CS410/510 Advanced Programming
Lecture 8:

Finite Automata
REs and NFSMs

A NFSM is defined as a tuple:
- an alphabet, \( A \)
- a set of states, \( S \)
- a transition function, \( A \times S \rightarrow 2^S \)
- a start state, \( S_0 \)
- a set of accepting states, \( S_F \), a subset of \( S \)

Defined by cases over the structure of regular expressions:
- Let \( A, B \) be R.E.’s, “x” in A, then
  - \( \varepsilon \) is a R.E.
  - “x” is a R.E.
  - \( AB \) is a R.E.
  - \( A + B \) is a R.E.
  - \( A^* \) is a R.E.

One construction rule for each case
Rules

- $\varepsilon$

- "x"

- $AB$

- $A+B$

- $A^*$
Example: \((a+b)^*abb\)

- Note the many \(\epsilon\) transitions
- Loops caused by the *
- Non-Determinism: many paths out of state 0 on “a”
FSMs in Smalltalk

• Add a class: NFSM
  • instance variables
  • class-side methods to create simple machines
FSMs in Smalltalk

Class definition for NFSM

Object subclass: *NFSM
  instanceVariableNames: 'states initialState finalStates'
  classVariableNames: ''
  poolDictionaries: ''
  category: 'CS510ap-RegularExpressions'

I represent a Nondeterministic Finite State Machine

Structure:
states          a Set of FSMState -- my states, and the transitions that they can take.
initialStates  a subset of States -- my initial states
finalStates    a subset of States -- my final (accepting) states
What about the transitions?

• I decided to put the transitions in the states
  • so, a transition $q_1 \xrightarrow{x} q_2$ is a property of state $q_1$
  • represent it as an association $x \rightarrow q_2$

• Why?
What about the transitions?

I decided to put the transitions in the states, so a transition $q_1 \rightarrow q_2$ is a property of state $q_1$.

Why?

I represent a state in a Finite State Machine.

My instance variable transitions holds a set of associations from input symbols to states. If symb $\rightarrow S$ is in transitions, then the machine of which I am part can make the transition to state $S$ when reading input symb.
States have interesting behavior

- We can ask a state: to what states can you make a transition when reading aSymbol from the input?
- This hides the particular representation of the transition function from clients.
Internals of FSMState

**instance variables:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>InitialValue</td>
<td>a Set()</td>
</tr>
<tr>
<td>inferredType</td>
<td>Set</td>
</tr>
</tbody>
</table>

**transitions**

**instance methods:**

**accessing**

transitionsOn: aSymbol

^ transitions
  select: [:assoc | assoc key = aSymbol]
  thenCollect: [:assoc | assoc value]

**as yet unclassified**

initialize

transitions := Set new

printOn: s

  super printOn: s.
  s nextPutAll: '([ #'.
  s print: self identityHash.
  s nextPutAll: ': '.
  transitions do: [:each | s print: each key; nextPutAll: ' -> #'; print: (each value identityHash) ] separatedBy: [s nextPutAll: ', '].
  s nextPutAll: ' )]

**initialization**

add: anAssociation

^ transitions add: anAssociation.
More details …

- Look at:
  - tests
  - machine creation
  - sets of states
Building an NFSM from a RE

- **factory methods** in the NFSM class:
Simulating an NFSM

- Given a string, say “ababb”, run the NFSM and determine if the NFSM “accepts” the string.
- $\epsilon$-closure: all the states reachable from a given set via $\epsilon$-transitions.
  - effective initial state is $\epsilon$-closure of $\{q_0\}$
- at all times, keep track of what set of states the machine could possibly be in.
• Initial state is 0
• ε –closure of 0 is {0;1,2,4,7}
• From any of {0;1,2,4,7}
  – We can make a transition on “a” to {3,8}
  – We can make a transition on “b” to {5}
• ε –closure of {3,8} is {3,8;6,7,0,1,2,4}
• ε –closure of {5} is {5;6,7,0,1,2,4}
• From any of {3,8;6,7,0,1,2,4}
  – We can make a transition on “a” to {3,8} -- which we’ve seen before
  – We can make a transition on “b” to {9,5} -- which is new
• From any of {4;6,7,0,1,2,4}
  – We can make a transition on “a” to {3,8} -- which we’ve seen before
  – We can make a transition on “b” to {5} -- which we’ve seen before
ε –closure of {9,5} is {9;6,7,0,1,2,4}
• From any of {9;6,7,0,1,2,4}
  – We can make a transition on “a” to {3,8} -- which we’ve seen before
  – We can make a transition on “b” to {10,5} -- which is new
Example: “ababb”

<table>
<thead>
<tr>
<th>state</th>
<th>input</th>
</tr>
</thead>
<tbody>
<tr>
<td>{0;1,2,4,7}</td>
<td>“ababb”</td>
</tr>
<tr>
<td>{3,8;6,7,0,1,2,4}</td>
<td>“babb”</td>
</tr>
<tr>
<td>{9,5;6,7,0,1,2,4}</td>
<td>“abb”</td>
</tr>
<tr>
<td>{9,5;6,7,0,1,2,4}</td>
<td>“bb”</td>
</tr>
<tr>
<td>{9,5;6,7,0,1,2,4}</td>
<td>“b”</td>
</tr>
<tr>
<td>{10,5;6,7,0,1,2,4}</td>
<td>“”</td>
</tr>
</tbody>
</table>

Final state includes the accepting state, 10, so the string is accepted.
Example: “ababb”

<table>
<thead>
<tr>
<th>state</th>
<th>input</th>
<th>final state includes the accepting state, 10, so the string is accepted.</th>
</tr>
</thead>
<tbody>
<tr>
<td>{0;1,2,4,7}</td>
<td>“ababb”</td>
<td></td>
</tr>
<tr>
<td>{3,8;6,7,0,1,2,4}</td>
<td>“babb”</td>
<td></td>
</tr>
<tr>
<td>{9,5;6,7,0,1,2,4}</td>
<td>“abb”</td>
<td></td>
</tr>
<tr>
<td>{3,8;6,7,0,1,2,4}</td>
<td>“bb”</td>
<td></td>
</tr>
<tr>
<td>{9,5;6,7,0,1,2,4}</td>
<td>“b”</td>
<td></td>
</tr>
<tr>
<td>{10,5;6,7,0,1,2,4}</td>
<td>“”</td>
<td></td>
</tr>
</tbody>
</table>

accepts: aString

<table>
<thead>
<tr>
<th>possibleStates</th>
</tr>
</thead>
<tbody>
<tr>
<td>possibleStates := FSMStateSet with: initialState.</td>
</tr>
<tr>
<td>possibleStates := possibleStates withEpsilonTransitions.</td>
</tr>
<tr>
<td>aString do:</td>
</tr>
<tr>
<td>[:c</td>
</tr>
<tr>
<td>possibleStates := possibleStates transitionsOn: c.</td>
</tr>
<tr>
<td>possibleStates ifEmpty: [^false].</td>
</tr>
<tr>
<td>possibleStates := possibleStates withEpsilonTransitions].</td>
</tr>
<tr>
<td>^finalStates includesAnyOf: possibleStates</td>
</tr>
</tbody>
</table>
Deterministic FSM

- A DFSM is defined as a tuple:
  - an alphabet, $A$
  - a set of states, $S$
  - a transition function, $A \times S \rightarrow S$
  - a start state, $S_0$
  - a set of accepting states, $S_F$, a subset of $S$
- Only one choice on each input
Converting an NFSM into a DFSM

• States in the deterministic $d$ machine represent \textit{sets of states} in the nondeterministic machine $n$

• if $n$ has $k$ states, then $d$ has $2^k$ states, one for each subset of $n$’s states

• the accepting states are those that contain a state of $n$ that was in $n$’s set of accepting states.

• the transitions of $d$ are …
Homework

- Start with my RegEx.\.cs changeset; make sure that the tests run green.
- Implement a DFSM class.
- Write tests that compare the strings accepted by equivalent NFSM and DFSMs.
- Implement the algorithm to convert an NFSM to a DFSM.
- Make the tests run green.