

LING/C SC/PSYC 438/538

Lecture 17

Prof. Sandiway Fong

Today's Topics

- Homework 10 Review
- Regex example: `xkcd simplewriter`
- FSA *contd.*
 - formal definition
 - Perl and Python implementations
 - e-transitions
 - single vs. multiple start states

Homework 10 Review

- Consider again the nested object relative clause construction from Homework 9

- Examples:

1. [NP the **boy** [CP who the girl saw]]
2. [NP the boy [CP who [NP the **girl** [CP who the woman found]] saw]]
3. [NP the boy [CP who [NP the girl [CP who [NP the **woman** [CP who the dog chased]] found]] saw]]

etc.

4. the boy who the girl who the woman who the **man** who the dog chased knew found saw

Homework 10 Review

- `perl -le '$_ = qq{@ARGV}; /^ ( ... ) $/'`
- *Recursion must always have a stopping condition!*
- Base case:
 - the (\w+)
 - the boy who **the girl** saw
- Recursive case:
 - the (\w+) who (?1) (\w+)
 - **the boy who** the girl **saw**
- *But the start of (?1) will be the subject!*
- *How do we access that? Use **lookahead!***
 - the (\w+) who (**?=the (\w+))**(?1) (\w+)

Homework 10 Review

- `perl -le '$_ = qq{@ARGV}; /^ ( ... ) $/'`
- the `(\w+)` *overlaps* the `(\w+)` who `(?=the (\w+)) (?1) (\w+)`
- *Can make the non-overlapping part optional!*
- the `(\w+)( who (|=the (\w+)) (?1) (\w+)) ?`
- 2 3 4 5
- *Check!*
- `perl -le '$_ = qq{@ARGV}; print "match!" if /^(the (\w+)( who (|=the (\w+)) (?1) (\w+)) ?) $/'`

Homework 10 Review

- `$ perl -le '$_ = qq{@ARGV}; print "match!" if /^(the (\w+)( who (?:the (\w+))?(?1) (\w+))?)$/'` the boy who the girl saw

match!

- `$ perl -le '$_ = qq{@ARGV}; print "match!" if /^(the (\w+)( who (?:the (\w+))?(?1) (\w+))?)$/'` the boy who the girl

- `$ perl -le '$_ = qq{@ARGV}; print "match!" if /^(the (\w+)( who (?:the (\w+))?(?1) (\w+))?)$/'` the boy who the girl who the woman found saw

match!

- `$ perl -le '$_ = qq{@ARGV}; print "match!" if /^(the (\w+)( who (?:the (\w+))?(?1) (\w+))?)$/'` the boy who the girl who the woman who the dog chased found saw

match!

Homework 10 Review

- *Need to print the right groups!*
- `perl -le '$_ = qq{@ARGV}; /^ (the (\w+) ( who (?:the (\w+)) (?1) (\w+) (?{print ...})) ?) $/'`
- `perl -le '$_ = qq{@ARGV}; /^ (the (\w+) ( who (?:the (\w+)) (?1) (\w+) (?{print "$5($4, $2)"})) ?) $/'`
- Example:

```
perl -le '$_ = qq{@ARGV}; /^ (the (\w+) ( who (?:the (\w+)) (?1) (\w+) (?{print "$5($4, $2)"})) ?) $/' the boy who the girl who the woman who the man who the dog chased knew found saw  
chased(dog, man)  
knew(man, woman)  
found(woman, girl)  
saw(girl, boy)
```

Homework 10 Review

I'm ChatGPT powered by **GPT-6**.

Can you explain who did what to whom in the noun phrase
"the boy who the girl who the woman who the man who the
dog chased knew found saw"?

This is a famous kind of syntactic puzzle: a **multiply center-embedded relative clause**.
The trick is to keep track of which noun each verb belongs to.

The noun phrase is:

the boy who the girl who the woman who the man who the dog chased knew found saw

Homework 10 Review

The complete chain



In plain English:

The girl who saw the boy was found by the woman whom the man knew and whom the dog chased.

Homework 10 Review

The linguistic difficulty is called center embedding: each new relative clause begins before the previous one has finished. The sentence is grammatically interpretable, but its nested structure makes it unusually hard to process in real time.

The girl who saw the boy was found by the woman whom the man knew and whom the dog chased.



xkcd:simplewriter

<https://xkcd.com/simplewriter/>



SIMPLE WRITER

WRITE LIKE *UP GOER FIVE* AND *THING EXPLAINER*

PUT WORDS HERE

Let me explain why this is a good idea. Writing simpler helps people understand complicated concepts.

YOU USED SOME LESS SIMPLE WORDS

complicated concepts

xkcd:simplewriter

RELATIVITY

THE SPECIAL AND GENERAL THEORY

BY
ALBERT EINSTEIN, Ph.D.
PROFESSOR OF PHYSICS IN THE UNIVERSITY OF BERLIN

TRANSLATED BY
ROBERT W. LAWSON, M.Sc.
UNIVERSITY OF SHEFFIELD



NEW YORK
HENRY HOLT AND COMPANY
1921



xkcd:simplewriter

PUT WORDS HERE

THE **present** book is **intended**, as far as possible, to give an **exact insight** into the **theory** of **Relativity** to those readers who, from a **general scientific** and **philosophical** point of view, are interested in the **theory**, but who are not **conversant** with the **mathematical apparatus** of **theoretical physics**. The work **presumes** a **standard** of **education** corresponding to that of a **university matriculation examination**, and, despite the **shortness** of the book, a **fair amount** of **patience** and force of will on the part of the reader. The **author** has **spared** himself no pains in his **endeavour** to **present** the main ideas in the simplest and most **intelligible** form, and on the whole, in the **sequence** and **connection** in which they actually **originated**.

xkcd:simplewriter

```
7 var UpwriterHighlightRules = function() {
8
9   // regexp must not have capturing parentheses. Use (?:) instead.
10  // regexps are ordered -> the first match is used
11
12  this.$rules =
13  {
14    "start": [
15      {
16        "token" : "suffix",
17        "regex" : "['\"] [a-zA-Z]+"
18      },
19      {
20        "token" : "allowed",
21        "regex" : "\\b(?:" + window.__WORDS + ")\b",
22        "caseInsensitive": true
23      },
24      {
25        "token" : "disallowed",
26        "regex" : "[a-zA-Z]+"
27      }
28    ]
29  }
30
31 };
window.__WORDS = "understandings|understanding|conversations|disappearing|info...
rmations|grandmothers|grandfathers|questionings|conversation|information|appro...
aching|understands|immediately|positioning|questioning|grandmother|travellings...
|questioners|recognizing|recognizers|televisions|remembering|rememberers|expre...
ssions|discovering|disappeared|interesting|grandfather|straightest|controllers...
|controlling|considering|remembered|cigarettes|companying|completely|spreading...
s|considered|continuing|controlled|stationing|controller|straighter|stretching...
|businesses|somebodies|soldiering|countering|darknesses|situations|directions|...
disappears|younglings|suggesting|afternoons|breathings|distancing|screenings|s...
choolings|especially|everything|everywhere|explaining|explainers|expression|br...
anchings|revealings|repeating|surprising|rememberer|somewheres|television|the...
mselves|recognizer|recognizes|recognized|belongings|finishings|travelling|ques...
tioner|beginnings|travelings|questioned|followings|pretending|forgetting|forge...
tters|forwarding|positioned|travellers|gatherings|perfecting|understand|unders...
tood|weightings|approaches|officering|numberings|happenings|mentioning|letteri...
ngs|husbanding|imaginings|approached|apartments|whispering|interested|discover...
ed|spinnings|clearings|climbings|spendings|clothings|colorings|soundings|truck...
ings|somewhere|troubling|companies|companied|beautiful|computers|confusing|con...
siders|travelers|youngling|continues|continued|traveller|traveling|yellowing|a...
partment|beginning|wheels|travelled|sometimes|something|appearing|cornering...
```

- `grep -o '|' words.js | wc -l`

Finite State Automata (FSA)

- Given transition table encoded as a (nested) hash
- How to build a **decider** (Accept/Reject) in Perl?

Complications to think about:

- How about ϵ -transitions?
- Multiple end states?
- Multiple start states?
- Non-deterministic FSA?

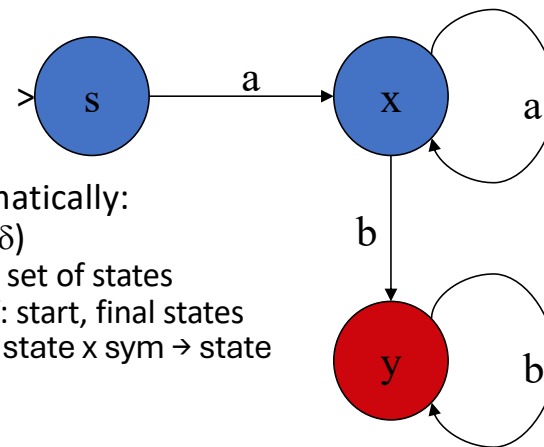
Finite State Automata (FSA)

fsm.perl

```
%transitiontable = (  
    s => {a    => "x"},  
    x => {a    => "x", b    => "y"},  
    y => {b    => "y"}  
);  
$state = "s";  
foreach $c (@ARGV) {  
    $state = $transitiontable{$state}{$c};  
}  
if ($state eq "y") { print "Accept\n"; }  
else { print "Reject\n" }
```

- Example runs:

- perl fsm.perl a b a b
- Reject
- perl fsm.perl a a a b b
- Accept



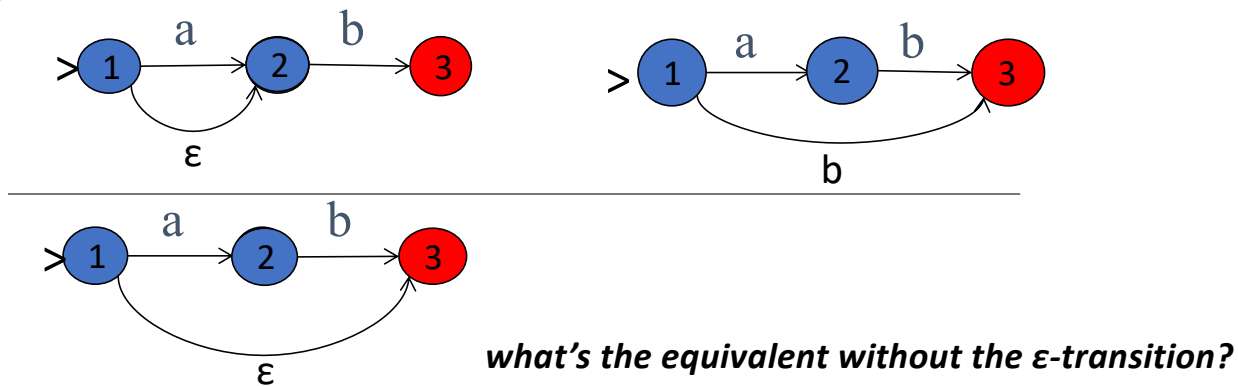
Mathematically:

$(Q, s, f, \Sigma, \delta)$

- Q: set of states
- s, f: start, final states
- δ : state $\times$ sym $\rightarrow$ state

ϵ -transitions

- jump from state to another state with the empty character
 - **ϵ -transition** (*textbook*) or **λ -transition**
 - no increase in expressive power (*meaning we could do without the ϵ -transition*)
- **examples**



ϵ -transitions

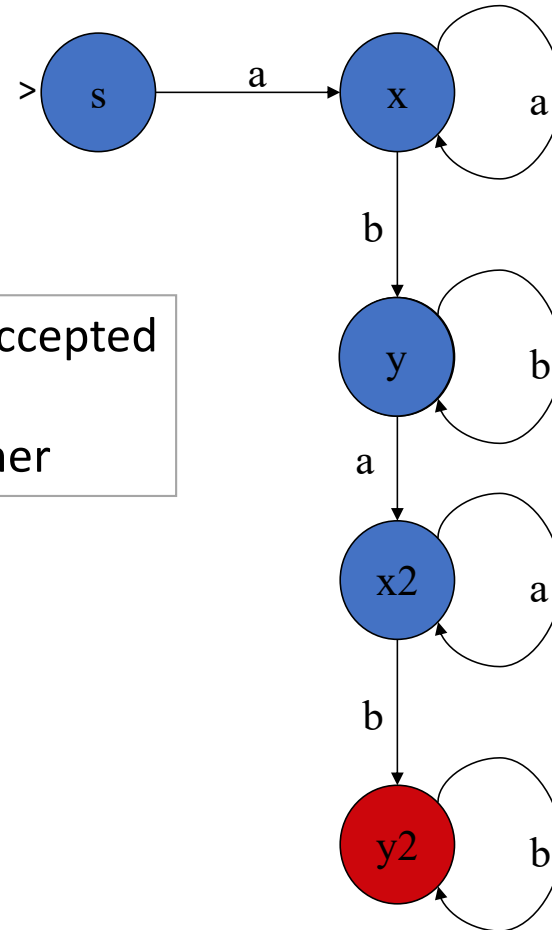
- Can be used to help encode:
 1. Multiple start states
 2. Multiple end states
- Next time, we'll see:
 - Then we can get rid of the ϵ -transition (*by construction*)

Backreferences and FSA

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

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

Doesn't work!
Why?



Backreferences and FSA

fsa.perl

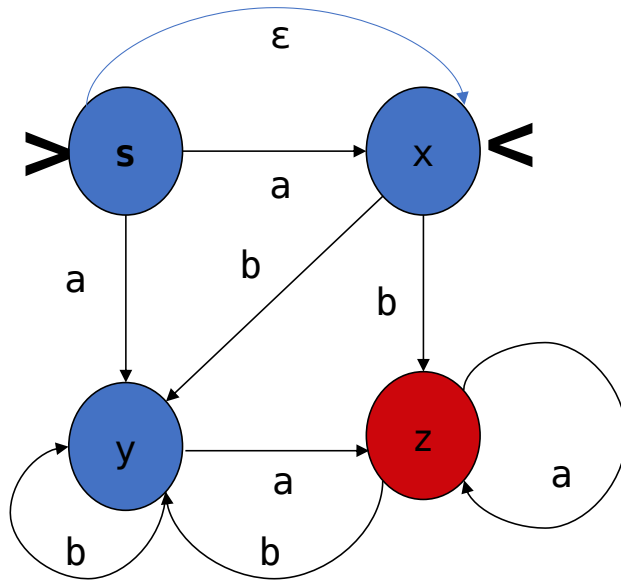
```
1 %delta = (
2     s => { a => "x" },
3     x => { a => "x", b => "y" },
4     y => { b => "y", a => "x2" },
5     x2 => { a => "x2", b => "y2" },
6     y2 => { b => "y2" });
7 $state = "s";
8
9 foreach $c (split(/,/, @ARGV[0])) {
10     $state = $delta{$state}{$c};
11 }
12
13 print (($state eq "y2")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

Perl:

- note line 10: next state is a function of previous state and current symbol **ONLY**
- $\therefore$ # of a's and b's in the two halves don't have to match:
- perl fsa.perl aabba
- Reject
- perl fsa.perl aabbbaaaabbbb
- Accept
- perl fsa.perl aabbbaaab
- Accept

Multiple start states

- Example: simulate this by using an ϵ -transition:



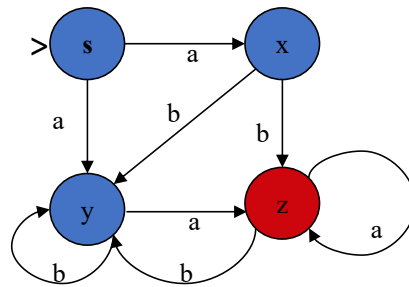
- Multiple final states vs. a single state: also same expressive power.
- Doesn't have to have any final states at all:
 $L(\text{machine}) = \{\}$
- *What's the simplest possible FSA?*

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 $\rightarrow$ (D)FSA

- Idea: 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**

