LING/C SC/PSYC 438/538

Lecture 18

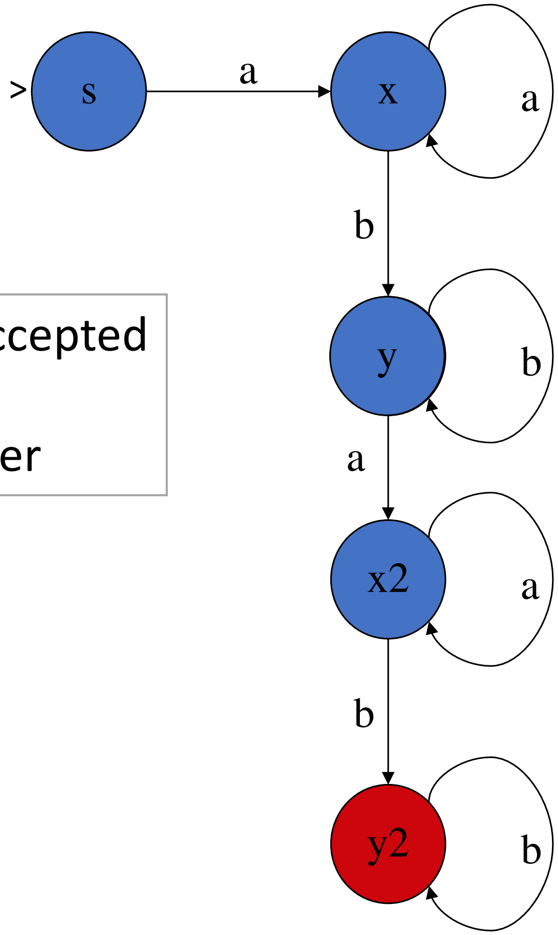
Sandiway Fong

Backreferences and FSA

- Deep question:
  - why are backreferences impossible in FSA?

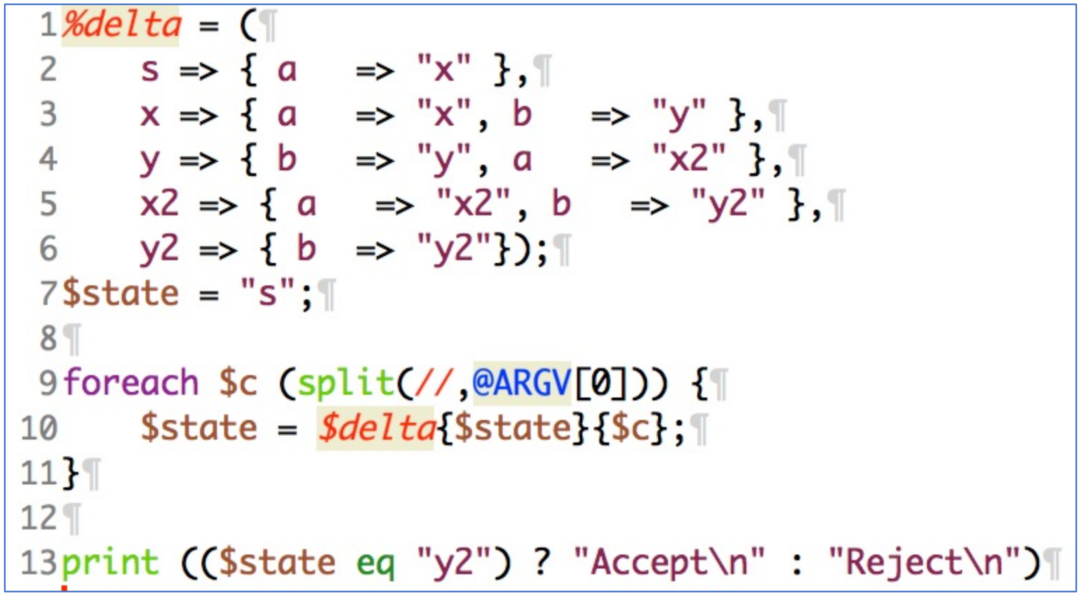
Example: Suppose you wanted a machine that accepted /(a+b+)\1/
One idea: link two copies of the machine together

Doesn’t work!
Why?



Backreferences and FSA

- `fsa2.perl`

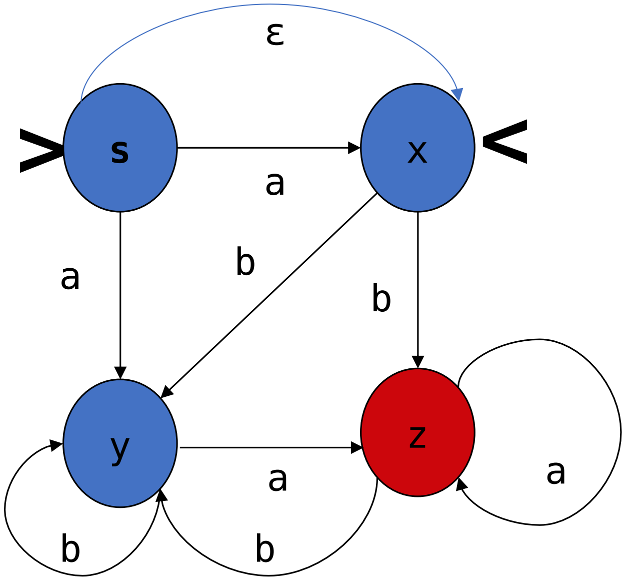


```
 1%delta = (
 2     s => { a   => "x" },
 3     x => { a   => "x", b   => "y" },
 4     y => { b   => "y", a   => "x2" },
 5     x2 => { a   => "x2", b   => "y2" },
 6     y2 => { b  => "y2"});
 7$state = "s";
 8
 9foreach $c (split(//,@ARGV[0])) {
10     $state = $delta{$state}{$c};
11}
12
13print (($state eq "y2") ? "Accept\n" : "Reject\n")
```

- Perl:
  - note line 10: next state is a function of previous state and current symbol ONLY
  - ∴ # of a's and b's in the two halves don't have to match:
- `perl fsa.perl aabba`
- `Reject`
- `perl fsa.perl aabbaaaabbbb`
- `Accept`
- `perl fsa.perl aabbaaaab`
- `Accept`

Multiple start states

- Example: simulate this by using an *e*-transition:

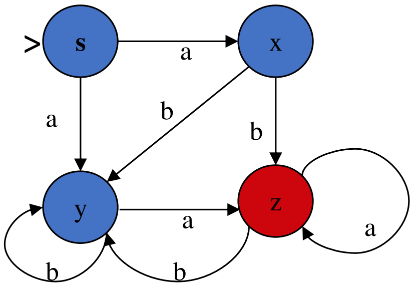


- Multiple final states vs. a single state: also same expressive power.
- Doesn't have to have any final states at all:

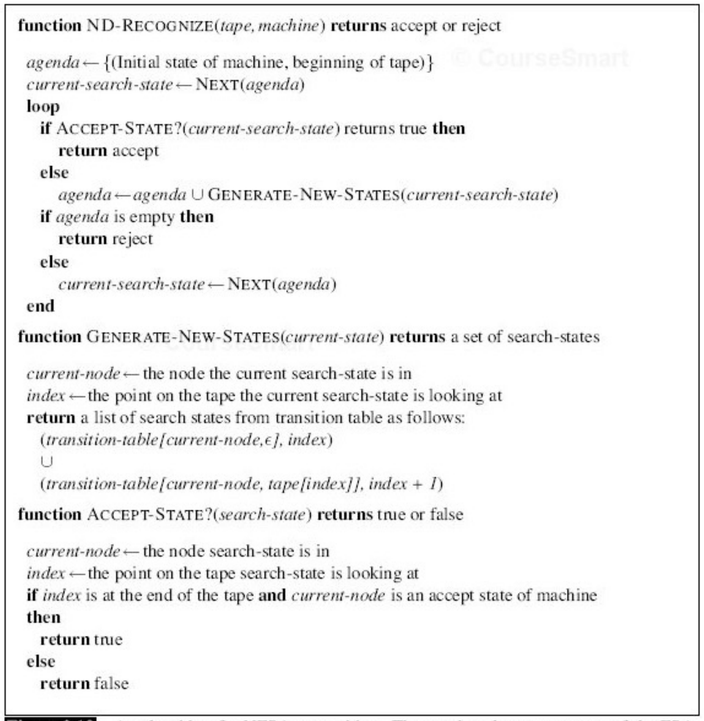
  L(*machine*) = {}
- What's the simplest possible FSA?

Non-Deterministic Finite State Automata (NDFSA)

- **non-deterministic FSA (NDFSA**)
- *no restriction on ambiguity (surprisingly, no increase in power)*
- Example:



Non-Deterministic Finite State Automata (NDFSA)



```
function ND-RECOGNIZE(tape, machine) returns accept or reject

 agenda ← {(Initial state of machine, beginning of tape)}
 current-search-state ← NEXT(agenda)
 loop
   if ACCEPT-STATE?(current-search-state) returns true then
      return accept
   else
      agenda ← agenda ∪ GENERATE-NEW-STATES(current-search-state)
   if agenda is empty then
      return reject
   else
      current-search-state ← NEXT(agenda)
 end
function GENERATE-NEW-STATES(current-state) returns a set of search-states

 current-node ← the node the current search-state is in
 index ← the point on the tape the current search-state is looking at
 return a list of search states from transition table as follows:
   (transition-table[current-node,ϵ], index)
   ∪
   (transition-table[current-node, tape[index]], index + 1)
function ACCEPT-STATE?(search-state) returns true or false

 current-node ← the node search-state is in
 index ← the point on the tape search-state is looking at
 if index is at the end of the tape and current-node is an accept state of machine
 then
   return true
 else
   return false
```

Figure 2.19 An algorithm for NFSA recognition. The word *node* means a state of the FSA, and *state* or *search-state* means “the state of the search process”, i.e., a combination of *node* and *tape position*.

Possible strategies for keeping track of multiple states:

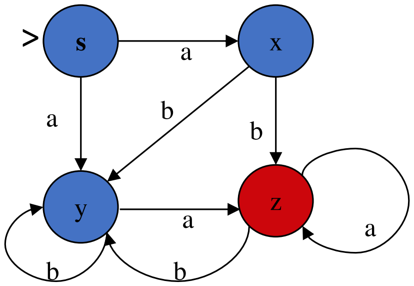
1. Backtracking (***backup***)
2. Parallelism (*split the computation*)

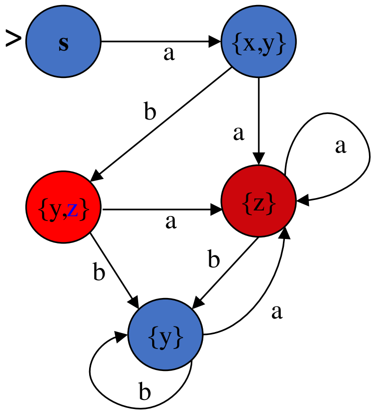
algorithm gets complicated fast

NDFSA → (D)FSA

[discussed at the end of section 2.2 in the textbook]

- construct a new machine
  - each state of the new machine represents the ***set of possible states*** of the original machine when stepping through the input
- **Note**:
  - new machine is equivalent to old one (*but has more states*)
  - new machine is deterministic
- **example**

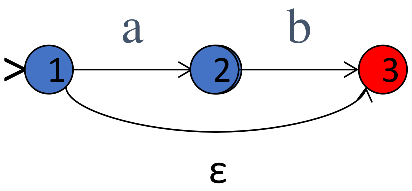


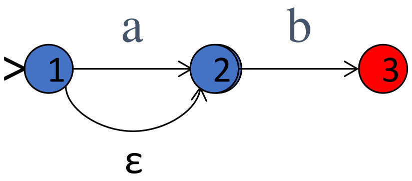


[Powerpoint Animation]

Ungraded EXERCISE

- Do the following exercise to check your understanding:
  - apply the set-of-states construction technique to the two machines on the ε−transition slide (repeated below)
  - self-check your answer:
    - verify in each case that the machine produced is **deterministic** and accurately simulates its ε−transition counterpart

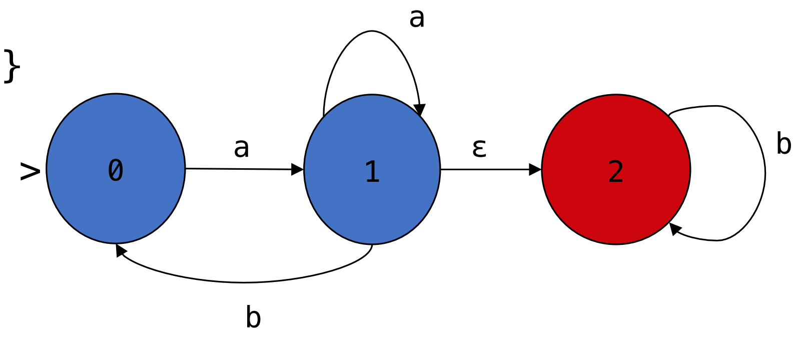




Homework 10

1. Give an equivalent Perl regex for the FSA shown below.
2. Convert the NDFSA to a (deterministic) FSA. Draw the machine.
3. Give the implementation of the FSA in Perl.
4. Run your two Perl programs and give examples:
   - your Perl regex should accept and reject (*) same strings as the Perl FSA
   - a, *b, aa, ab, *ba, aaab, abaabb, *abba, *abaabbaaabbb

Σ = {a, b}



Homework 10

- Due date:
  - Sunday midnight
  - One PDF file
  - Subject: 438/538 Homework 10 *YOUR NAME*
  - Cite sources, write your own code!