CSE 311 Foundations of Computing I

Lecture 24

FSM Limits, Pattern Matching

Autumn 2011

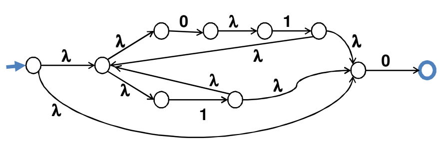
Announcements

- Reading assignments
  - 7^{th} Edition, Section 13.4
  - 6^{th} Edition, Section 12.4
  - 5^{th} Edition, Section 11.4

Last lecture highlights

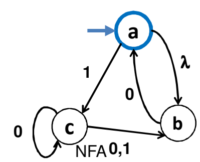
- NFAs from Regular Expressions

$(01 \cup 1)^*0$

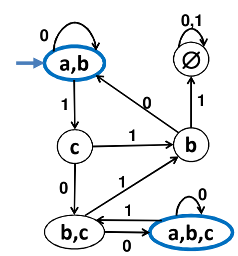


Last lecture highlights

- "Subset construction": NFA to DFA



NFA



DFA

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?
  - $\{ 0^n1^n : n \geq 0 \}$?
  - Binary Palindromes?
  - Strings of Balanced Parentheses?

$A=\{0^n1^n : n\geq 0\}$ cannot be recognized by any DFA

Consider the infinite set of strings

$S=\{\lambda, 0, 00, 000, 0000, \ldots\}$

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 $n \neq m$ 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

$S=\{\lambda, 0, 00, 000, 0000, ...\}$

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 $n \neq m$ 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

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

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

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
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

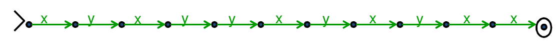
Lots of wasted work

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(**m**^{2}) but can be done in O(**m**) time.
- Total O(**m+n**) time

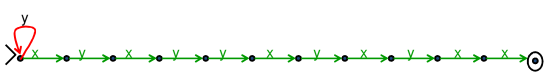
Building a DFA for the pattern

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



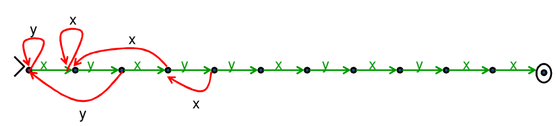
Preprocessing the pattern

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



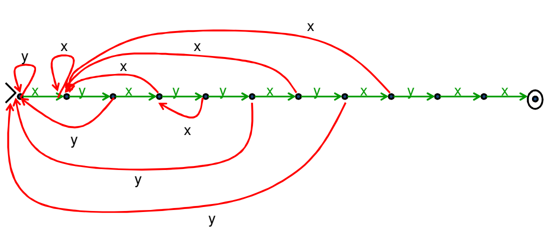
Preprocessing the pattern

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



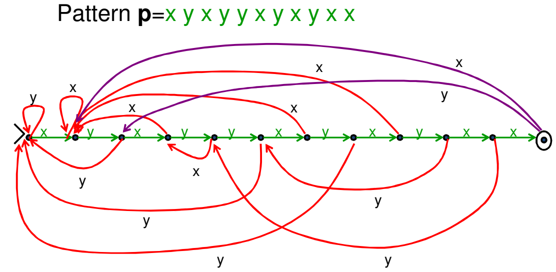
Preprocessing the pattern

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.

A Quick Note...

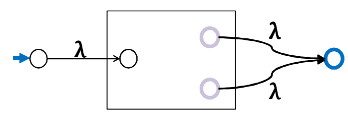
- On how to convert NFAs and DFAs to equivalent regular expressions...
- We've already seen
  - DFAs and NFAs recognize the same languages
  - NFAs (and therefore DFAs) recognize any language given by a regular expression
- This completes the equivalence of DFAs and regular expressions

Generalized NFAs

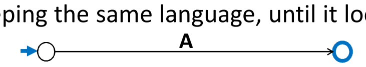
- Like NFAs but allow
  - Parallel edges
  - Regular Expressions as edge labels
    - NFAs already have edges labeled **λ** or ***a***
- An edge labeled by **A** can be followed by reading a string of input chars that is in the language represented by **A**
- A string x is accepted iff there is a path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

Starting from NFA

- Add new start state and final state



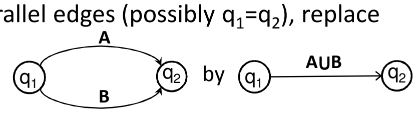
- Then eliminate original states one by one, keeping the same language, until it looks like:



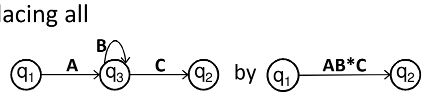
- Final regular expression will be **A**

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 $q_1=q_2$)