
- Its states are pairs $(StoreState, BankState)$, e.g. (a,1) or (c,3). (Because Customer has only one state and allows every action, it can be neglected.)
- It has a transition $(StoreState, BankState) \xrightarrow{action} (StoreState', BankState')$ whenever Store has a transition $StoreState \xrightarrow{action} StoreState'$ and Bank has a transition $BankState \xrightarrow{action} BankState'$.

Product automaton



Without unreachable states



Usefulness for protocol verification

- We can now answer all kinds of interesting questions, e.g. “Can it happen that Store ships the product and never receives the money transfer?”
- Yes! If Customer has indicated to pay, but sent a cancellation message to the Bank, we are in state (b,2). If Store ships then, we make a transition into (c,2), and the Store will never receive a money transfer!
- So store should never ship before redeeming.

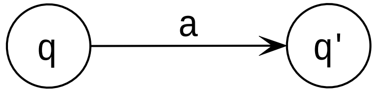
Formal definition of DFA’s

Definition. A **deterministic finite automaton** (**DFA**) consists of

- a finite set of **states**, often denoted Q,
- a finite set Σ of **input symbols**,
- a total **transition function** $\delta : Q \times \Sigma \to Q$,
- a **start state** $q_0 \in Q$, and
- a set $F \subseteq Q$ of **final** or **accepting states**.

Remark: we require the transition function to be total, but some people allow it to be partial

Terminology and intuitions

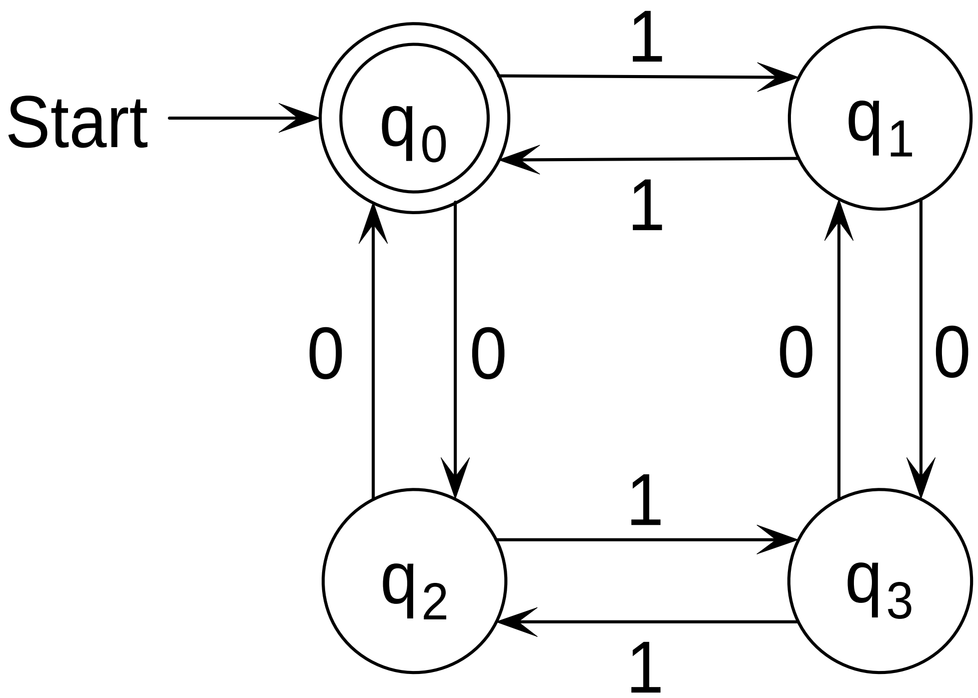
- The transition graph we used before is an informal presentation of the transition function δ. We have 

if $\delta(q, a) = q'$.
- "Deterministic" means that for every state q and input symbol a, there is a **unique** (i.e. exactly one) following state, $\delta(q, a)$.
- Later, we shall also see **non-deterministic finite automata** (NFA's), where (q, a) can have any number of following states.
- FA's are also called "finite state machines".

Useful notations for DFA's

- Transition graph, like that for Customer, Store, or Bank.
- Transition table, which is a tabular listing of the δ function.

Transition graph: example



$Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{0, 1\}$, $\delta(q_3, 0) = q_1$ . . . ,
$F = \{q_0\}$.

Meaning of the transition graph

- The nodes of the graph are the states.
- The labels of the arrows are input symbols.
- The labeled arrows describe the transition function.
- The node labeled “Start” is the start state q_0.
- The states with double circles are the final states.

Transition table: example

	0	1
$\rightarrow q_0$	q_2	q_0
$*q_1$	q_1	q_1
$*q_2$	q_2	q_1

$Q = \{q_0, q_1, q_2\}$, $\Sigma = \{0, 1\}$, $\delta(q_0, 0) = q_2$, $\delta(q_0, 1) = q_0 \ldots$, $F = \{q_1, q_2\}$.

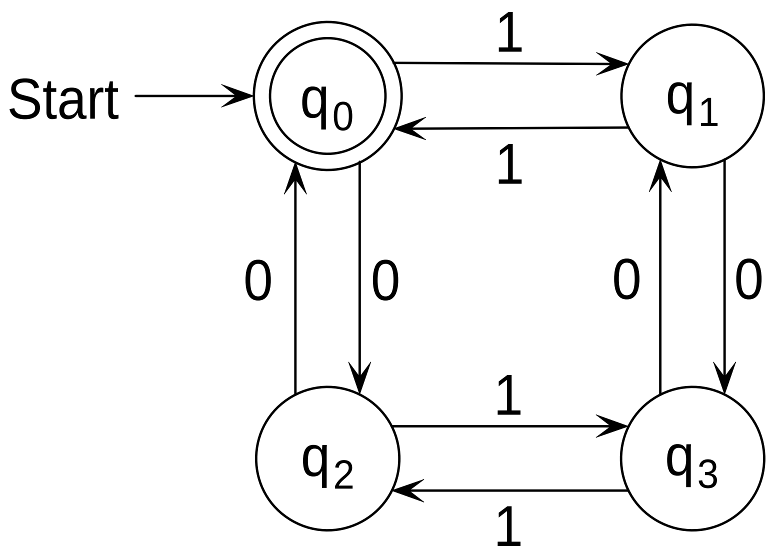
Meaning of the transition table

- The symbols in the leftmost column are the states.
- The symbols in the top row are the input symbols.
- The symbols “inside” the table describe the transition function.
- The arrow in the leftmost column marks the start symbol.
- The symbol $*$ in the leftmost column marks the final states.

How a DFA processes strings

- Let $a_1 a_2 \cdots a_n$ be a string of input symbols.
- Initially, the DFA is in its start state q_0.
- Let q be the state reached after the first i symbols $a_1 a_2 \cdots a_i$ of the input string. Upon reading the next symbol a_{i+i}, the DFA makes a transition into the new state $\delta(q, a_{i+1})$.
- Repeated until the last symbol a_n.
- The DFA said to **accept** the input string if the state reached after the last symbol a_n is in the set F of final states.

Accepted strings: example



- This DFA accepts 1010, but not 1110
- It accepts those strings that have an even number of 0's and an even number of 1's.
- Therefore, we call this DFA "parity checker".

Formal approach to accepted strings

We define the **extended transition function** $\hat{\delta}$. It takes a state q and an input **string** w to the resulting state. The definition proceeds by **induction** over the length of the input string.

- Induction basis (length 0): $\hat{\delta}(q, \epsilon) = q$. (The greek letter ϵ stands for the **empty string**, i.e. the word consisting of zero symbols.)
- Induction step (from length n to length $n+1$): $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$ (where w is an input string of length n, and a an input symbol).

The language of a DFA

- Intuitively, the language of a DFA A is the set of strings w that take the start state to one of the accepting states.
- Formally, the language $L(A)$ accepted by the DFA A is defined as follows:

$$L(A) = \{w \mid \hat{\delta}(q_0, w) \in F\}.$$

Exercises

Give DFA's accepting the following languages over the alphabet $\{0, 1\}$. (Note that you can choose between giving a transition table, a transition graph, or a formal presentation of Q, Σ, q_0, δ, and F.)

1. The set of all strings ending in 00.
2. The set of all strings with two consecutive 0's (not necessarily at the end).
3. The set of strings with 011 as a substring.

Exercise

For the alphabet $\{a, b, c\}$, give a DFA accepting all strings that have abc as a substring.

Exercises

(More advanced; do not worry if you need tutor's help to solve this.) Give DFA's accepting the following languages over the alphabet $\{0, 1\}$.

1. The set of all strings such that each block of five consecutive symbols contains at least two 0's.
2. The set of all strings whose tenth symbol from the right is a 1.
3. The set of strings such that the number of 0 is divisible by five, and the number of 1's is divisible by three.

Exercise

Consider the DFA with the following transition table:

	0	1
$\rightarrow A$	A	B
$*B$	B	A

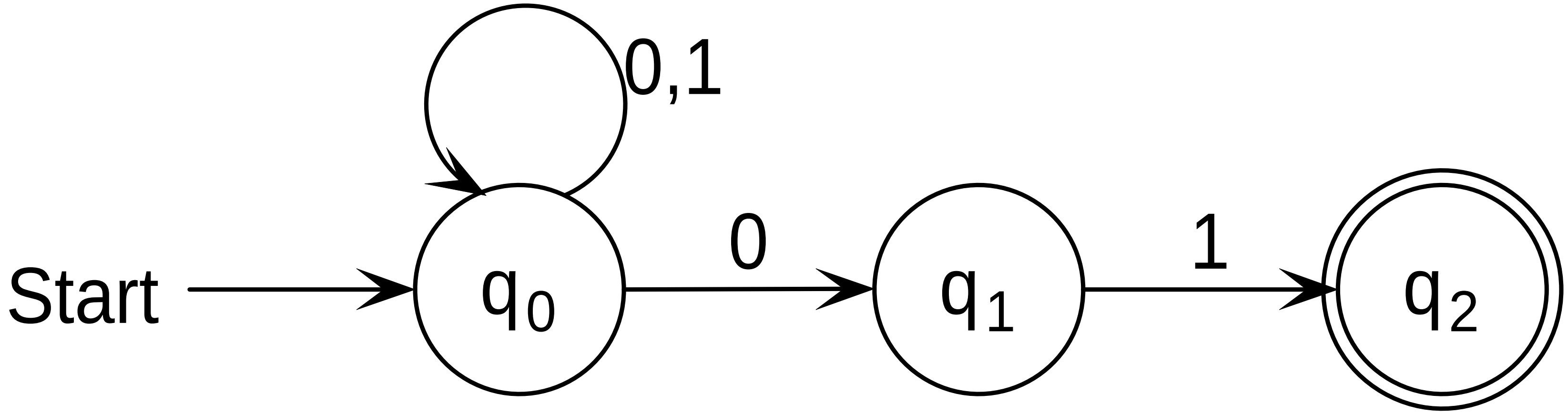
(1) Informally describe the language accepted by this DFA; (2)prove by induction on the length of an input string that your description is correct. (Don't worry if you need tutor's help for (2).)

Non-deterministic FA (NFA)

- An NFA is like a DFA, except that it can be in several states at once.
- This can be seen as the ability to guess something about the input.
- Useful for searching texts.

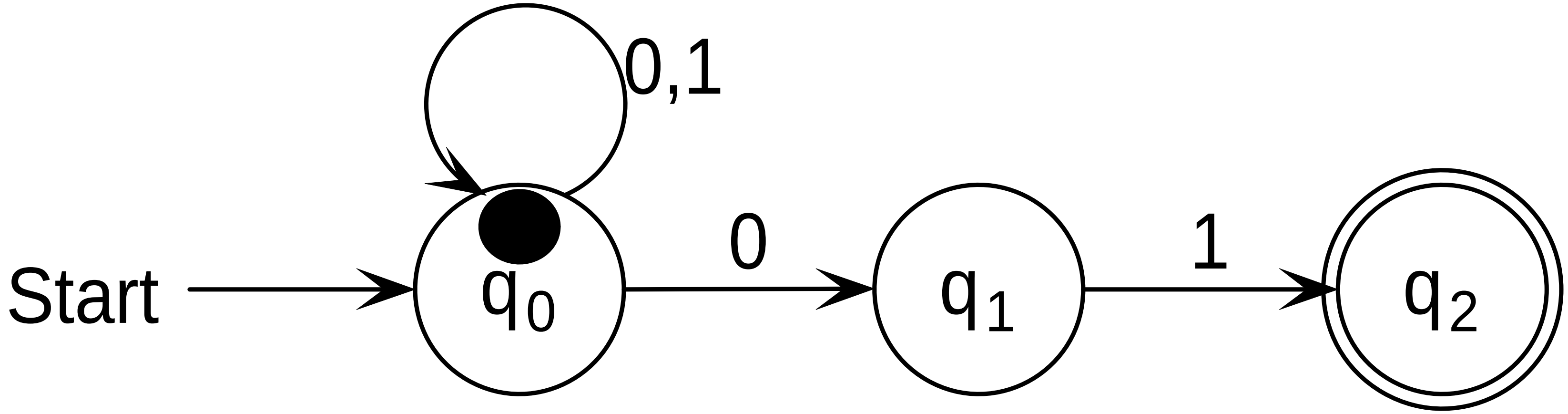
NFA: example

An NFA accepting all strings that end in 01:



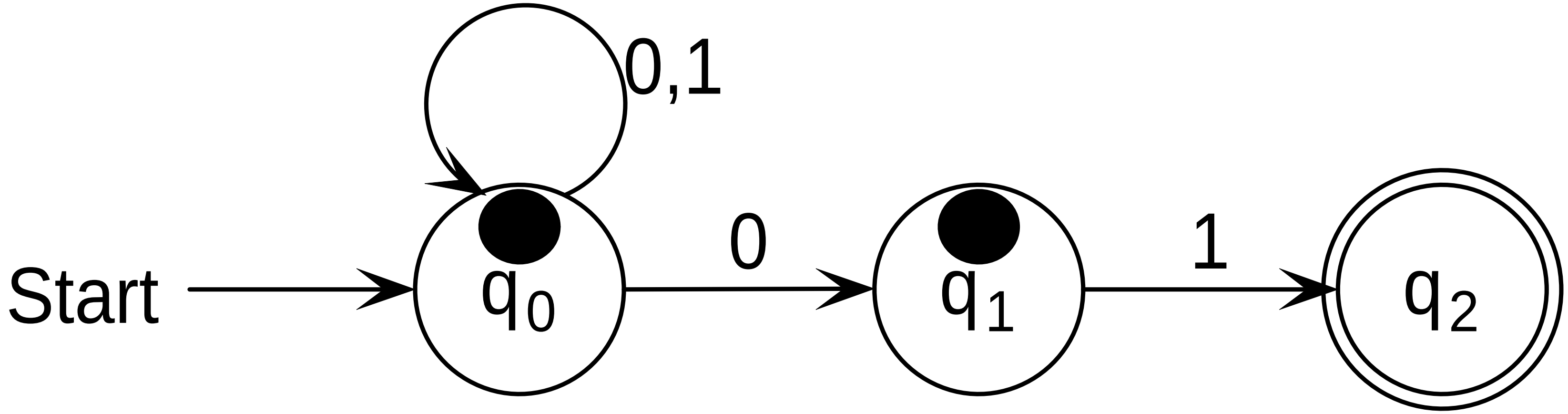
It is non-deterministic because input 0 in state q_0 can lead to both q_0 and q_1.

NFA: example



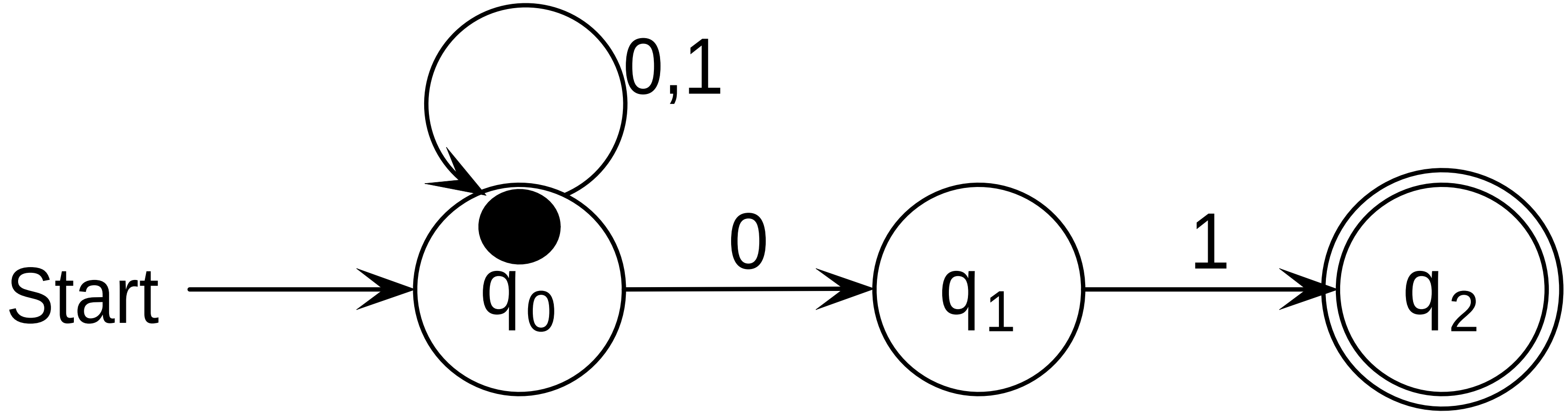
Suppose the input string is 00101. The NFA starts in state q_0, as indicated by the token.

NFA: example



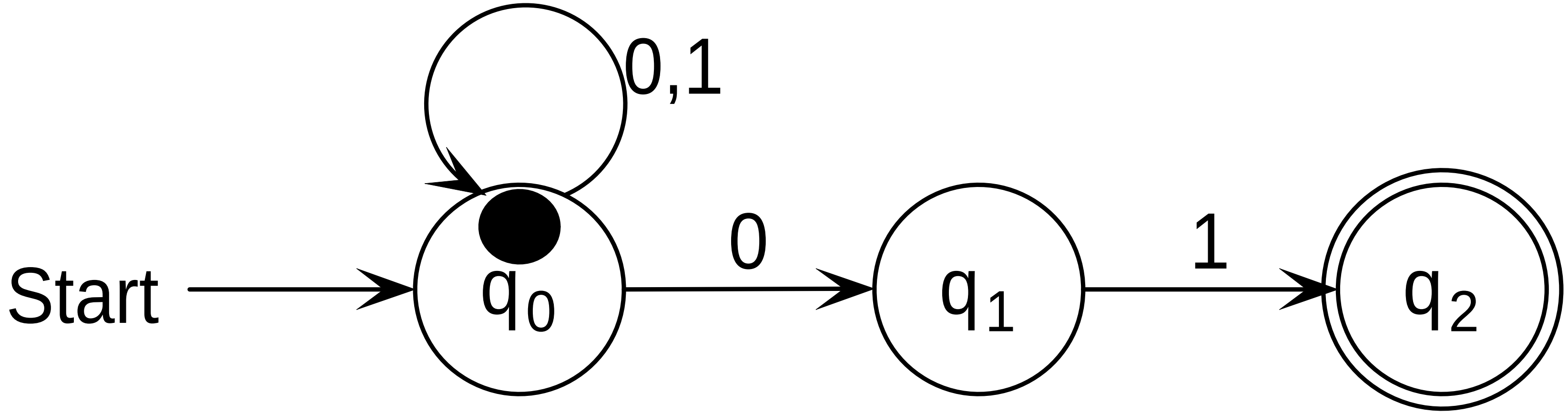
The remaining input string is 00101. The NFA reads the first symbol, 0. The resulting possible states are q_0 or q_1.

NFA: example



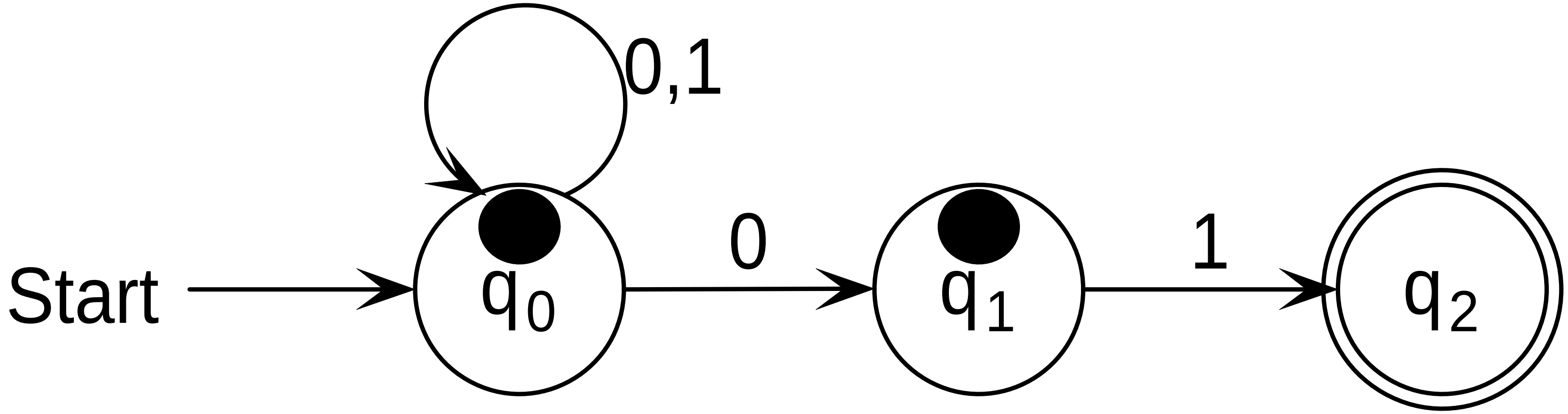
The remaining input string is 0101. The NFA reads the next symbol, 0. There is no transition for 0 from q_1. So that token “dies”, leaving only q_0 as a possible state.

NFA: example



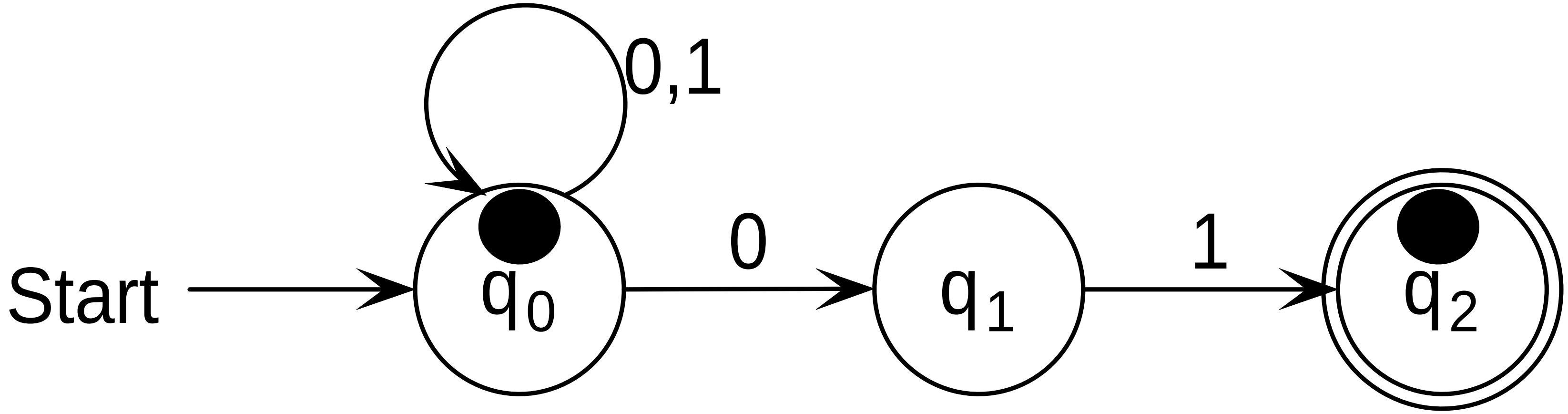
The remaining input string is 101. The NFA reads the next symbol, 1, leaving only q_0 as a possible state.

NFA: example



The remaining input string is 01. The NFA reads the next symbol, 0, and can be in state q_0 or q_1.

NFA: example



The remaining input string is 1. The NFA reads the next symbol, 1. The possible states are q_0 and q_2. Because one of the possible states is final, the NFA accepts.