Implementing Regular Expressions

- Build finite automata for all patterns
- Connect start states for NDFA
- Simplify to make DFA
- Algorithm:
  - When asked for token, start in combined state
  - Read characters, advancing state, until cannot advance further
  - If needed, push back last character(s)
  - Return token associate with last state

Finite State Automata

- A recognizer for a language is a program that takes a string x as an input and answers "yes" if x is a sentence of the language and "no" otherwise.
- One can compile any regular expression into a recognizer by constructing a generalized transition diagram called a finite automaton.
- A finite automaton can be non-deterministic or deterministic.
- Both automata are capable of recognizing regular expressions.
Transition Diagram

- Flowchart with states and edges; each edge is labeled with characters; certain subset of states are marked as “final states”
- Transition from state to state proceeds along edges according to the next input character
- Every string that ends up at a final state is accepted
- If get “stuck”, there is no transition for a given character, it is an error

\[
\text{rellop} \rightarrow < | > | <= | >= | = | <>
\]

TOKER getRelop()
{
    TOKEN relToken = new(RELOP);
    while(1){ /* repeat character processing until a return or failure occurs */
        switch(state){
            case 0: c = nextChar();
                if (c == '<' ) state = 1;
                else if ( c == '>' ) state = 5;
                else if ( c == '=' ) state = 6;
                else fail(); /* lvalue is not a relop */
                break;
            case 1: ...
                ...
            case 8: retract();
                relToken.attribute = GT;
                return(relToken);
            }
    }
}
NFA vs. DFA

- **Nondeterministic Finite Automata (NFA):**
  - ε can label edges (these edges are called ε-transitions)
  - some character can label 2 or more edges out of the same state
- **Deterministic Finite Automata (DFA):**
  - no edges are labeled with ε
  - each character can label at most one edge out of the same state
- **NFA and DFA accepts string x if there exists a path from the start state to a final state labeled with characters in x**

(a|b)*abb

Figure 3.21: Trie for keywords he, she, his, hers

Figure 3.26: A nondeterministic finite automaton

Figure 3.28: DFA accepting (a|b)*abb
Transition Tables

• For NFA, each entry is a set of states:

<table>
<thead>
<tr>
<th>STATE</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(0,1)</td>
<td>(0)</td>
</tr>
<tr>
<td>1</td>
<td>(2)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(3)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

• For DFA, each entry is a unique state:

<table>
<thead>
<tr>
<th>STATE</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

NFA w/ ϵ-transitions

• (aa* | bb*)

NFA to DFA

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε-closure(s)</td>
<td>Set of NFA states reachable from NFA state s on ε-transitions alone.</td>
</tr>
<tr>
<td>ε-closure(T)</td>
<td>Set of NFA states reachable from some NFA state s in set T on ε-transitions alone: ( \bigcup_{s \in T} \varepsilon\text{-closure}(s) ).</td>
</tr>
<tr>
<td>move(T, a)</td>
<td>Set of NFA states to which there is a transition on input symbol a from some state s in T.</td>
</tr>
</tbody>
</table>

while ( there is an unmarked state T in Dstates ) {
  mark T;
  for ( each input symbol a ) {
    U = ε-closure(move(T, a));
    if ( U is not in Dstates )
      add U as an unmarked state to Dstates;
    Dtrans[T, a] = U;
  }
}
DFA - (a|b)*abb

Figure 3.34: NFA N for (a|b)*abb

<table>
<thead>
<tr>
<th>NFA STATE</th>
<th>DFA STATE</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>{0, 1, 2, 4, 7}</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>{1, 2, 3, 4, 6, 7, 8}</td>
<td>B</td>
<td>B</td>
<td>D</td>
</tr>
<tr>
<td>{1, 2, 4, 5, 6, 7}</td>
<td>C</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>{1, 2, 4, 5, 6, 7, 9}</td>
<td>D</td>
<td>B</td>
<td>E</td>
</tr>
<tr>
<td>{1, 2, 4, 5, 6, 7, 10}</td>
<td>E</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>

Figure 3.36: Result of applying the subset construction to Fig. 3.34