Modelling Computation

Marco Carbone

IT University of Copenhagen

October 2014

Computational Models?

Which tasks can be carried out by a computer? Which can't?

Example. A program (a compiler) takes another program as input and computes an (executable) program.

Example. A program P takes another program Q as input and decides if Q terminates on any input.

What is a ***computational model***?

We shall consider three classical computational models in detail:

- Finite automata (or finite state machines)
- Grammars
- Turing machines

Strings and Languages (I)

- An **alphabet** Σ is a finite set of symbols.
- A **string** ω over Σ is a finite sequence of symbols from Σ.
- A **language** A is a set of strings over some Σ.
- ϵ is the **empty string**.

Example. If $\Sigma = \{0, 1\}$ then 100111 is a string over Σ and $|100111| = 6$. ϵ is a string over any Σ.

Strings and Languages (II)

- Given $\omega_1 = a_1 a_2 \ldots a_k$ and $\omega_2 = b_1 b_2 \ldots b_l$, $\omega_1\omega_2$ denotes their **concatenation** $a_1 a_2 \ldots a_k b_1 b_2 \ldots b_l$.

- Similarly, AB is the concatenation of two languages defined as $AB = \{\omega\omega' : \omega \in A$ and $\omega' \in B\}$.

- A^* denotes the set made by concatenating zero or more strings from A (the Kleene closure).

Example. Given $A = \{000, 0\}$ and $B = \{1\}$,

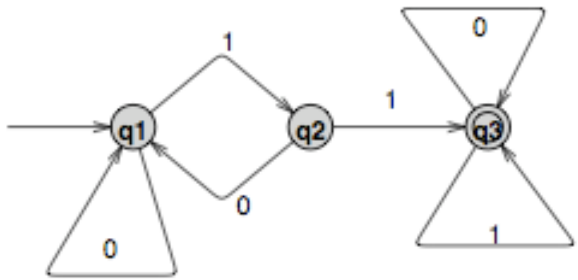
- $AB = \{0001, 01\}$

- $B^* = \{\epsilon, 1, 11, 111, 1111, 11111, \ldots\}$

Finite Automata (I)

Informally, a **Finite Automaton** (FA) consists of **states** and **transitions** between states. Only one **start** state, and some states are **accepting**.

Below is a **state diagram** of a FA M_1 with start state q_1 and accept state q_3: Beginning in q_1 M_1 processes an input string, say 000111,



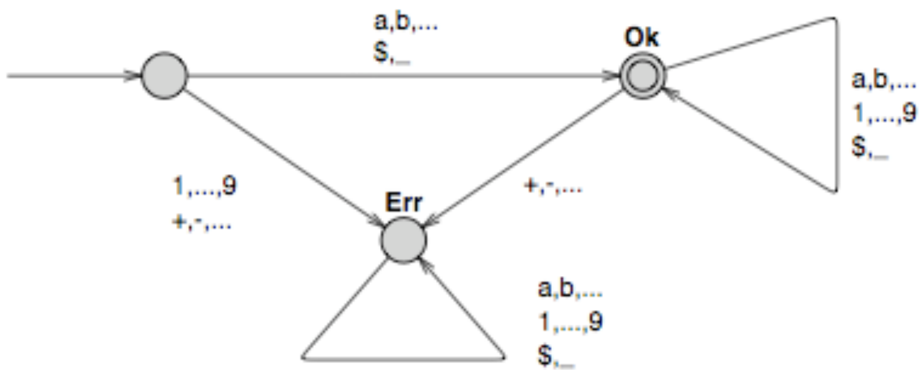
reading (from left to right) the next symbol and taking the matching transition. If it terminates in q_3 the output is **accept**, otherwise **reject**.

Exercise. Is 000111 accepted, is 10110?

Finite Automata (II)

Example A Java ***identifier*** is a sequence of letters and digits, starting with a letter. The symbols $ and _ (underscore) are letters. **Ok** is the accepting state.

The FA below is a ***recognizer*** for the language of Java identifiers.



Note. We used (abuse of notation) the shorthand of having multiple input symbols on one edge.

Finite Automata (III)

A **Finite Automaton** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

- Q is a finite set of ***states***,
- Σ is a finite set, the ***alphabet***,
- $\delta : Q \times \Sigma \to Q$ is the ***transition function***,
- q_0 is the ***start*** state, and
- $F \subseteq Q$ is the set of ***accept*** states.

Example. M_1 from slide 5 can be defined by:

$$(Q, \Sigma, \delta, q_0, F)$$

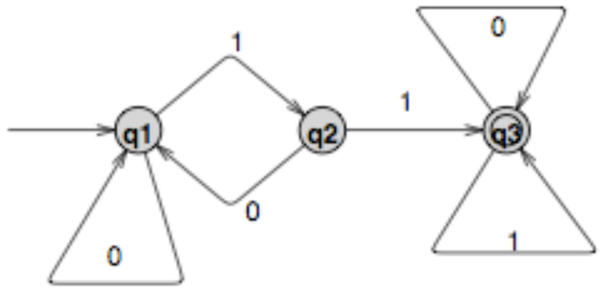
where $Q = \{q_1, q_2, q_3\}$, $\Sigma = \{0, 1\}$, $q_0 = q_1$, $F = \{q_3\}$, and

$\delta = \{(q_1, 0, q_1), (q_1, 1, q_2), (q_2, 0, q_1), (q_2, 1, q_3), (q_3, 0, q_3), (q_3, 1, q_3)\}$

Finite Automata (IV)

The set of all strings A accepted by a FA M is the **language** of M, denoted by $L(M) = A$. We say that M **recognizes** A.

Exercise. What's the language recognized by



Exercise. Define a recognizer for
$\{\omega \in \{0,1\}^* : \omega$ contain at least four 1's$\}$. Is there a recognizer for
$\{\omega \in \{0,1\}^* : \ \omega$ contains the substring 1010$\}$?

Exercise. Let $\Sigma = \{0,1\}$. Define a recognizer for $\emptyset$ and one for $\{\epsilon\}$.

Finite Automata (V)

The notion of acceptance and recognition can be defined as below.

Let $M = (Q, \Sigma, \delta, q_0, F)$ and let $\omega = a_1 \ldots a_k$ be a string over Σ. M **accepts** ω if there exists a sequence $r_0, r_1, \ldots, r_k \in Q$ such that

1. $r_0 = q_0$,
2. $\delta(r_i, a_{i+1}) = r_{i+1}$ for $i = 0, \ldots, k-1$, and
3. $r_k \in F$.

M **recognizes** A if $A = \{\omega : M \text{ accepts } \omega\}$

Example. The set of Java identifiers is recognized by some FA.

Example. $\{\omega \in \{0,1\}^* : \omega$ contains at least four 1's$\}$ is recognized by some FA.

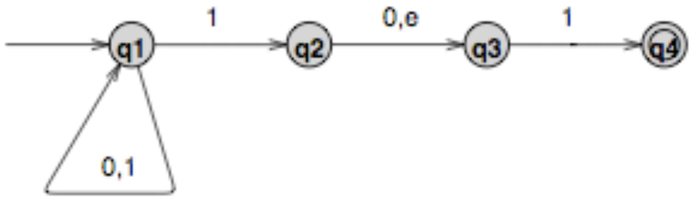
Example. $\emptyset$ and $\{\epsilon\}$ are recognized by some FA.

Non-deterministic FA's (I)

The FA's seen until now are all **deterministic**, in a given state and reading next input the next state is uniquely determined.

In a **non-deterministic** FA (NFA) several possible choices for the next state, and hence several possible computations, may exists.

Example. In q_1 N_1 below may chose between staying in q_1 or move to q_2 on input 1. In q_2 it may move immediately (due to $e = \epsilon$) to q_3 or stay. In q_3 it terminates on input 0.

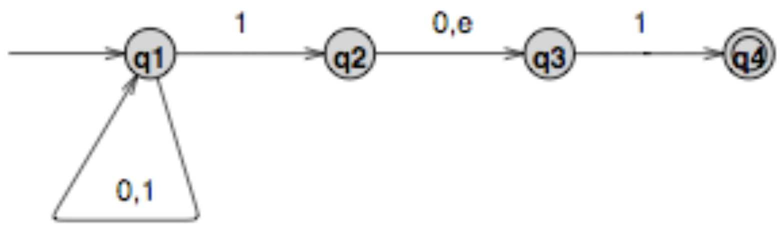


N_1 accepts ω if there is a computation accepting ω. N_1 accepts 101, 0111, and ...?

Non-deterministic FA's (II)

Viewed as ***parallel computation*** non-determinism lets several copies of the NFA run concurrently. When an NFA makes a choice it splits into as many copies as choices and each copy proceeds separately.

Example. N_1 executes the string 101 concurrently:

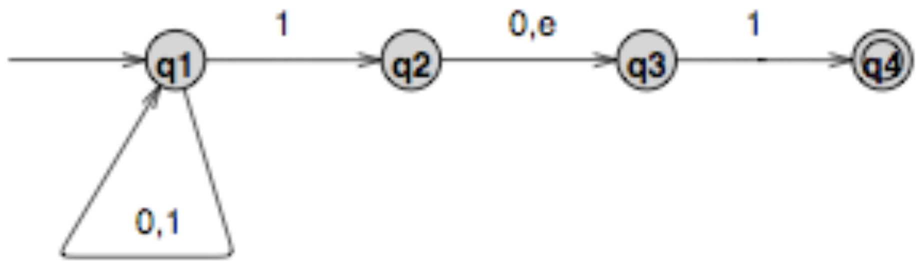


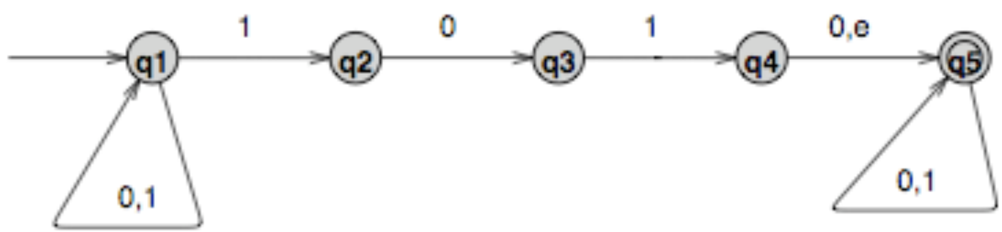
$$\begin{array}{rcl}\{(q_1, 101)\} & \to & \{(q_1, 01), (q_2, 01), (q_3, 01)\} \\ & \to & \{(q_1, 1), (q_3, 1), (q_3, 01)\} \\ & \to & \{(q_1, \epsilon), (q_2, \epsilon), (q_4, \epsilon), (q_3, 01)\}\end{array}$$

$\to$ symbolizes a concurrent computation step, and each set contains pairs of the current state and the remaining input for each concurrent computation instance.

Non-deterministic FA's (III)

Exercise. What's the language recognized by N_1 and by N_2





Non-deterministic FA's (IV)

Formally a NFA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

- Q is a finite set of ***states***,
- Σ is a finite set, the ***alphabet***,
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)$ is the ***transition function***,
- q_0 is the ***start*** state, and
- $F \subseteq Q$ is the set of ***accept*** states.

Example. N_1 from above can be defined by: $(Q, \Sigma, \delta, q_0, F)$ where $Q = \{q_1, q_2, q_3, q_4\}$, $\Sigma = \{0, 1\}$, $q_0 = q_1$, $F = \{q_4\}$, and

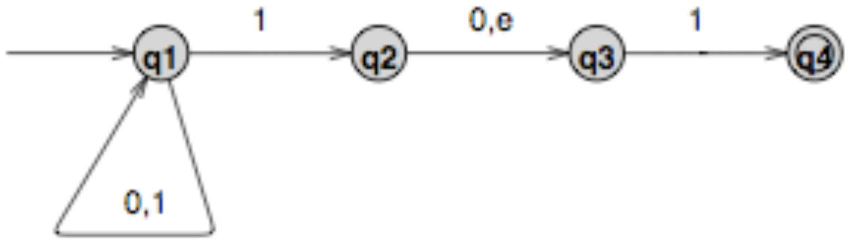
$$\begin{array}{rcll} \delta & = & \{ & (q_1, 0, \{q_1\}),\ (q_1, 1, \{q_1, q_2\}),\ (q_1, \epsilon, \emptyset), \\ & & & (q_2, 0, \{q_3\}),\ (q_2, 1, \emptyset),\ (q_2, \epsilon, \{q_3\}), \\ & & & (q_3, 0, \emptyset),\ (q_3, 1, \{q_4\}),\ (q_3, \epsilon, \emptyset), \\ & & & (q_4, 0, \emptyset),\ (q_4, 1, \emptyset),\ (q_4, \epsilon, \emptyset) \qquad\qquad \} \end{array}$$

Equivalence of FA's and NFA's

NFA's seems to have more computational power than FA's, but . . .

Theorem. Every NFA has a FA recognizing the same language.

Example. How to determinise



Idea: Construct a FA that simulates it! The initial state should be $r_1 = \{q_1\}$.

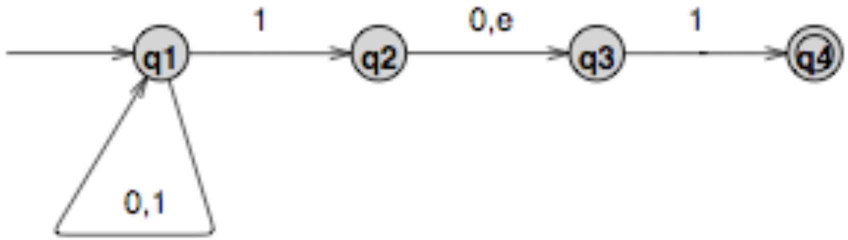
$$(r_1, 0, r_1),\ (r_1, 1, \{q_1, q_2, q_3\})$$

Equivalence of FA's and NFA's

NFA's seems to have more computational power than FA's, but

Theorem. Every NFA has a FA recognizing the same language.

Example. How to determinise



Idea: Construct a FA that simulates it! The initial state should be $r_1 = \{q_1\}$. Let $r_2 = \{q_1, q_2, q_3\}$

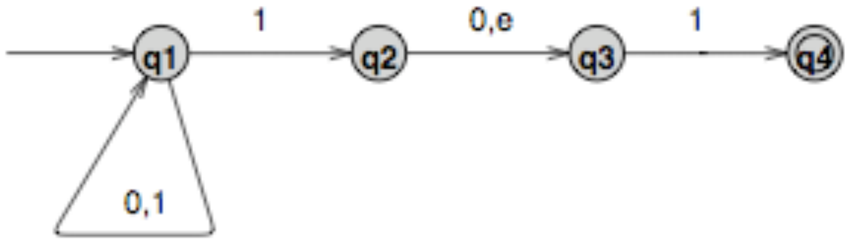
$$(r_1, 0, r_1),\ (r_1, 1, r_2),\ (r_2, 0, \{q_1, q_3\}),\ (r_2, 1, \{q_1, q_2, q_3, q_4\})$$

Equivalence of FA's and NFA's

NFA's seems to have more computational power than FA's, but

Theorem. Every NFA has a FA recognizing the same language.

Example. How to determinise



Idea: Construct a FA that simulates it! The initial state should be $r_1 = \{q_1\}$. Let $r_2 = \{q_1, q_2, q_3\}$, and let $r_3 = \{q_1, q_3\}$ and $r_4 = \{q_1, q_2, q_3, q_4\}$.

$$(r_1, 0, r_1),\ (r_1, 1, r_2),\ (r_2, 0, r_3),\ (r_2, 1, r_4)$$
$$(r_3, 0, r_1),\ (r_3, 1, r_4),\ (r_4, 0, r_3),\ (r_4, 1, r_4)$$

The accepting state is r_4 because $q_4 \in r_4$.

Equivalence of FA's and NFA's

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA recognizing A, define a FA recognizing A by:

$$M = (Q', \Sigma, \delta', {q_0}', F')$$

where

- $Q' = \mathcal{P}(Q)$
- ${q_0}' = \{q_0\}$
- $\delta'(R, a) = \{q \in Q : q \in E(\delta(r, a))$ for some $r \in R\}$
- $F' = \{R \in Q' : R$ contains a state in $F\}$

where

$$E(R) = \{q \in Q : q \text{ can be reached from } R \text{ by } 0 \text{ or more } \epsilon \text{ arrows}\}$$