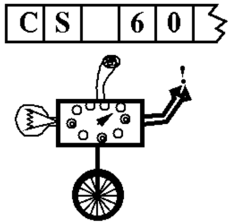


Regular Languages

November 30, 2001

Give Regular Expressions (over alphabet {0, 1}) for

- The set of strings with at most two 0's
- The set of strings with more than two 0's
- The set of strings in which 0's and 1's strictly alternate

Kleene's Remarkable Result

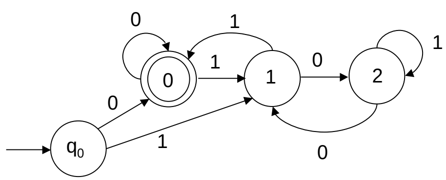
- The languages accepted by finite-state acceptors and the languages denoted by regular expressions are the same thing!

In other words:

- Part I: The language accepted by any finite-state acceptor can be expressed as a regular expression.
- Part II: For every regular expression, there is a finite state acceptor that accepts the language denoted by the expression.

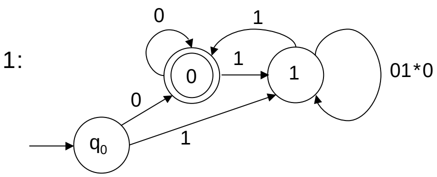
Idea of Part I:
(analogous to Gaussian elimination)

- Given a finite-state acceptor, how to derive a regular expression?
  - Eliminate states, replacing paths with regular expressions that represent those paths.
- Example:

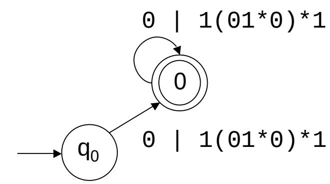


Idea of Part I (2)

- Replacement 1:



- Replacement 2:



- Final: `(0 | 1(01*0)*1) (0 | 1(01*0)*1)*`

Non-Deterministic Finite Automata

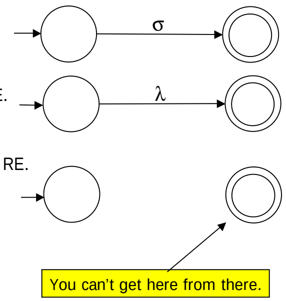
- The easiest way to prove part II is to appeal to the idea of a non-deterministic finite-state acceptor (NFA):
  - Part IIa: For every regular expression R, there is an **N**FA that accepts L(R).
  - Part IIb: For every NFA N there is a (deterministic) finite-state acceptor that accepts L(N).

Proof of Part IIa

- Part IIa: For every regular expression R, there is an NFA that accepts L(R) with exactly one start state and exactly one final (accepting) state
- This proof is by **structural induction** on the formation of regular expressions.
  - Basis:
    - Any symbol in Σ is an RE.
    - The special symbol λ is an RE.
    - The special symbol $\varnothing$ is an RE.
  - Induction step: If R and S are RE's, then so are:
    - RS
    - R | S
    - R*

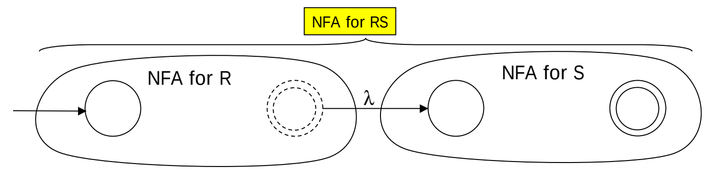
Proof of Part IIa (1)

– We construct an accepting NFA for each RE introduced in the definition.
  - Basis:
    – Any symbol in Σ is an RE.
    – The special symbol λ is an RE.
    – The special symbol $\varnothing$ is an RE.



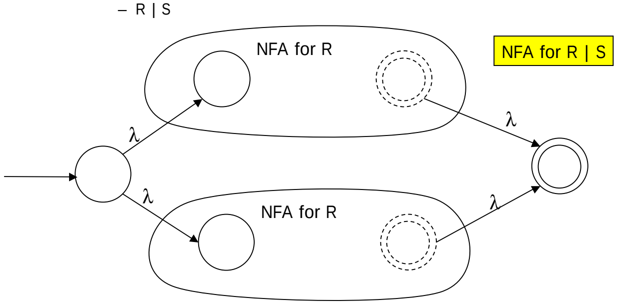
Proof of Part IIa (2)

– We construct an accepting NFA for each RE introduced in the definition.
  - Induction step: If R and S are RE's, then so are:
    – RS
    – R | S
    – R*
  - We assume that NFA's exist for R and S, and construct them for these three cases:
    – RS



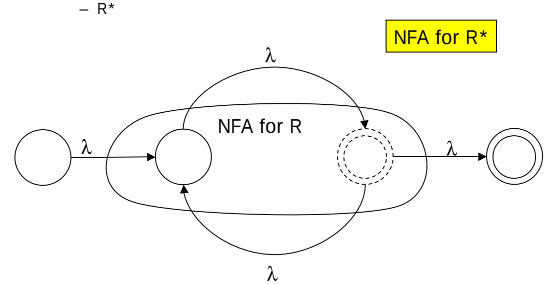
Proof of Part IIa (3)

- We assume that NFA's exist for R and S, and construct them for these three cases:
  – R | S



Proof of Part IIa (4)

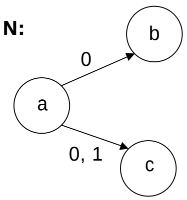
- We assume that NFA's exist for R and S, and construct them for these three cases:
  – R*



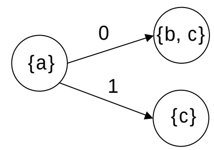
Proof of Part IIb (1)

- For every NFA N there is a (deterministic) FSA that accepts L(N).
- The idea is that for an NFA N we can construct a FSA D accepting L(N) by "simulating in parallel" all the **choices** the NFA could make. An input sequence is accepted iff **any** of those choices led to acceptance in N.

In N:



In D:

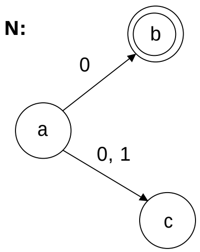


Definition of f, the state transition for D:
$f(S, \sigma) = \{q' \mid (\exists q \in S)\ q \rightarrow q' \text{ in } N\}$

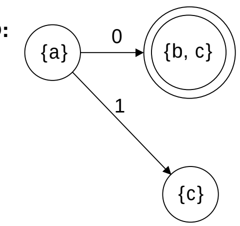
Proof of Part IIb (3)

- An **accepting** state in D is any that has an accepting state of N as a member.

In N:



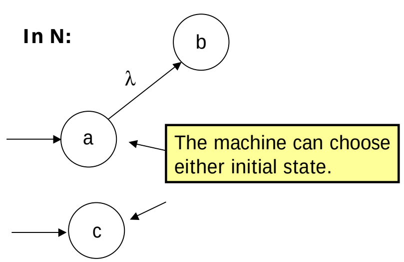
In D:



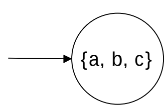
Proof of Part IIb (4)

- The **initial** state in D is the set of all states reachable from ***some*** initial state in N by the empty sequence (i. e. including λ transitions)

In N:

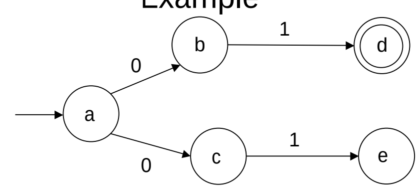


In D:



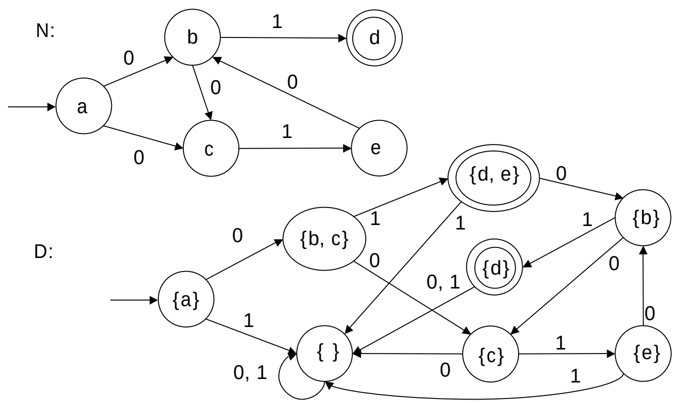
The Complete Construction for a Simple Example

N:



D:

A More Complex Example with a Loop

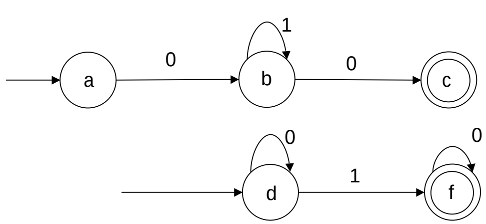


This Completes the Proof of Kleene's Theorem

- We now know that the following are equivalent:
  - L is a language denoted by some regular expression.
  - L is a language accepted by an NFA.
  - L is a language accepted by an DFA.

Example: Regular Expression to FSA (1)

- Construct an FSA for the RE
  01*0 | 0*10*
- By inspection we can do NFA's for 01*0 and 0*10*:



Example (continued)

Regular Expressions in Everday Practice:
e.g. Unix **egrep**
used for searching for ***lines containing*** matching strings in files

- Do **`man regexp`** to get this information on turing:
  - Most single characters match themselves (exceptions: **`. * [ ] \ ^ $`**)
  - . matches any character, except new-line
  - ^ matches beginning of line (must occur first)
  - $ matches end of line (must occur last)
- Examples:
  - **`egrep 'elle'`** *filename*
  - **`egrep 'll.*ll'`** *filename*   .* is like Σ^*
  - **`egrep 'll$'`** *filename*
  - **`egrep '^Ll'`** *filename*
  - **`egrep 'aa|bb|cc'`** *filename*
  - **`egrep '(aa|bb)c'`** *filename*

Regular Expressions in Everday Practice:
Unix shells

- Unix shells use a form of regular expression, but with somewhat different syntax and some restrictions
  - e.g., `*` instead of `.*`

```
> ls
car  cat  cool  coot  cot  cut
> ls c*t
cat  coot  cot  cut
> ls c{a,oo}*
car  cat  cool  coot
> ls *t
cat  coot  cot  cut
> ls c[au]t
cat  cut
> ls c[a-t]t
cat  cot
> ls c[^a-t]t
cut
```