LING/C SC/PSYC 438/538
Computational Linguistics

Sandiway Fong
Lecture 13: 10/9
Administrivia

• Homework 2
  – grade (to be) sent by email today

• Upcoming midterm
  – this Thursday
NDFSA → (D)FSA: Recap

- **set of states construction**
  - new machine simulates in parallel all possible situations for the original machine

- **procedure must terminate**
  - 4 states
  - $2^4 - 1 = 15$ possible different states in the new machine
Practice Question

• convert the NDFSA into a deterministic FSA
• implement both the NDFSA and the equivalent FSA in Prolog using the “one predicate per state” encoding
• run both machines on the strings \textit{abab} and \textit{abaaba},
  – how many steps (transitions + final stop) does each machine make?

\[ \text{from figure 2.27 in the textbook} \]
Today’s Topics

• FSA to regexp
• FSA and complementation
• Regexp and complementation
FSA to Regexp

• Give a Perl regexp for this FSA
FSA to Regexp

• Let’s define $E_i$ to be the possible strings that can lead to an accepting computation from state $i$

• Then:
  - $E_1 = aE_2$
  - $E_2 = bE_3 | bE_4$
  - $E_3 = aE_2 | \lambda$
  - $E_4 = aE_3$
FSA to Regexp

- Solve simultaneous equations
  - $E_1 = aE_2$
  - $E_2 = bE_3 \mid bE_4$
  - $E_3 = aE_2 \mid \lambda$
  - $E_4 = aE_3$
FSA to Regexp

• Solve simultaneous equations
  – $E_1 = aE_2$
  – $E_2 = bE_3 \mid bE_4$ (eliminate)
  – $E_3 = aE_2 \mid \lambda$
  – $E_4 = aE_3$
• Then
  – $E_1 = abE_3 \mid abE_4$
  – $E_3 = abE_3 \mid abE_4 \mid \lambda$
FSA to Regexp

- Solve reduced equations
  - $E_1 = abE_3 \mid abE_4$
  - $E_3 = abE_3 \mid abE_4 \mid \lambda$
  - $E_4 = aE_3$
FSA to Regexp

• Solve reduced equations
  – \( E_1 = abE_3 \mid abE_4 \)
  – \( E_3 = abE_3 \mid abE_4 \mid \lambda \)
  – \( E_4 = aE_3 \) (eliminate \( E_4 \))
• Then
  – \( E_1 = abE_3 \mid abaE_3 \)
  – \( E_3 = abE_3 \mid abaE_3 \mid \lambda \)
FSA to Regexp

• Solve reduced equations
  – \( E_1 = abE_3 \mid abaE_3 \)
  – \( E_3 = abE_3 \mid abaE_3 \mid \lambda \)
• Solve recursive definition for \( E_3 \)
  rewrite \( E_3 \) as
  – \( E_3 = (ab|aba)E_3 \mid \lambda \)
FSA to Regexp

• Then
  – $E_1 = abE_3 \mid abaE_3$
  – $E_3 = (ab\mid aba)^*$

• Also
  – $E_1 = (ab\mid aba)E_3$

• Substituting for $E_3$
  – $E_1 = (ab\mid aba)(ab\mid aba)^*$
FSA to Regexp

- $E_1 = (ab|aba)(ab|aba)^*$

  • We know
    - $e^+ = ee^* = e^*e$

  • Hence
    - $E_1 = (ab|aba)^+$

Note: the order of variable elimination matters
FSA and Complementation

- FSA are closed under complementation
  - i.e. make a new FSA’ accept strings rejected by FSA, and reject strings accepted by FSA over the alphabet
  - $\Sigma = \{a, b\}$
FSA and Complementation

• A Simple Idea:
  – make old accepting state(s) non-final states
  – make non-final and reject states accepting states

• Nearly correct, but why does this not work?
  – hint: consider $aba$
FSA and Complementation

- **Prolog**
  1. `one([a|L]) :- two(L).`
  2. `two([b|L]) :- three(L).`
  3. `two([b|L]) :- four(L).`
  4. `three([]).`
  5. `three([a|L]) :- two(L).`
  6. `four([a|L]) :- three(L).`

  weakness is that \+ can’t be used to generate

- To construct the complement FSA in Prolog:
  - *add a single line*
  7. `fsab(L) :- \+ one(L).`

- \+ is the Prolog operator for negation as failure to prove
  - i.e.
  - \+ one(L) is true if one(L) cannot be true
  - \+ one(L) is false if one(L) can be true
Regexp and Complementation

Not part of basic regexps

• **Formally, we can define:**
  – complement of regexp $e = \Sigma^* - e$
  – where
    • $\Sigma^* = $ set of all possible strings over alphabet $\Sigma$
    • i.e. zero or more occurrences ...

• **Class Exercise**
  – Let $\Sigma = \{a,b\}$
  – Give a Perl regexp equivalent to the complement regexp for $(ab|aba)^+$