COMS3261: Computer Science Theory

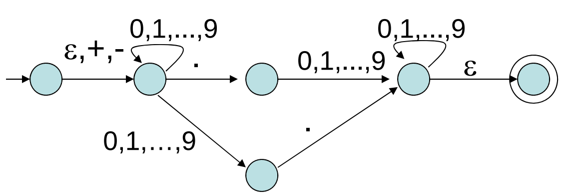
Spring 2013

Mihalis Yannakakis

Lecture 4, 9/16/13

ε–NFA: NFA with ε transitions

- ε transitions: spontaneous, silent moves
- Modeling e.g. local internal invisible moves of processes



- Label of a path: Sequence of input symbols only; ε omitted (silent)
- Language of ε-NFA: Set of labels of all accepting paths
- Example: +3.5, -2. , .40

Definition of ε-NFA

- Similar to definition of NFA, except transitions also on ε
- $A = (Q, \Sigma, \delta, q_0, F)$
- transition function $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q = \mathcal{P}(Q)$

  i.e., for each q in Q, and each a in $(\Sigma \cup \{\varepsilon\})$,

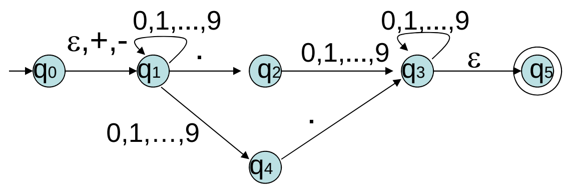
  $\delta(q,a) \subseteq Q$ is a set of 0, 1 or more states
- Alternatively (equivalently), can represent it as a *transition relation* $R = \{(q,a,p) \mid p \in \delta(q,a)\}$

Representation of ε-NFA

- Transition Diagram: Nodes = States, Labeled edges = tuples of transition relation
- Accepting path (computation) : path from start state q_0 to a state in F.
- Input string x is accepted by A iff there is an accepting path labeled by x (where ε is omitted in the label of path)
- Language of A, L(A) = set of labels of all accepting paths

- Transition Table: Same as for NFA except that we have also a column for ε

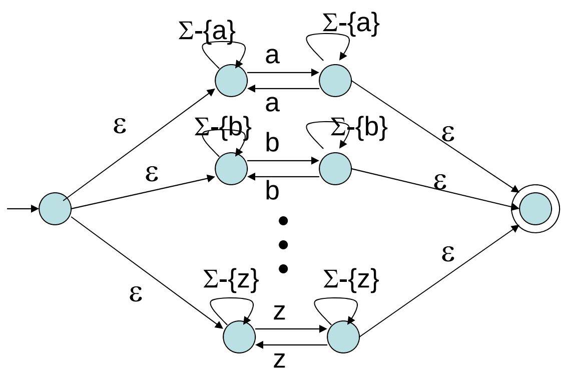
Transition table -example



	ε	+, -	.	0,1,…,9
$\rightarrow q_0$	$\{q_1\}$	$\{q_1\}$	$\varnothing$	$\varnothing$
q_1	$\varnothing$	$\varnothing$	$\{q_2\}$	$\{q_1,q_4\}$
q_2	$\varnothing$	$\varnothing$	$\varnothing$	$\{q_3\}$
q_3	$\{q_5\}$	$\varnothing$	$\varnothing$	$\{q_3\}$
q_4	$\varnothing$	$\varnothing$	$\{q_3\}$	$\varnothing$
$*q_5$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$

Example of ε-NFA

- Set of strings over Σ={a,b,…,z} that contain an odd number of some letter



ε-Closure

ECLOSE(state q) = all states that q can reach with a sequence of ε-transitions (zero, one, or more transitions)

Transitive Closure in the subgraph Gε of the transition graph that contains only the ε-transitions

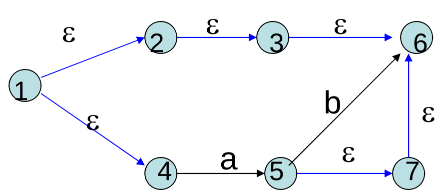
Computed inductively as in graph search.

Basis: q is in ECLOSE(q)

Induction: If $p \in$ ECLOSE(q) and $r \in \delta(p,\varepsilon)$

then r is in ECLOSE(q) (i.e., add all of $\delta(p,\varepsilon)$)

Example of ε-CLOSE



ECLOSE(1) = {1,2,3,4,6}

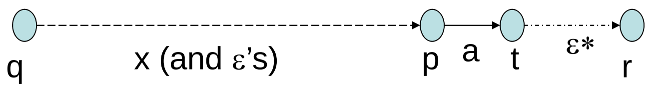
ECLOSE(4)={4}

ECLOSE(5)={5,6,7}

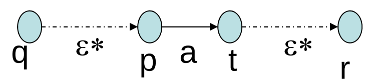
Extension of δ to strings

- $\hat{\delta}(q,w)$ = set of states r such that there is a path from q to r labeled w (recall: we omit ε's in label of path)
- Inductive definition/computation:
- Basis: $\hat{\delta}(q,\varepsilon)$ = ECLOSE(q)
- Induction: $\hat{\delta}(q,xa) = \bigcup_{t \in \delta(\hat{\delta}(q,x),a)}$ ECLOSE(t)

  for $x \in \Sigma^*$, $a \in \Sigma$



Example: $\hat{\delta}(q,a)$:



L(ε-NFA) = $\{ w \in \Sigma^* \mid \hat{\delta}(q_0,w) \cap F \neq \emptyset \}$

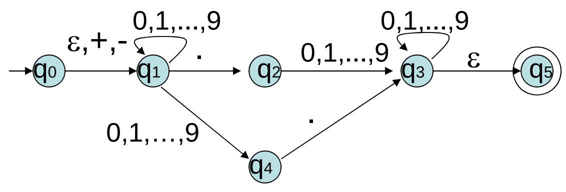
ε-NFA to NFA Translation

- Theorem: For every ε-NFA N, we can construct an equivalent NFA B (without ε transitions) that accepts the same language, and hence an equivalent DFA D
- Construction of NFA B: Same set of states Q, start state q_0
- Transition function of B: $\delta_B(q,a) = \hat{\delta}_N(q,a)$, $a \in \Sigma$
- Accepting set of states: same set F as N, except that if $\varepsilon \in L(N)$ (i.e. ECLOSE(q_0)$\cap F \neq \emptyset$) then add q_0 to accepting set
- Can construct DFA D from B by usual subset construction
- Or can construct DFA D also directly from the ε-NFA N

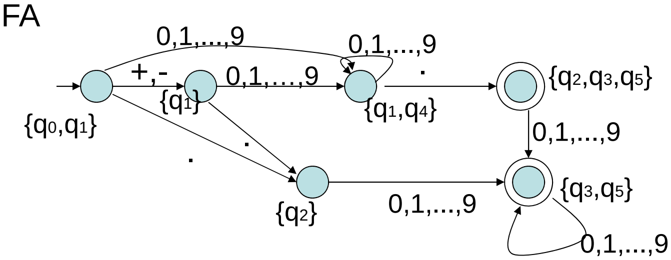
Direct translation ε-NFA to DFA

- Subset construction: *Like the NFA-to-DFA construction except that in each step of the construction we take the ε-closure of all the states.*
- Given ε-NFA $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$,
  Construct DFA $D = (Q_D, \Sigma, \delta_D, q_D, F_D)$
- Q_D = set of ε-closed subsets of Q_N, i.e. sets S such that ECLOSE(S) (= $\cup_{p \in S}$ECLOSE(p)) = S
  (Could also take the set of all subsets of Q_N - but only the closed subsets are reachable)
- q_D= ECLOSE(q_0)
- $F_D = \{ S \in Q_D \mid S \cap F_N \neq \emptyset \}$
- $\delta_D(S,a)$ = ECLOSE($\cup_{p \in S}\, \delta_N(p,a)$) = $\cup_{p \in S}$ ECLOSE($\delta_N(p,a)$)
  i.e. if $S=\{p_1,..,p_k\}$, compute $\delta_N(S,a) = \delta_N(p_1,a) \cup \ldots \cup \delta_N(p_k,a)$
  compute ECLOSE(t) for each t in $\delta_N(S,a)$ and union the resulting sets

Example



DFA



Missing transitions go to the dead (rejecting) state ∅

Proof of ε-NFA → DFA translation

Show by induction on the length of an input string *w* the following

- Claim: $\hat{\delta}_N(q_0, w) = \hat{\delta}_D(q_D, w)$

  = set of nodes reachable in N from q_0 by path with label w

The claim implies the correctness of the translation:

For every string *w*, *w* is accepted by the NFA N iff

$$\hat{\delta}_N(q_0, w) \cap F_N \neq \varnothing$$

$$\Leftrightarrow \hat{\delta}_D(q_D, w) \cap F_N \neq \varnothing$$

$$\Leftrightarrow \hat{\delta}_D(q_D, w) \in F_D$$

$\Leftrightarrow$ *w* is accepted by the DFA D

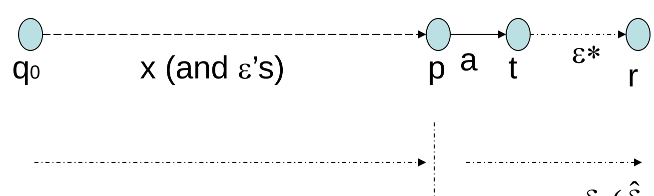
Proof of Claim $\hat{\delta}_N(q_0, w) = \hat{\delta}_D(q_D, w)$

- Basis: *w* = ε . By definition of extension of δ functions to strings: $\hat{\delta}_N(q_0, \varepsilon) = ECLOSE(q_0) = q_D = \hat{\delta}_D(q_D, \varepsilon)$

- Induction step: w =xa for some x∈Σ* , a∈Σ

  Induction hypothesis says: $\hat{\delta}_N(q_0, x) = \hat{\delta}_D(q_D, x) =$

  = set of nodes reachable from q_0 by paths with label x



$\hat{\delta}_N(q_0, x) = \hat{\delta}_D(q_D, x)$

$\delta_D(\hat{\delta}_D(q_D, x), a) =$

$\hat{\delta}_D(q_0, xa)$

Finite Automata Relations Summary

DFA, NFA, ε-NFA accept the same set of languages, regular languages, but their size (#states) may differ

$$
\begin{tikzcd}
\varepsilon\text{-NFA} \arrow[d, shift right=2, "\begin{array}{c}\text{linear}\\(\varepsilon\text{-close})\end{array}"'] \\
\text{NFA} \arrow[u, shift right=2, "\text{linear}"'] \arrow[d, shift right=2, "\begin{array}{c}\text{exponential}\\\text{(subset}\\\text{construction)}\end{array}"'] \\
\text{DFA} \arrow[u, shift right=2, "\text{linear}"']
\end{tikzcd}
$$

Operations on Languages

- Union: $L \cup M = \{ x \mid x \in L \text{ or } x \in M \}$
- Concatenation: L.M or LM = $\{ x.y \mid x \in L \text{ and } y \in M \}$
  i.e., LM = $\{ w \mid \exists x \in L \text{ and } \exists y \in M \text{ such that } w=x.y \}$

Examples:
{red,green}.{ball,toy}={redball,greenball,redtoy,greentoy}
{aba,ab}.{a,aa}={abaa,abaaa,aba}
abaa can be written in two ways as x.y with $x \in L$, $y \in M$
$\{\varepsilon\}L = L\{\varepsilon\} = L$, for every language L
$\emptyset L = L\emptyset = \emptyset$, for every language L

Operations on Languages, ctd.

- Powers: $L^0 = \{\varepsilon\}$; $L^1 = L$; $L^{i+1} = L.L^i$ for all $i \geq 1$
- Kleene closure or * (star) operation:

$$L^* = L^0 \cup L^1 \cup L^2 \cup \ldots = \bigcup_i L^i$$

$$= \{ w \mid w=\varepsilon \text{ or } \exists k \geq 1 \text{ and } \exists \text{ strings } x_i \in L \text{ for } i=1,\ldots,k \text{ such that } w = x_1 \ldots x_k \}$$

Examples:

$\{\varepsilon\}^* = \{\varepsilon\}$

$\varnothing^* = \{\varepsilon\}$

$\{0\}^* = \{\varepsilon,0,00,000,\ldots\} = \{ 0^i \mid i=0,1,2,\ldots\}$

REGULAR EXPRESSIONS

- An algebraic way of defining languages
- Used in pattern matching (eg. grep), lexical analysis (eg. lex)

- Each expression E denotes a language L(E)
- Expressions built inductively from
  - the constant expressions $\varnothing$, $\{\varepsilon\}$, a , for all $a \in \Sigma$
  - using operations + (union), . (concatenation), * (star)

Regular expressions

	Expression	Language
Basis:	$\varnothing$	$\varnothing$
	ε	$\{ \varepsilon \}$
	a, $\forall a \in \Sigma$	$\{ a \}$
Induction:	(E)	L(E)
(Operations)		
Union	E+F	$L(E) \cup L(F)$
Concatenation	E.F or EF	L(E).L(F)
Kleene *	E*	(L(E))*

Precedence order

- Order : * , . , +
- Example: a+bc* = a + (b. (c*))
- . , + are associative , so can omit (’s
  e.g. a+b+c = (a+b)+c or a+(b+c) , same language
- caution: . is not commutative (a.b not same as b.a)

Examples

- 0100 : the singleton set {0100}
- 0*: all strings of 0's (including the empty string)
- (0+1)*: all binary strings, including the empty string
- 0*1* : a sequence of 0's (possibly none) followed by a sequence of 1's (possibly none):

  - includes ε, 0, 1, 01, 00 … but not 10
- (0*1*)* = (0+1)* : all binary strings
- 0+10* : {0, 1, 10, 100, 1000 ,…}
- 0(1+0)* : all binary strings that start with 0
- ((0+1) (0+1))* : all binary strings of even length