Ling 5801: Lecture Notes 2
From FSAs to Regular Expressions

It's often useful to search strings for patterns (e.g. to find all sentences containing two commas).

Regular Expressions, based on FSAs, provide a nice shorthand for such patterns.

Contents

2.1 Regular expression syntax

A Regular Expression (RE) ρ is a string made up of:

observation symbols: x	e.g.: `a`	language: { a }
concatenations of REs: $\rho'\rho''$	e.g.: `ab`	language: { ab }
disjunctions of REs: $(\rho' \mid \rho'')$	e.g.: `(ab|b)`	language: { ab, b }
'Kleene star' repetitions of RE: $(\rho')^*$	e.g.: `(ab)*`	language: { ϵ, ab, abab, ababab, ... }

(Epsilon ϵ means the **empty string**: literally, no characters.)

For example:

```
the (dog|cat|rat) (that (chased|ate|nibbled) the (cat|rat|malt))*
```

recognizes the following sentences:

{the rat,
the rat that nibbled the malt,
the cat that ate the rat that nibbled the malt,
the dog that chased the cat that ate the rat that nibbled the malt that ate the rat that chased the cat,
. . . }

REs are often augmented with the following (equiv. to combinations of concat, disjn, star):

wildcard symbols: . = $(x\|x'\|x''\|...)$	e.g.: `.`	lang: { a, b, c, d, ... }
symbol disjunctions: $[xx'x'']$ = $(x\|x'\|x'')$	e.g.: `[ace]`	lang: { a, c, e }
symbol ranges: $[x\text{-}x']$ = $(x\|...\|x')$	e.g.: `[a-e]`	lang: { a, b, c, d, e }
one or more repetitions of REs $(\rho')^+ = \rho'(\rho')^*$	e.g.: `(ab)+`	lang: { ab, abab, ababab, ... }
zero or one repetitions of REs $(\rho')^? = (\rho'\|\epsilon)$	e.g.: `(ab)?`	lang: { ϵ, ab }

Most RE implementations assume `.*`ρ`.*`, let *anchors* '`^`' and '`$`' match beginning/end of line.

2.2 Reduction of REs to FSAs

We can build an FSA $FSA(\rho)$ that accepts the same language as any RE ρ.

In other words: $\forall \rho \,.\, L(FSA(\rho)) = L(\rho)$.

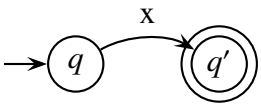
In other words: $\mathcal{L}(RE) \subseteq \mathcal{L}(FSA)$.

Base case – for observations in RE:

- observation symbols x:

  $FSA(x) = \langle\ \{q, q'\},\ \{x\},\ \{q\},\ \{q'\},\ \{\langle q, x, q'\rangle\}\ \rangle$

  graphically:

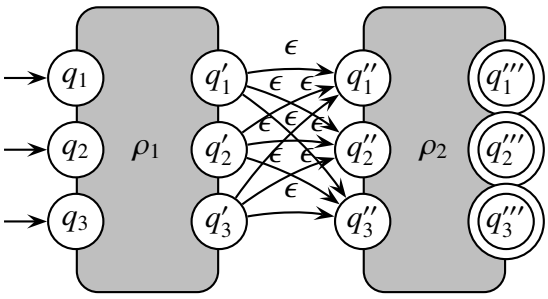


Inductive step – combine REs using 'ϵ-transitions' w/o associated obs, then compile out (assume state sets of sub-expressions $Q_{FSA(\rho_1)}$ and $Q_{FSA(\rho_2)}$ are disjoint):

- concatenations of REs ρ_1, ρ_2:

  $$FSA(\rho_1\rho_2) = \langle\ Q_{FSA(\rho_1)} \cup Q_{FSA(\rho_2)},\ X_{FSA(\rho_1)} \cup X_{FSA(\rho_2)},\ S_{FSA(\rho_1)},\ F_{FSA(\rho_2)},\ M_{FSA(\rho_1)} \cup M_{FSA(\rho_2)} \cup \{\langle q', \epsilon, q''\rangle \mid q' \in F_{FSA(\rho_1)}, q'' \in S_{FSA(\rho_2)}\}\ \rangle$$

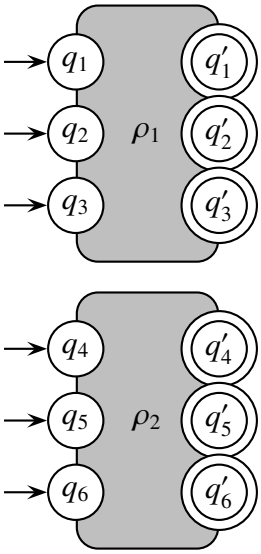
  graphically:



- disjunctions of REs ρ_1, ρ_2:

  $FSA(\rho_1 \mid \rho_2) = \langle\ Q_{FSA(\rho_1)} \cup Q_{FSA(\rho_2)},$
  $X_{FSA(\rho_1)} \cup X_{FSA(\rho_2)},$
  $S_{FSA(\rho_1)} \cup S_{FSA(\rho_2)},$
  $F_{FSA(\rho_1)} \cup F_{FSA(\rho_2)},$
  $M_{FSA(\rho_1)} \cup M_{FSA(\rho_2)}\ \rangle$

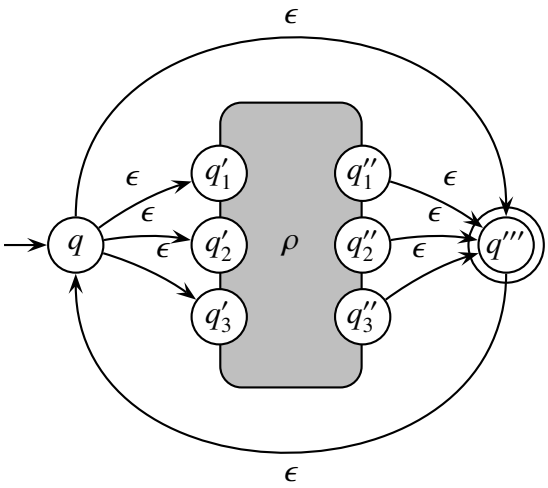
  graphically:



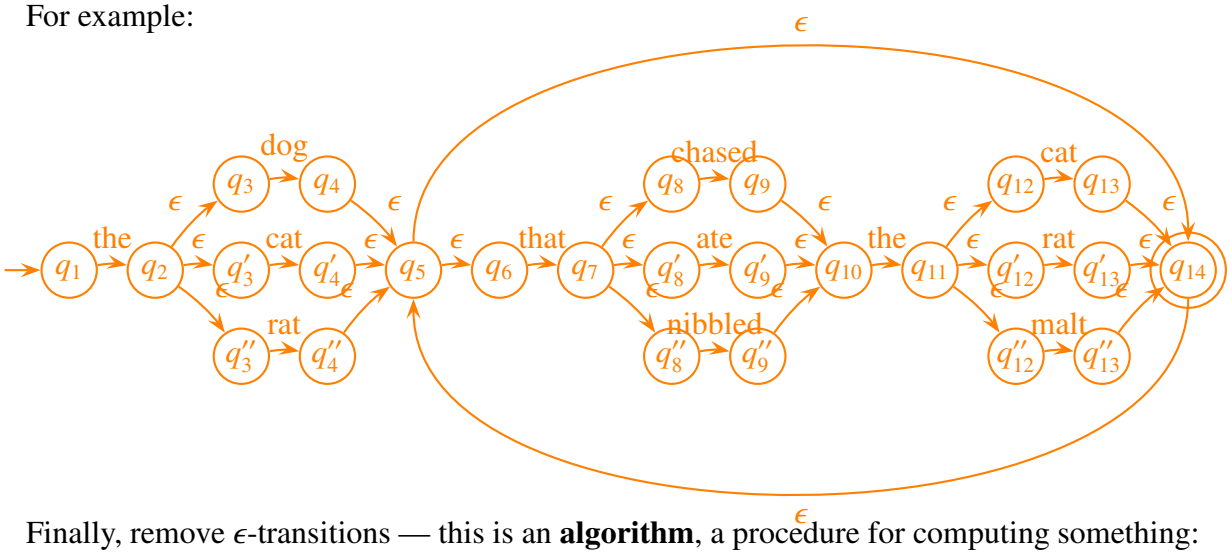
- Kleene star repetitions of RE ρ:

  $FSA(\rho^*) = \langle\ Q_{FSA(\rho)} \cup \{q, q'''\},$
  $X_{FSA(\rho)},$
  $\{q\},$
  $\{q'''\},$
  $M_{FSA(\rho)} \cup \{\langle q, \epsilon, q'\rangle \mid q' \in S_{FSA(\rho)}\} \cup \{\langle q'', \epsilon, q'''\rangle \mid q'' \in F_{FSA(\rho)}\}$
  $\cup\ \{\langle q, \epsilon, q'''\rangle, \langle q''', \epsilon, q\rangle\}\ \rangle$

  graphically:



For example:



Finally, remove ϵ-transitions — this is an **algorithm**, a procedure for computing something:

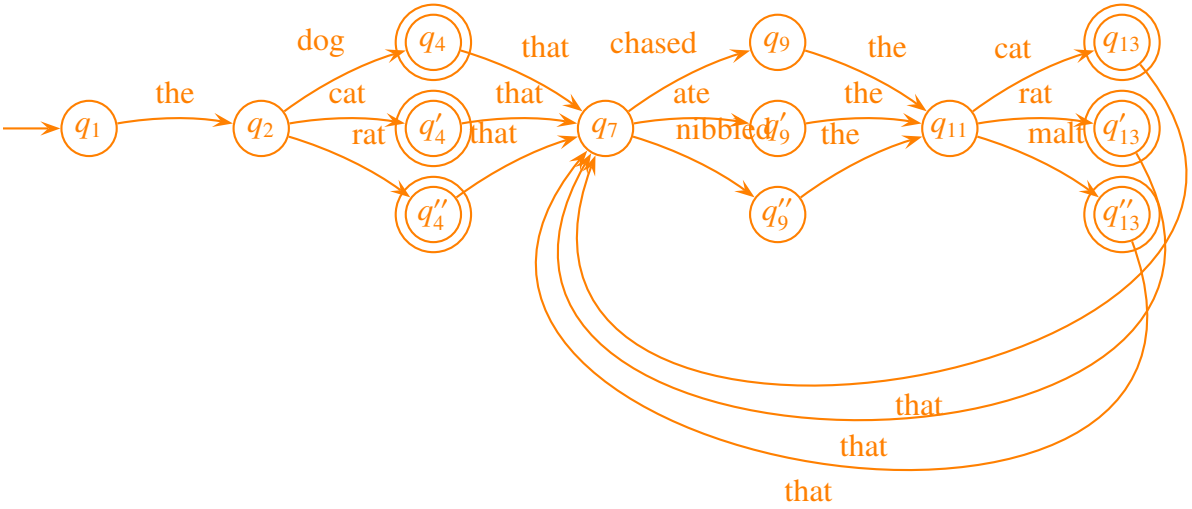
1. ϵ closure – add shortcuts for progressively longer chains of ϵ-transitions:

   $M_A^0 = M_A$
   for each chain length k from 1 to $|Q|$:
   $M_A^k = M_A^{k-1} \cup \{\langle q, \epsilon, q'' \rangle \mid \langle q, \epsilon, q' \rangle \in M_A^{k-1},\ \langle q', \epsilon, q'' \rangle \in M_A\}$

2. merge ϵ-transitions with labeled transitions, start/final states to get new automaton A':

   $A' = \langle\ Q_A,$
   $X_A,$
   $S_A\ \cup\ \{q' \mid \exists q.\ q \in S_A,\ \langle q, \epsilon, q' \rangle \in M_A^{|Q|}\},$
   $F_A\ \cup\ \{q \mid \exists q'.\ \langle q, \epsilon, q' \rangle \in M_A^{|Q|},\ q' \in F_A\},$
   $\{\langle q, x, q' \rangle \mid \langle q, x, q' \rangle \in M_A,\ x \in X_A\}\ \cup\ \{\langle q, x, q'' \rangle \mid \langle q, \epsilon, q' \rangle \in M_A^{|Q|},\ \langle q', x, q'' \rangle \in M_A\}\ \rangle$

For example (ignoring unconnected states):



Practice:

Write a regular expression to recognize the infinite language containing the following (treat each word as a single symbol):

hello ok bye
hello ok ok bye
hello ok ok ok bye
hello ok ok ok ok bye
⋮

2.3 Closure properties

FSAs also closed under the following operations (so REs could support them):

- reversal of RE ρ: (change direction of all arrows)

$$FSA(\rho^R) = \langle\ Q_{FSA(\rho)},\ X_{FSA(\rho)},\ F_{FSA(\rho)},\ S_{FSA(\rho)},\ \{\langle q', x, q\rangle \mid \langle q, x, q'\rangle \in M_{FSA(\rho)}\}\ \rangle$$

- negation of RE ρ: (swap final and non-final states)

$$FSA(\neg\rho) = \langle\ Q_{FSA(\rho)},\ X_{FSA(\rho)},\ S_{FSA(\rho)},\ Q_{FSA(\rho)} - F_{FSA(\rho)},\ M_{FSA(\rho)}\ \rangle$$

- conjunction of REs ρ_1, ρ_2: (use pairs of sub-expression states)

  $$\begin{aligned} FSA(\rho_1 \wedge \rho_2) = \langle\ & Q_{FSA(\rho_1)} \times Q_{FSA(\rho_2)}, \\ & X_{FSA(\rho_1)} \cap X_{FSA(\rho_2)}, \\ & \{\langle q, q' \rangle \mid q \in S_{FSA(\rho_1)}, q' \in S_{FSA(\rho_2)}\}, \\ & \{\langle q, q' \rangle \mid q \in F_{FSA(\rho_1)}, q' \in F_{FSA(\rho_2)}\}, \\ & \{\langle \langle q, q'' \rangle, x, \langle q', q''' \rangle \rangle \mid \langle q, x, q' \rangle \in M_{FSA(\rho_1)}, \langle q'', x, q''' \rangle \in M_{FSA(\rho_2)}\}\ \rangle \end{aligned}$$

- exclusion of REs ρ_1, ρ_2: (combine negation and conjunction)

  $$FSA(\rho_1 - \rho_2) = FSA(\rho_1 \wedge \neg\rho_2)$$

2.4 Limits of FSAs / REs

FSAs (and therefore REs) can only recognize sequences with finitely-bounded memory

Pumping lemma:

If L is an infinite regular language (in $\mathcal{L}(FSA)$), then $\exists x, y, z$ such that $y \neq \epsilon$ and $xy^n z \in L$ for all $n \geq 0$ (where y^n means n repetitions of string y).

Example: $a^n b^n$: $\{\epsilon, ab, aabb, aaabbb, \ldots\}$ is not regular.

Why not?

Because, in order to allow infinite languages with finites states, y must occur either...

- within the a's, generating strings like *aaaabbb* when pumped, or
- within the b's, generating strings like *aaabbbb* when pumped, or
- within the crossover from a's to b's, generating strings like *aababbb* when pumped

none of which are in $a^n b^n$.

NOTE: the same problem comes up in trying to recognize nested parentheses!

2.5 Cognitive plausibility of FSAs

Are we FSAs?

- Problem for FSAs – we seem to learn general syntactic patterns w. unbounded nesting:

  '[$_{NP}$ [$_{NP}$ the photos] [$_{NP}$ the reporter] [$_{V}$ took]] were good' (NP → NP NP V)

  When center NP is expanded, this generates non-regular language $NP^n\ NP\ V^n$:

  e.g. '[$_{NP}$ the photos] [$_{NP}$ [$_{NP}$ the reporter] [$_{NP}$ I] [$_{V}$ hired]] [$_{V}$ took] were good'

- But *in practice* – we can't keep track of more than 4 or so disconnected ideas:

  'the malt the rat the cat the dog the man I know bought bit chased ate was rancid'

This is called a 'competence / performance' distinction: we are FSAs emulating non-FSAs.