Are NFA more powerful than DFA?

- NFA can solve every problem that DFA can (DFA are also NFA)
- Can DFA solve every problem that NFA can?

Equivalence of NFA, DFA

- Pages 54-58 (Sipser, 2nd ed)
- We will prove that every NFA is equivalent to a DFA (with upto exponentially more states).
- Non-determinism does not help FA's to recognize more languages!

Epsilon Closure

- Let $N=(Q,\Sigma,\delta,q_0,F)$ be any NFA
- Consider any set $R \subseteq Q$
- $E(R) = \{q|q$ can be reached from a state in R by following 0 or more ε-transitions$\}$
- E(R) is the epsilon closure of R under ε-transitions

Proving equivalence

For all languages $L \subseteq \Sigma^*$

$$L = L(N) \qquad \textit{iff} \qquad L = L(M)$$

for some	for some
NFA N	DFA M

One direction is easy:

A DFA M is also a NFA N. So N does not have to be \`constructed' from M

Proving equivalence – contd.

The other direction:

Construct M from N

- $N = (Q,\Sigma,\delta,q_0,F)$
- Construct $M = (Q',\Sigma,\delta',q'_0,F')$ such that,
  - for any string $w \in \Sigma^*$,
  - w is accepted by N iff w is accepted by M

Special case

- Assume that ε is not used in the NFA N.
  - Need to keep track of each subset of N
  - So $Q' = \mathcal{P}(Q)$, $q'_0 = \{q_0\}$
  - $\delta'(R,a) = \cup(\delta(r,a))$ over all $r \in R$, $R \in Q'$
  - $F' = \{R \in Q' \mid R$ contains an accept state of $N\}$
- Now let us assume that ε is used.

Construction (general case)

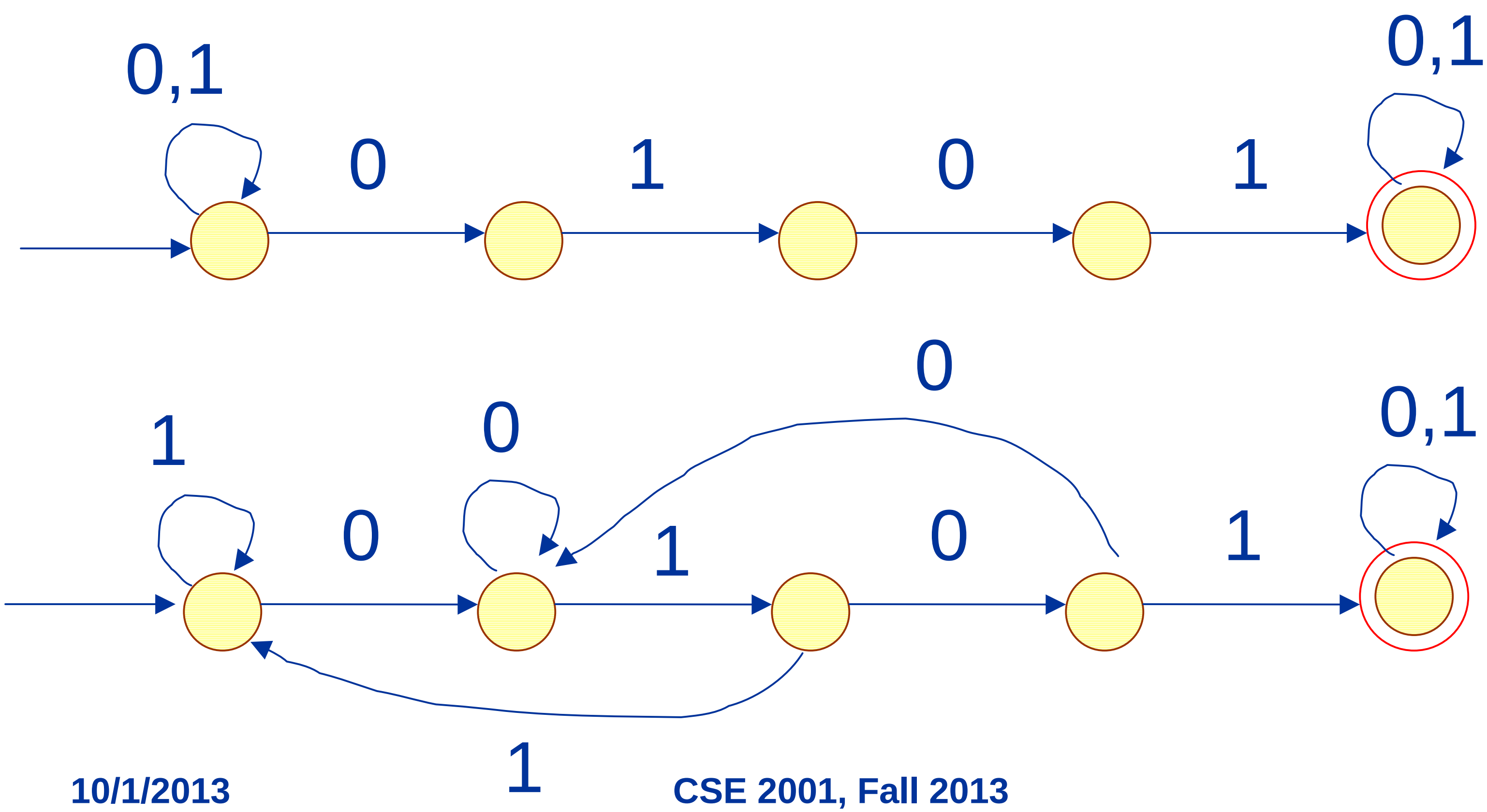
1. $Q' = \mathcal{P}(Q)$
2. $q'_0 = E(\{q_0\})$
3. for all $R \in Q'$ and $a \in \Sigma$
   $\delta'(R, a) = \{q \in Q | q \in E(\delta(r,a))$ for some $r \in R\}$
4. $F' = \{ R \in Q' |$ R contains an accept state of N$\}$

Why the construction works

- for any string w $\in \Sigma^*$,
- w is accepted by N iff w is accepted by M
- Can prove using induction on the number of steps of computation…

State minimization

It may be possible to design DFA's without the exponential blowup in the number of states. Consider the NFA and DFA below.



NFA to DFA conversion

- Completely mechanical (can be programmed easily)
- Can design a NFA for a given problem and convert it using a programme.

Characterizing FA languages

- Regular expressions

Regular Expressions (Def. 1.52)

Given an alphabet Σ, R is a regular expression if:
(INDUCTIVE DEFINITION)

- $R = a$, with $a \in \Sigma$
- $R = \varepsilon$
- $R = \emptyset$
- $R = (R_1 \cup R_2)$, with R_1 and R_2 regular expressions
- $R = (R_1 \bullet R_2)$, with R_1 and R_2 regular expressions
- $R = (R_1{}^*)$, with R_1 a regular expression

Precedence order: $*$, $\bullet$, $\cup$

Regular Expressions

- Unix ‘grep’ command: Global Regular Expression and Print
- Lexical Analyzer Generators (part of compilers)
- Both use regular expression to DFA conversion

Examples

- $e_1 = a \cup b$, $L(e_1) = \{a,b\}$
- $e_2 = ab \cup ba$, $L(e_2) = \{ab,ba\}$
- $e_3 = a^*$, $L(e_3) = \{a\}^*$
- $e_4 = (a \cup b)^*$, $L(e_4) = \{a,b\}^*$
- $e_5 = (e_m . e_n)$, $L(e_5) = L(e_m) \bullet L(e_n)$
- $e_6 = a^*b \cup a^*bb$,

  $L(e_6) = \{w |\ w \in \{a,b\}^*$ and w has 0 or more a's followed by 1 or 2 b's$\}$

Characterizing Regular Expressions

- We prove that Regular expressions (RE) and Regular Languages are the same set, i.e.,

$$RE = RL$$

Thm 1.54: RL ~ RE

We need to prove both ways:

• If a language is described by a regular expression, then it is regular (Lemma 1.55)
(We will show we can convert a regular expression R into an NFA M such that L(R)=L(M))

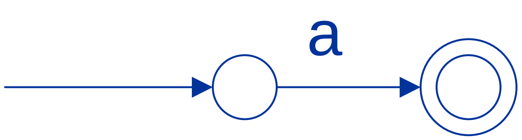
• The second part:
If a language is regular, then it can be described by a regular expression (Lemma 1.60)

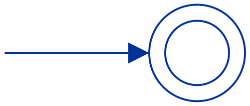
Regular expression to NFA

Claim: If L = L(e) for some RE *e*, then L = L(M) for some NFA M

Construction: Use inductive definition

1. R = a, with $a \in \Sigma$
2. R = ε
3. R = $\emptyset$
4. R = $(R_1 \cup R_2)$, with R_1 and R_2 regular expressions
5. R = $(R_1 \bullet R_2)$, with R_1 and R_2 regular expressions
6. R = $(R_1{}^*)$, with R_1 a regular expression

1. 

2. 

3. 

4,5,6: similar to closure of RL under regular operations.

Examples of RE to NFA conv.

L = {ab,ba}

L = {ab,abab,ababab,……}

L = {w | w = $a^m b^n$, $m<10$, $n>10$}