Nondeterministic Finite Automata (NFA)

(LECTURE 4)

Introduction

- Non deterministic finite automata
- Language accepted by a NFA
- String accepted by Non Deterministic finite automata

Nondeterminism

- An important notions(or abstraction) in computer science
- refer to situations in which the next state of a computation is not uniquely determined by the current state.
  - Ex: find a program to compute max(x,y):
  - pr1: case $x \geq y$ => print x;
  - $y \geq x$ => print y
  - endcase;
  - Then which branch will be executed when x = y ?
  - ==> *don't care nondeterminism*
  - Pr2: do-one-of {
  - {if x < y fail; print x},
  - {if y < x fail, print y} }.
  - ==>The program is powerful in that it will never choose branches that finally lead to ‘fail’ -- an unrealistic model.
  - ==> *don't know nondeterminism*.

nondeterminism (cont'd)

- a nondeterministic sorting algorithm:
- nondet-sort(A, n)
  - 1. for i = 1 to n do
  - 2. nondeterministically let k := one of {i, ..., n} ;
  - 3. exchange A[i] and A[k]
  - 4. endfor
  - 5 for i = 1 to n-1 do if A[i] > A[i+1] then fail;
  - 6. return(A).
  - Notes: 1. Step 2 is magic in that it may produce many possible outcomes. However all incorrect results will be filtered out at step 5.
  - 2. The program run in time NTIME O(n)
  - cf: O(n lg n) is required for all sequential machines.

nondeterminism (cont'd)

- Causes of nodeterminism in real life:
  - incomplete information about the state
  - external forces affecting the course of the computation
  - ex: the behavior of a process in a distributed system
- Nondeterministic programs *cannot be executed directly but can be simulated by real machine*.
- Nondeterminism can be used as a tool for the specification of problem solutions.
- an important tool in the design of efficient algorithms
  - There are many problems with efficient nondeterministic algorithm but no known efficient deterministic one.
  - the open problem NP = P ?
- How to make DFAs become nondeterministic ?

  ==> allow multiple transitions for each state-input-symbol pair

  ==> modify the transition function d.

Formal Definition of NFAs

- A NFA is a five-tuple N = (Q,S,d,S,F) where everything is the same as in a DFA, except:
  - S ⊆ Q is a set of ***starting states***, instead of a single state.
  - d is the ***transition function*** d: Q x S -> 2^Q. For each state p and symbol a, d(p,a) is the set of all states that N is allowed to move from p in one step under input symbol a.
  - diagrammatic notation: p --a--> q

$$p \xrightarrow{a} q$$

Note: d(p,a) can be the empty set

- The extended transition function D (multi-step version of d) for NFA can be defined analogously to that of DFAs:

D: 2^QxS* -> 2^Q is defined inductively as follows:

1. Basis: D(A, e) = _____ for every set of states A (6.1)
2. Ind. case: D(A, xa) = _____ for every x ∈ S* and a ∈ S (6.2)

Note: Intuitively q ∈ D(A,x) means q can be reached from some state ∈ A after scanning input string x.

Languages accepted by NFAs

- Note: Like DFAs, the extended transition function D on a NFA N is uniquely determined by N.
  - pf: left as an exercise.
- N = (Q,S,d,S,F) : a NFA;   x: any string over S;

  D: the extended transition function of N.

1. x is said to be *accepted* by N if D(S,x) $\cap$ F $\neq$ {}
   - i.e., x is accepted if there is an accept state q $\in$ F such that q is reachable from a start state under input string x (i.e., q $\in$ D(S,x))
2. The set (or language) accepted by N, denoted L(N), is the set of all strings accepted by N. i.e.,
   - L(N) $=_{def}$ {x $\in$ S* | N accepts x }.
3. Two finite automata (FAs, no matter deterministic or nondeterministic) M and N are said to be equivalent if L(M) = L(N).

Equivalence of FAs

Note: under such definition, every DFA M = (Q,S,d,s,F) is equivalent to an NFA N = (Q,S,d',{s},F) where

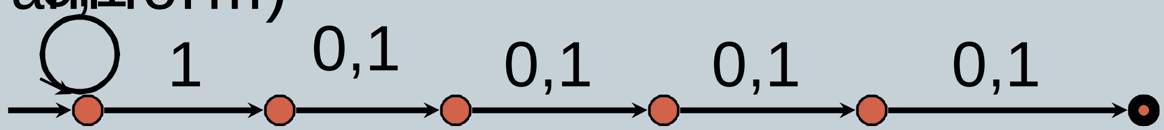
- d'(p,a) = {d(p,a)} for every state p and input a.

- Problem: Does the converse hold as well ?
  - i.e. For every NFA N there is a DFA M s.t. L(M) = L(N).
  - Ans: _____

Some examples of NFAs

Ex: Find a NFA accepting A = { x $\in$ {0,1}* | the fifth symbol from the right is 1 } = {010000, 11111,...}.

Sol: 1. (in diagram form)

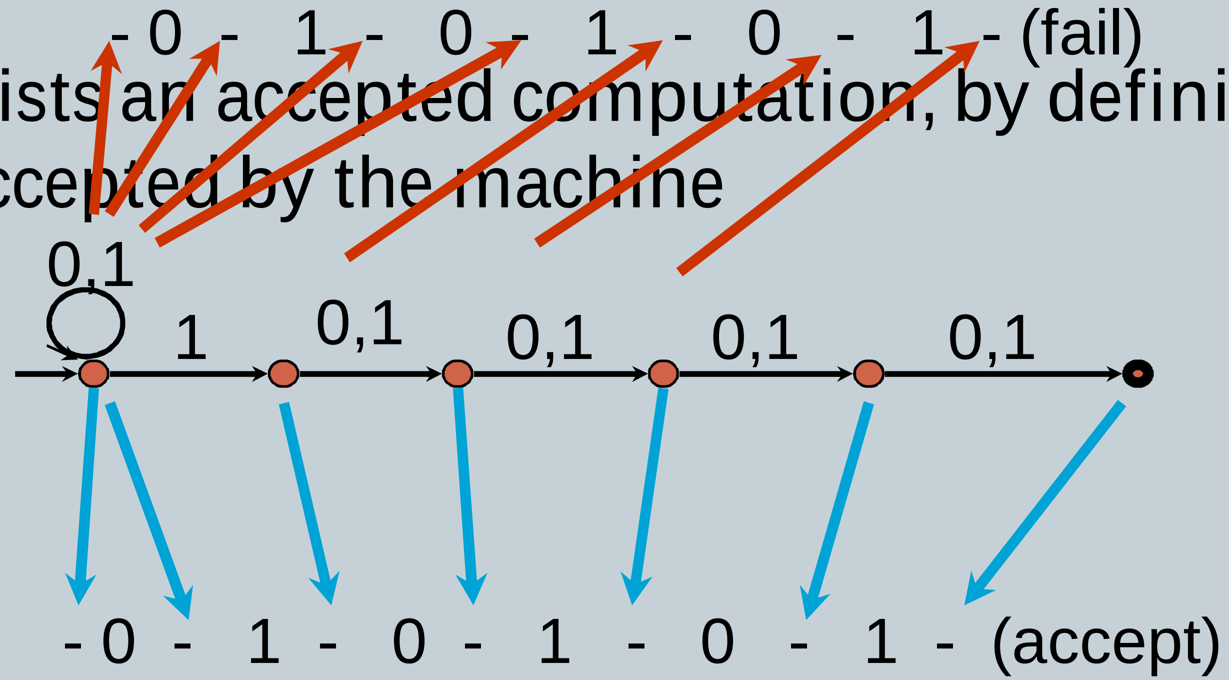


2: tabular form:

3. tuple form: (Q,S,d,S,F) = (__,__,__,__,__).

Example of strings accepted by NFAs

- Note: there are many possible computations on the input string: 010101, some of which reach the (only) final state (accepted or successful computation), some of which do not (fail).
- Since there exists an accepted computation, by definition, the string is accepted by the machine



Some properties about the extended transition function D

- Lem 6.1: D(A,xy) = D(D(A,x),y).
- pf: by inducition on |y|:
  1. |y| = 0 => D(A,xe) = D(A,x) = D(D(A,x),e) -- (6.1).
  2. y = zc => D(A,xz c) = $U_{q \in D(A,xz)}$ d(q,c) -- (6.2)

     = $U_{q \in \Delta(D(A,x),z)}$ d(q,c) -- ind. hyp.

     = D(D(A,x),zc) -- (6.2)
- Lem 6.2 D commutes with set union:
  - i.e., D ($U_{i \in I}$ A_i,x) = $U_{i \in I}$ D(A_i,x). in particular, D(A,x) = $U_{p\in A}$ D({p},x)
- pf: by ind. on |x|. Let B = $U_{i \in I}$ A_i

1. |x|= 0 => D ($U_{i \in I}$ A_i, e) = $U_{i \in I}$ A_i = $U_{i \in I}$ D(A_i, e) -- (6.1)

2. x = ya => D ($U_{i \in I}$ A_i, ya) = $U_{p \in D(B,y)}$ d(p,a) -- (6.2)

= $U_{p\in U_{i\in I} D(Ai,y)}$ d(p,a) -- ind. hyp. = $U_{i\in I}U_{p\in D(Ai,x)}$ d(P,a) -- set theory = $U_{i \in I}$ D(A_i,ya) (6.2)

The subset construction

- $N = (Q_N, S, d_N, S_N, F_N)$: a NFA.
- $M = (Q_M, S, d_M, s_M, F_M)$ (denoted 2^N): a DFA where
  - $Q_M = 2^{Q_N}$
  - $d_M(A,a) = D_N(A,a)$ ($= \cup_{q \in A} d_N(q,a)$) for every $A \subseteq Q_N$.
  - $s_M = S_N$ and
  - $F_M = \{A \subseteq Q_N \mid A \cap F_N \neq \{\}\}$.
  - note: States of M are subsets of states of N.
- Lem 6.3: for any $A \subseteq Q_N$. and x in S^*, $D_M(A,x) = D_N(A,x)$.

pf: by ind on $|x|$. if $x = e \Rightarrow D_M(A,e) = A = D_N(A,e)$. --(def)

if $x = ya \Rightarrow D_M(A,ya) = d_M(D_M(A,y),a)$ -- (def) $= d_M(D_N(A,y),a)$ --

ind. hyp. $= D_N(D_N(A,y),a)$ -- def of d_M $= D_N(A, ya)$ -- lem 6.1

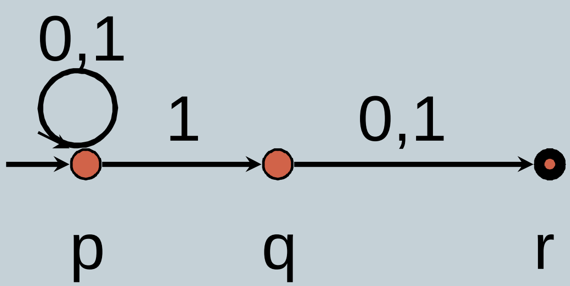
Theorem 6.4: M and N accept the same set.

pf: $x \in L(M)$ iff $D_M(s_M,x) \in F_M$ iff $D_N(S_N,x) \cap F_N \neq \{\}$ iff $x \in L(N)$.

Equivalence of NFAs and DFAs - an example

1. NFA N accepting A = { $x \in \{0,1\}^*$ | the second symbol from the right is 1 } = {$x1a$ | $x \in \{0,1\}^*$ and $a \in \{0,1\}$ }.

sol:



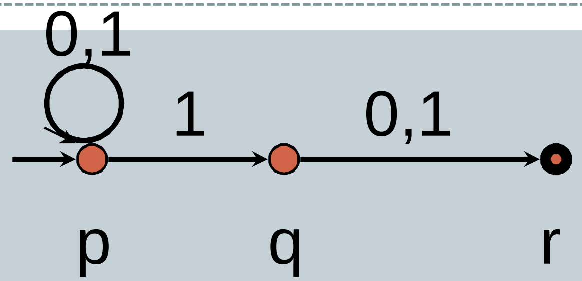
		0	1
	{}	{}	{}
->	{p}	{p}	{p,q}
	{q}	{r}	{r}
	{r}F	{}	{}
	{p,q}	{p,r}	{p,q,r}
	{p,r}F	{p}	{p,q}
	{q,r}F	{r}	{r}
	{p,q,r}F	{p,r}	{p,q,r}

2. DFA M equivalent to N is given as :

3. some states of M are redundant in the sense that they are never reachable from the start state and hence can be removed from the machine w/o affecting the languages accepted.

A more human friendly method

sol:



		0	1
	{}	{}	{}
->	{p}	{p}	{p,q}
	{q}	{r}	{r}
	{r}F	{}	{}
	{p}	{p}	{p,q}
	{p,q}	{p,r}	{p,q,r}
	{p,r}F	{p}	{p,q}
	{q,r}F	{r}	{r}
	{p,q,r}F	{p,r}	{p,q,r}

1. Copy the transition table
2. add Row(S) /* $=_{def}$

   $Sum_{p \in S}$ Row(p) to table */

3. D={X|X in Row(p).tail } – {S} // S is the initial set of states
4. While D != {} do {

   S1 = D.pop() ; // remove any element from D.

   add(Row(S1)) to table

   D = D U Row(S1).tail.

}

e-transition

- Another extension of FAs, useful but adds no more power.
- An e-transition is a transition with label e, a label standing for the empty string e.

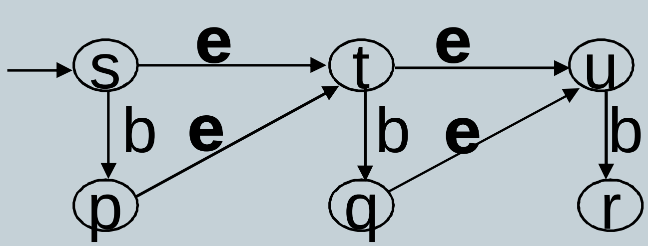
$$p \xrightarrow{\mathbf{e}} q$$

- The FA can take such a transition anytime w/o reading an input symbol.

Ex 6.5 : The set accepted by the FA is {b,bb,bbb}.

Ex 6.6 : A NFA-e accepting the set {x ∈ {a}* | |x| is dividable by 3 or 5 }.

- real advantage of e-transition:
  - convenient for specification
  - add no extra power



Ex6.5

NFA-e

- N = (Q,S,d,S,F) : a NFA-e,where
  - Q, S, S and F are the same as NFA,
  - d : Q x (SU {e}) -> 2^Q.
- The set Eclosure(A) is the set of ref. and transitive closure of the e-transition of A =

  { q ∈ Q |∃ e-path p – p_1 – p_2 ... -p_n with p ∈ A and p_n = q }

Note: Eclosure(A) (abbreviated as EC(A)) = EC(EC(A)).

- The multistep version of d is modified as follows:
  - D: 2^Q x S* → 2^Q where, for all A ⊆ Q , y ∈ S*, a ∈ A
  - D(A, e) = Eclosure(A)
  - D(A, ya) = $U_{p \in D(A,y)}$ Eclosure(d(p,a))
- L(N) = { x | D(S), x) ∩ F ≠ {} } //The language accepted by N

E-closure

- Eclosure(A) is the set of states reachable from states of A without consuming any input symbols,

(i.e., q∈Eclosure(A) iff∃p∈ A s.t. q ∈ D(p, e^k) for some k ≥ 0).

- Eclosure(A) can be computed as follows:

```
1. R=F={}; nF=A; //F: frontier; nF: new frontier
2. do {  R = R U nF; F = nF; nF={};
3.   For each q ∈ F do
4.         nF = nF U (d(q,e)- R)
5. }while nF ≠ {};
6. return R
```

Note:1. q ∈ D(A, e^k) => q ∈ R after k-th iteration of the program.

2. We can precompute the matrix T* where T is the e-transition matrix of the NFA. and use the result to get Eclosure(A) for all required As.

The subset construction for NFA-e

- $N = (Q_N,S,d_N,S_N,F_N)$: a NFA-e.where $d_N : Q \times (S \cup \{e\}) \to 2^Q$.
- $M = (Q_M,S,d_M,s_M,F_M)$ (denoted 2^N): a DFA where
  - $Q_M = \{ EC(A) \mid A \subseteq Q_N \}$
  - $d_M(A,a) = \bigcup_{q \in Ec(A)} EC(d_N(q,a))$ for every $A \in Q_M$.
  - $s_M = EC(S_N)$ and
  - $F_M = \{A \in Q_M \mid A \cap F_N \neq \{\}\}$.
  - note: States of M are subsets of states of N.
- Lem 6.3: for any $A \subseteq Q_N$. and $x \in S^*$, $\Delta_M(A,x) = \Delta_N(A,x)$.

pf: by ind on $|x|$. if $x = e \Rightarrow \Delta_M(A,e) = A = EC(A) = \Delta_N(A,e)$. --(def)

if $x = ya \Rightarrow \Delta_M(A,ya) = d_M(\Delta_M(A,y),a)$ -- (def)

$= d_M(\Delta_N(A,y),a)$ -- ind. hyp.

$= \bigcup_{q \in \Delta N(A,y)} EC(d_N(q,a))$ -- def of d_M

$= \Delta_N(A, ya)$ – def of Δ_N

Theorem 6.4: M and N accept the same set.

pf: $x \in L(M)$ iff $\Delta_M(s_M,x) \in F_M$ iff $\Delta_N(EC(S_N),x) \cap F_N \neq \{\}$ iff $x \in L(N)$.

More closure properties

- If A and B are regular languages, then so are AB and A*.
- $M = (Q_1,S,d_1,S_1,F_1)$, $N=(Q_2,S,d_2,S_2,F_2)$: two NFAs
- The machine $M \bullet N$, which firstly executes M and then execute sN, can be defined as follows:
- $M \bullet N =_{def} (Q, S, d, S, F)$ where
  - Q = disjoint union of Q_1 and Q_2,
  - $S = S_1$,
  - $F = F_2$,
  - $d = d_1 \cup d_2 \cup \{ (p, e, q) \mid p \in F_1 \text{ and } q \in S_2 \}$
- Lem: 1. $x \in L(M)$ and $y \in L(N)$ then $xy \in L(MN)$

  2. $x \in L(MN) \Rightarrow \$\, y,z$ s.t. $x = yz$ and $y \in L(M)$ and $z \in L(N)$.

  Corollary: $L(MN) = L(M)\, L(N)$

M* machine

- $M = (Q_1, S, d_1, S_1, F_1)$: a NFA
- The machine M*, which executes M a nondeterministic number of times, can be defined as follows:
- $M^* =_{def} (Q, S, d, S, F)$ where
  - $Q = Q \cup \{s,f\}$, where s and f are two new states $\notin Q$
  - $S = \{s\}$, $F = \{f\}$,
  - $d = d_1 \cup \{(s, e, f)\} \cup \{(s,e,p) \mid p \in S_1\} \cup \{(q,e,s) \mid q \in F_1\}$

Theorem: L(M*) = L(M)*