Language and Computation

week 3, Thursday, January 30, 2014

Tamás Biró
Yale University
tamas.biro@yale.edu
http://www.birot.hu/courses/2014-LC/

Today:

- Understand an algorithm / pseudo-code
- Examples of and issues with algorithms
- Implementing a non-deterministic FSA as a search
- Edit distance as dynamic programming

On pseudo-codes

- The lingo when speaking about algorithms
- Half way between human language and programming languages
- Relatively straightforward to translate to your favorite programming language
- Focus on important aspects, skip over details

Running a deterministic FSA

A massively distributed, non-silicon-based implementation:

- Each state = one student
- Each student with detailed instructions

Running a deterministic FSA

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

  index ← Beginning of tape
  current-state ← Initial state of machine
  loop
   if End of input has been reached then
    if current-state is an accept state then
      return accept
    else
      return reject
   elsif transition-table[current-state,tape[index]] is empty then
     return reject
   else
     current-state ← transition-table[current-state,tape[index]]
     index ← index + 1
  end
```

Non-determinism

Two sources:

- Two arcs with the same symbol
- ϵ-transitions.

Running a non-deterministic FSA

Q: How to have a deterministic computer simulate a non-deterministic automaton?

- Transform ND-FSA into D-FSA (by replacing states with set of states)
- look-ahead
- parallelism
- backup: maintaining an $agenda$, that is, a set of all currently unexplored choices ($search\ states$: node-position pairs).

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
```

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*)
$\cup$
(*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

State-space search

- Depth-first (LIFO)
- Breadth-first (FIFO)

See you next week!