CSE 311 Foundations of Computing I

Lecture 27

FSM Limits, Pattern Matching

Autumn 2012

Announcements

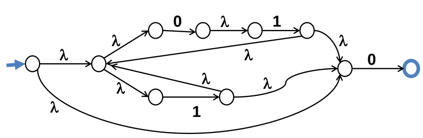
- Reading assignments
  - 7th Edition, Section 13.4
  - 6th Edition, Section 12.4
  - 5th Edition, Section 11.4
- Next week
  - 7th edition: 2.5 (Cardinality) + p. 201 and 13.5
  - 6th edition: pp. 158-160 (Cardinality)+ p 177 and 12.5
  - 5th edition: Pages 233-236 (Cardinality) and 11.5

- Topic list and sample final exam problems have been posted
- Comprehensive final, roughly 67% of material post midterm
- Review session, Saturday, December 8, 10 am – noon (tentatively)
- Final exam, Monday, December 10
  - 2:30-4:20 pm or 4:30-6:20 pm, Kane 220.
  - If you have a conflict, contact instructors ASAP

Last lecture highlights

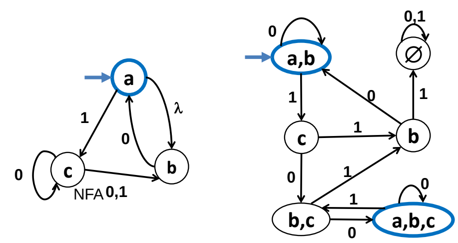
- NFAs from Regular Expressions

(01 ∪1)*0



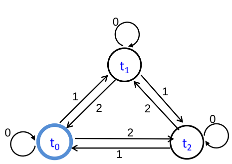
Last lecture highlights

- "Subset construction": NFA to DFA



Converting an NFA to a regular expression

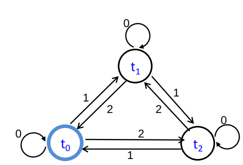
- Consider the DFA for the mod 3 sum
  - Accept strings from {0,1,2}* where the digits mod 3 sum of the digits is 0



Splicing out a node

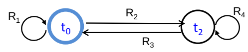
- Label edges with regular expressions

$t_0 \to t_1 \to t_0$: 10*2
$t_0 \to t_1 \to t_2$: 10*1
$t_2 \to t_1 \to t_0$: 20*2
$t_2 \to t_1 \to t_2$: 20*1



Finite Automaton without t_1

R_1: 0 | 10*2
R_2: 2 | 10*1
R_3: 1 | 20*2
R_4: 0 | 20*1



R_5: R_1 | R_2R_4*R_3

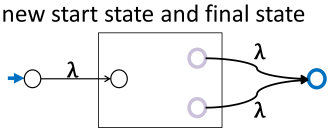
Final regular expression:
(0 | 10*2 | (2 | 10*1)(0 | 20*1)*(1 |20*2))*

Generalized NFAs

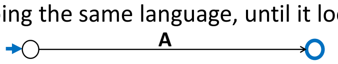
- Like NFAs but allow
  - Parallel edges
  - Regular Expressions as edge labels
    - NFAs already have edges labeled λ or ***a***
- An edge labeled by **A** can be followed by reading a string of input chars that is in the language represented by **A**
- A string x is accepted iff there is a path from start to final state labeled by a regular expression whose language contains x

Starting from NFA

- Add new start state and final state



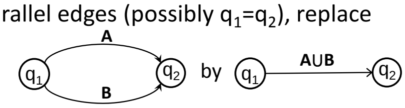
- Then eliminate original states one by one, keeping the same language, until it looks like:



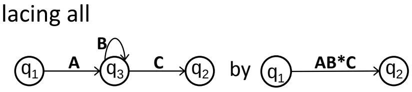
- Final regular expression will be **A**

Only two simplification rules:

- **Rule 1**: For any two states q_1 and q_2 with parallel edges (possibly q_1=q_2), replace



- **Rule 2**: Eliminate non-start/final state q_3 by replacing all



for *every* pair of states q_1, q_2 (even if q_1=q_2)

Automata Theory Summary

- Every DFA is an NFA
- Every NFA can be converted into a DFA that accepts the same language
  - However there may be an exponential increase in the number of states
- Given a regular expression, we can construct an NFA that recognizes it
- Given an NFA we can construct an regular for the strings accepted by it

What can Finite State Machines do?

- We've seen how we can get DFAs to recognize all regular languages
- What about some other languages we can generate with CFGs?
  - { 0^n1^n : n≥0 }?
  - Binary Palindromes?
  - Strings of Balanced Parentheses?

A={0^n1^n : **n≥0**} cannot be recognized by any DFA

Consider the infinite set of strings

S={λ, 0, 00, 000, 0000, ...}

Claim: No two strings in S can end at the same state of any DFA for A, so no such DFA can exist

Proof: Suppose n≠m and 0^n and 0^m end at the same state p.

Since 0^n1^n is in A, following 1^n after state p must lead to a final state.

But then the DFA would accept 0^m1^n which is a contradiction

The set B of binary palindromes cannot be recognized by any DFA

Consider the infinite set of strings

S={λ, 0, 00, 000, 0000, ...}

Claim: No two strings in S can end at the same state of any DFA for B, so no such DFA can exist

Proof: Suppose n≠m and 0^n and 0^m end at the same state p.

Since 0^n10^n is in B, following 10^n after state p must lead to a final state.

But then the DFA would accept 0^m10^n which is a contradiction

The set P of strings of balanced parentheses cannot be recognized by any DFA

The set P of strings {1^j | j = n^2} cannot be recognized by any DFA

Suppose 1^j and 1^k reach the same state p with j < k

$1^k1^{k(k-1)}$ must reach an accepting state q

$1^j1^{k(k-1)}$ must reach the same accepting state q

Thus, $j + k(k-1) = k^2 – k + j$ must be a perfect square

Is that possible?

Pattern Matching

- Given
  - a string **s** of **n** characters
  - a pattern **p** of **m** characters
  - usually **m<<n**
- Find
  - all occurrences of the pattern **p** in the string **s**
- Obvious algorithm:
  - try to see if **p** matches at each of the positions in **s**
    - stop at a failed match and try the next position

Pattern **p** = x x x x x y

String **s** = x x x x x x x x x y x x x x x x x x x x x x y x x

Pattern **p** = x y x y y x y x y x x

String **s** = x y x x y x y x y y x y x y x y y x y x y x x

String **s** = x y x x y x y x y y x y x y x y y x y x y x x
x y x y y x y x y x x

String **s** = x y x x y x y x y y x y x y x y y x y x y x x
x y x y
x y x y y x y x y x x

String **s** = x y x x y x y x y y x y x y x y y x y x y x x
x y x y
x
x y x y y x y x y x x

String **s** = x y x x y x y x y y x y x y x y y x y x y x x
x y x y
x
x y
x y x y y x y x y x x

String **s** = x y x x y x y x y y x y x y x y y x y x y x x
x y x y
x
x y
x y x y y
x y x y y x y x y x x

String **s** = x y x x y x y x y y x y x y x y y x y x y x x
x y x y
x
x y
x y x y y
x
x y x y y x y x y x x

String **s** = x y x x y x y x y y x y x y x y y x y x y x x
x y x y
x
x y
x y x y y
x
x y x y y x y x y x x
x y x y y x y x y x x

String **s** = x y x x y x y x y y x y x y x y y x y x y x x
x y x y
x
x y
x y x y y
x
x y x y y x y x y x x
x
x y x y y x y x y x x

String **s** = x y x x y x y x y y x y x y x y y x y x y x x
x y x y
x
x y
x y x y y
x
x y x y y x y x y x x
x
x y x
x y x y y x y x y x x

String **s** = x y x x y x y x y y x y x y x y y x y x y x x
x y x y
x
x y
x y x y y
x
x y x y y x y x y x x
x
x y x
x
x y x y y x y x y x x

String **s** = x y x x y x y x y y x y x y x y y x y x y x x
x y x y
x
x y
x y x y y
x
x y x y y x y x y x x
x
x y x
x
x
x y x y y x y x y x x

String **s** = x y x x y x y x y y x y x y x y y x y x y x x

x y x y
x
x y
x y x y y
x
x y x y y x y x y x x
x
x y x
x
x
x y x y y
x y x y y x y x y x x

String **s** = x y x x y x y x y y x y x y x y y x y x y x x

x y x y
x
x y
x y x y y
x
x y x y y x y x y x x
x
x y x
x
x
x y x y y
x
x y x y y x y x y x x

Worst-case time
O(**mn**)

String **s** = x y x x y x y x y y x y x y x y y x y x y x x

x y x y
x
x y
x y x y y
x
x y x y y x y x y x x
x
x y x
x
x
x y x y y
x
x y x y y x y x y x x

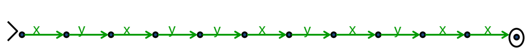
Lots of wasted work

Better Pattern Matching via Finite Automata

- Build a DFA for the pattern (preprocessing) of size O(**m**)
  - Keep track of the 'longest match currently active'
  - The DFA will have only **m**+1 states
- Run the DFA on the string **n** steps
- Obvious construction method for DFA will be O(**m²**) but can be done in O(**m**) time.
- Total O(**m+n**) time

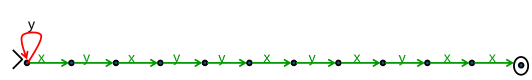
Building a DFA for the pattern

Pattern **p**=x y x y y x y x y x x



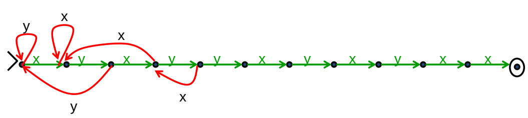
Preprocessing the pattern

Pattern **p**=x y x y y x y x y x x



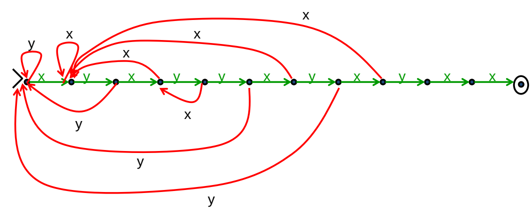
Preprocessing the pattern

Pattern **p**=x y x y y x y x y x x



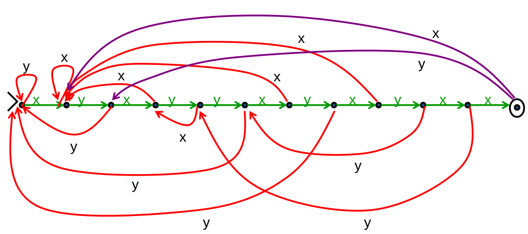
Preprocessing the pattern

Pattern **p**=x y x y y x y x y x x



Preprocessing the pattern

Pattern **p**=x y x y y x y x y x x



Generalizing

- Can search for arbitrary combinations of patterns
  - Not just a single pattern
  - Build NFA for pattern then convert to DFA 'on the fly'.
    - Compare DFA constructed above with subset construction for the obvious NFA.