Ling 5801: Lecture Notes 2
From FSAs to Regular Expressions

1. Pattern matching with regular expressions

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

   Regular Expressions provide a nice shorthand for such patterns
   all regular expressions can be recognized by FSAs

2. Regular expression syntax

   a Regular Expression (RE) \( \rho \) 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: \( (\rho' | \rho'') \)  
   e.g.: \( (ab | b) \)  
   language: \( \{ ab, b \} \)
   
   ‘Kleene star’ repetitions of RE: \( (\rho')^* \)  
   e.g.: \( (ab)^* \)  
   language: \( \{ \epsilon, ab, abab, ababab, ... \} \)
   (epsilon \( \epsilon \) means the empty string)

   For example:

   \( \text{the (dog|cat|rat) (that (chased|ate|nibbled) the (cat|rat|malt))}^* \)

   recognizes the following sentences:

   \( \{ \text{the rat,} \) \)
   \( \text{the rat that nibbled the malt,} \)
   \( \text{the cat that ate the rat that nibbled the malt,} \)
   \( \text{the dog that chased the cat that ate the rat that nibbled the malt that ate the rat that chased the cat,} \)
   \( \ldots \) \}

   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: \( [x-x'] = (x|...|x') \)  
   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: \( \{ \epsilon, ab \} \)

   Most RE implementations assume \( .^* \), \( \rho^* \), let anchors ‘\(^\cdot\)' and ‘\(^$\)’ match beginning/end of line

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

   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 $x$:

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

graphically:

inductive step – combine REs using ‘$\epsilon$-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 $\rho_1, \rho_2$:
  $$FSA(\rho_1 \rho_2) = \langle Q_{FSA(\rho_1)} \cup Q_{FSA(\rho_2)}, X_{FSA(\rho_1)} \cup X_{FSA(\rho_2)}, S_{FSA(\rho_1)}, F_{FSA(\rho_2)}, M_{FSA(\rho_1)} \cup M_{FSA(\rho_2)} \cup \{(q', \epsilon, q'') \mid q' \in F_{FSA(\rho_1)}, q'' \in S_{FSA(\rho_2)}\} \rangle$$

  graphically:

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

2
• Kleene star repetitions of RE $\rho$:

$$FSA(\rho^*) = \langle Q_{FSA(\rho)} \cup \{q, q''\},$$

$$X_{FSA(\rho)},$$

$$\{q\},$$

$$\{q''\},$$

$$M_{FSA(\rho)} \cup \{(q, \epsilon, q') \mid q' \in S_{FSA(\rho)}\} \cup \{(q'', \epsilon, q'') \mid q'' \in F_{FSA(\rho)}\}$$

$$\cup \{(q, \epsilon, q''), (q'', \epsilon, q)\}\rangle$$
for example:

finally, remove \( \epsilon \)-transitions — this is an \emph{algorithm}, a procedure for computing something:

(a) \( \epsilon \) closure – add shortcuts for progressively longer chains of \( \epsilon \)-transitions:

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

(b) merge \( \epsilon \)-transitions with labeled transitions, start/final states to get new automaton \( A' \):

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

for example (ignoring unconnected states):
4. **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

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

- **reversal of RE** \( \rho \): (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** \( \rho \): (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** \( \rho_1, \rho_2 \): (use pairs of sub-expression states)

\[
FSA(\rho_1 \land \rho_2) = \langle Q_{FSA(\rho_1)} \times Q_{FSA(\rho_2)},
X_{FSA(\rho_1)} \cap X_{FSA(\rho_2)},
\{\langle q, q' \rangle \mid q \in S_{FSA(\rho_1)}, q' \in S_{FSA(\rho_2)} \},
\{\langle q, q' \rangle \mid q \in F_{FSA(\rho_1)}, q' \in F_{FSA(\rho_2)} \},
\{\{\langle q'', x, q''' \rangle \mid \langle q, x, q' \rangle \in M_{FSA(\rho_1)}, \langle q'', x, q''' \rangle \in M_{FSA(\rho_2)} \} \rangle
\]

- **exclusion of REs** \( \rho_1, \rho_2 \): (combine negation and conjunction)

\[
FSA(\rho_1 - \rho_2) = FSA(\rho_1 \land \neg \rho_2)
\]

6. **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 \( y \))
Exception: $a^n b^n$: $\{\epsilon, ab, aabb, aaabbb, \ldots\}$ 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 $aaababbb$ when pumped

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

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

7. Cognitive plausibility of 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