Regular expressions and regular languages

Two base cases and three inductive rules:

<table>
<thead>
<tr>
<th>$E$</th>
<th>$L(E)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. $\epsilon$</td>
<td>{\epsilon}</td>
</tr>
<tr>
<td>2. $a$ (for any $a \in \Sigma$)</td>
<td>{a}</td>
</tr>
<tr>
<td>3. $R \mid S$</td>
<td>$L(R) \cup L(S)$</td>
</tr>
<tr>
<td>4. $R \cdot S$</td>
<td>$L(R) \cdot L(S)$</td>
</tr>
<tr>
<td>5. $R^*$</td>
<td>$L(R)^*$</td>
</tr>
</tbody>
</table>

Example: $L((a \cdot b)|c)^*$ is

\{\epsilon, “ab”, “c”, “abab”, “abc”, “cab”, …\}
Non-deterministic finite-state acceptors

An abstract machine that *accepts* or *rejects* a string

accepts \((ab) \mid (ac(c*))\)
Run the automaton on strings accc, abc (use two tacks)

Stress the idea of non-determinism as angelic choice or exploring all possibilities simultaneously

2-1
Why NFA’s?

- Easy construction from regular expressions
- Automatic transformation to DFA’s
DFA

Deterministic finite-state acceptors

Also accepts \((ab) \mid (ac(c^*))\)
DFA as a table:

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>Final?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>3</td>
<td>Yes</td>
</tr>
</tbody>
</table>
• Trap state takes place of —

• In practical applications, transition has an action associated with it (like assignment 1)
DFA formal definition

\[\langle Q, \Sigma, \delta, q_0, F \rangle\]

\(Q\)         A finite set of states

\(\Sigma\)    A finite alphabet (set of symbols)

\(\delta: Q \times \Sigma \to Q\) the transition function

For an NFA: \(\Delta: 2^Q \times \Sigma \to 2^Q\)

\(q_0 \in Q\) the start state

\(F \subseteq Q\) the final states
From NFA to DFA

DFA states represent *sets* of NFA states

Fundamental operation: Transition from set of NFA states to set of NFA states
$\epsilon$-closure

Typical graph reachability problem

Equivalent to identifying a connected component after removing all edges except $\epsilon$
Subset construction

Traces paths through the NFA

Let NFA be \( \langle S, \Sigma, \Delta, s_0, G \rangle \)

Let DFA be \( \langle Q, \Sigma, \delta, q_0, F \rangle \)

\( q_0 = \epsilon - \text{closure}(s_0) \)

\( Q \) is initially \( \{q_0\} \)

\( \text{Extend}(q) = \)
  
  for \( a \in \Sigma \) do
  
  \( r := \epsilon - \text{closure}(\Delta(q, a)) \);
  
  add \( ((q, a), r) \) to \( \delta \);
  
  if \( r \not\in Q \) then
    
    add \( r \) to \( Q \);
    
    extend\( (r) \);
    
  fi;
  
  od;

end;
Minimizing DFA size

May have extra states because of repetition, e.g. \((a \mid b) * abb\)

Typical partition refinement algorithm, starting from final states.

Examples:

- \((a \mid b) * abb\)

- \(((xx * y) \mid (xy * x))z\)
Real-world lexical analysis

We want a scanner, not an acceptor

- *Lots* of patterns — distinguish keywords from numbers from quoted strings from ...

  *No problem: Just alternation, but keep the final states straight*

- A *sequence* of tokens

  *No problem: when no transition is defined, produce token from last accepting state*

- Each pattern should be associated with an action

  *No problem: associate actions with final states*
Limitations of Regular Languages

Recall — a language is a *set of strings*

A *regular* language can be described by a regular expression

Every regular language can be recognized by a DFA

But not every language is regular
Pumping lemma

If a language is regular, it is recognized by a DFA

A DFA has a fixed number of states, $n$

A very long string $z$ ( $|z| \geq n$) must repeat a state
Pumping lemma (2)

A very long string $z$ ( $|z| \geq n$) must repeat a state

Divide it into $u, v, w$, with $v$ the first loop

If $uvw$ is in $L$, so is $uw, uvvw, uvvww$, etc.
Pumping lemma — formal statement

Let $L$ be a regular language.

There exists a fixed $n$ such that

if $z \in L$ and $|z| \geq n$
then $z = uvw$, $|uv| \leq n$, $|v| \geq 1$,
and for all $i \geq 0$, $uv^iw \in L$
Pumping lemma — consequences

\(a^ib^i\) is not regular — no DFA can recognize it.

(Because: You can form a long enough string to pump just ‘a’, wrecking the balance)

Balanced and hierarchical structures in general are not regular

- Balanced parenthesis
- Properly nested begin/end pairs
- if / else pairs
- \ldots

Programming language grammars are not regular
Pumping lemma — intuition

Rule of thumb: Regular expressions can’t count

The whole “memory” of a DFA is encoded in a single state; since there are a finite number of states, a DFA has bounded memory.

Counting (with no limit) requires unbounded memory.

Note: All finite languages are regular