Finite Automata

Definition Finite Automata
A non-deterministic finite automaton (NFA) is a tuple $A = (Q, \Sigma, I, F, \delta)$ with:

- $Q$ a finite set of states;
- $\Sigma$ a finite alphabet of inputs;
- $I \subseteq Q$ the set of start states;
- $F \subseteq Q$ the set of final states and
- $\delta$ the set of transitions (-relation)

For an NFA, we reckon:

Definition Deterministic Finite Automata
Given $\delta : Q \times \Sigma \rightarrow Q$ a function and $|I| = 1$, then we call the NFA $A$ deterministic (DFA).

Finite Automata

- Computations are paths in the graph.
- Accepting computations lead from $I$ to $F$.
- An accepted word is the sequence of labels along an accepting computation ...

Once again, more formally:

- We define the transitive closure $\delta'$ of $\delta$ as the smallest set $\delta'$ with:

  $$(p, \epsilon, p), (p, x, p_n) \in \delta' \quad \text{and} \quad \forall (p, x, q) \in \delta \quad \text{and} \quad \forall (p, w, q) \in \delta',$$

  $\delta'$ characterizes for a path between the states $p$ and $q$ the words obtained by concatenating the labels along it.

- The set of all accepting words, i.e. $A$'s accepted language can be described compactly as:

  $$\mathcal{L}(A) = \{ w \in \Sigma^* : |w| \in I \in F \in \delta' \}$$
In Linear Time from Regular Expressions to NFAs

Thompson’s Algorithm
Produces $O(n)$ states for regular expressions of length $n$.

Berry-Sethi Approach

Berry-Sethi Algorithm
Produces exactly $n + 1$ states without $\varepsilon$-transitions and demonstrates $\rightarrow$ Equality Systems and $\rightarrow$ Attribute Grammars

Idea:
The automaton tracks (conceptionally via a marker “$\ast$”), in the syntax tree of a regular expression, which subexpressions in $e$ are reachable consuming the rest of input $w$.

Berry-Sethi Approach

... for example:

Glushkov Automaton
Produces exactly $n + 1$ states without $\varepsilon$-transitions and demonstrates $\rightarrow$ Equality Systems and $\rightarrow$ Attribute Grammars
Berry-Sethi Approach

Construction (naive version):

States: \( *r, r, e \) with \( r \) nodes of \( e \);
Start state: \( e \);
Final state: \( e \);
Transitions: for leaves \( r = \ast \ldots \ast \) we require: \( (*r, r, e) \).
The leftover transitions are:

<table>
<thead>
<tr>
<th>( r_1 \cup r_2 )</th>
<th>Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_1 )</td>
<td>( *r, r, e )</td>
</tr>
<tr>
<td></td>
<td>( e, r, e )</td>
</tr>
<tr>
<td></td>
<td>( e, e, e )</td>
</tr>
<tr>
<td>( r_2 )</td>
<td>( *r, r, e )</td>
</tr>
<tr>
<td></td>
<td>( e, r, e )</td>
</tr>
<tr>
<td></td>
<td>( e, e, e )</td>
</tr>
</tbody>
</table>

Berry-Sethi Approach

Discussion:
- Most transitions navigate through the expression
- The resulting automaton is in general nondeterministic

\[ \Rightarrow \text{Strategy for the sophisticated version:} \]
Avoid generating \( e \)-transitions

Idea:
Pre-compute helper attributes during \( D(\text{epht})F(\text{irst})S(\text{each})! \)

Necessary node-attributes:
- \( \text{first} \): the set of read states below \( r \) which may be reached \( \text{first} \), when descending into \( r \).
- \( \text{next} \): the set of read states to the right of \( r \), which may be reached \( \text{first} \) in the traversal after \( r \).
- \( \text{last} \): the set of read states below \( r \), which may be reached \( \text{last} \) when descending into \( r \).
- \( \text{empty} \): can the subexpression \( r \) consume \( e \)?

Berry-Sethi Approach: 1st step

\[ \text{empty}[r] = 1 \text{ if and only if } \exists e \subseteq [r] \]

... for example:
Berry-Sethi Approach: 1st step

Implementation:
DFS post-order traversal

for leaves \( r \equiv \text{leaf} \) we find \( \text{empty}(r) = (x = \epsilon) \).

Otherwise:
\[
\begin{align*}
\text{empty}(r_1 \cdot r_2) &= \text{empty}(r_1) \vee \text{empty}(r_2) \\
\text{empty}(r_1 \cdot r_2) &= \text{empty}(r_1) \wedge \text{empty}(r_2) \\
\text{empty}(r_1^*) &= t
\end{align*}
\]

Berry-Sethi Approach: 2nd step

Implementation:
DFS post-order traversal

for leaves \( r \equiv \text{leaf} \) we find \( \text{first}(r) = \{ i \mid x \neq \epsilon \} \).

Otherwise:
\[
\begin{align*}
\text{first}(r_1 \cdot r_2) &= \text{first}(r_1) \cup \text{first}(r_2) \\
\text{first}(r_1 \cdot r_2) &= \begin{cases} \\
\text{first}(r_1) \cup \text{first}(r_2) & \text{if} \quad \text{empty}(r_1) = t \\
\text{first}(r_1) & \text{if} \quad \text{empty}(r_1) = f
\end{cases}
\end{align*}
\]

Berry-Sethi Approach: 3rd step

Implementation:
DFS post-order traversal

The may-set of first reached read states: The set of read states, that may be reached from \( \tau \) (i.e. while descending into \( \tau \)) via sequences of \( \epsilon \)-transitions:
\[
\text{first}(r) = \{ i \mid (i, \epsilon, * \tau, \epsilon) \in \delta' \}, \quad x \neq \epsilon
\]

... for example:

![Diagram](image1)

The may-set of next read states: The set of read states within the subtrees right of \( \tau \) that may be reached next via sequences of \( \epsilon \)-transitions.
\[
\text{next}(r) = \{ i \mid (i, \epsilon, * \tau, \epsilon) \in \delta' \}, \quad x \neq \epsilon
\]

... for example:

![Diagram](image2)
**Implementation:**

**DFS pre-order traversal**

For the root, we find: \( \text{next}[r] = \emptyset \)

Apart from that we distinguish, based on the context:

<table>
<thead>
<tr>
<th>( r )</th>
<th>( \text{next}[r] )</th>
<th>Equalities</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_1 )</td>
<td>( r_1 )</td>
<td>( \text{next}[r_1] = \emptyset )</td>
</tr>
<tr>
<td>( r_1 )</td>
<td>( r_2 )</td>
<td>( \text{next}[r_2] = \text{next}[r] )</td>
</tr>
</tbody>
</table>
| \( r_1 \) | \( r_3 \) | \( \text{next}[r_3] = \left\{ \begin{array}{l} \text{first}[r_3] \cup \text{next}[r] \\
\text{first}[r_3] \end{array} \right\} \text{if empty}[r_3] = f \\
\text{next}[r_2] \text{if empty}[r_3] = t \) |
| \( r_1 \) | \( r_4 \) | \( \text{next}[r_4] = \text{next}[r] \) |
| \( r_1 \) | \( r_7 \) | \( \text{next}[r_7] = \text{next}[r] \) |

**Berry-Sethi Approach: 3rd step**

The may-set of next read states: The set of read states within the subtrees right of \( \ast \), that may be reached next via sequences of \( \epsilon \)-transitions.

\[ \text{next}[r] = \{ r | (r, \ast, \epsilon, \emptyset, \emptyset) \in \delta', x \neq \epsilon \} \]

... for example:

```
  0 1 2 0 1
  +---------+
  0 1 2 3 4
  +---------+
```

**Implementation:**

**DFS pre-order traversal**

For the root, we find: \( \text{next}[r] = \emptyset \)

Apart from that we distinguish, based on the context:

<table>
<thead>
<tr>
<th>( r )</th>
<th>( \text{next}[r] )</th>
<th>Equalities</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_1 )</td>
<td>( r_2 )</td>
<td>( \text{next}[r_2] = \text{next}[r] )</td>
</tr>
</tbody>
</table>
| \( r_1 \) | \( r_3 \) | \( \text{next}[r_3] = \left\{ \begin{array}{l} \text{first}[r_3] \cup \text{next}[r] \\
\text{first}[r_3] \end{array} \right\} \text{if empty}[r_3] = f \\
\text{next}[r_2] \text{if empty}[r_3] = t \) |
| \( r_1 \) | \( r_4 \) | \( \text{next}[r_4] = \text{next}[r] \) |
| \( r_1 \) | \( r_7 \) | \( \text{next}[r_7] = \text{next}[r] \) |
| \( r_1 \) | \( r_8 \) | \( \text{next}[r_8] = \text{next}[r] \) |
Berry-Sethi Approach: 3rd step

The may-set of next read states: The set of read states within the subtrees right of \( \rightarrow x \) that may be reached next via sequences of \( \epsilon \)-transitions.

\[ \text{next}[r] = \{ i \mid \langle r, x, \epsilon, \rightarrow x \rangle \in \delta', x \neq \epsilon \} \]

... for example:

```
  12
    1
   / \
  0   2
    \   \
     0
```

Berry-Sethi Approach: 4th step

Implementation:

DFS post-order traversal

for leaves \( r \equiv \epsilon \cdot x \) we find \( \text{last}[r] = \{ i \mid x \neq \epsilon \} \).

Otherwise:

\[
\begin{align*}
\text{last}[r_1 \cdot r_2] &= \text{last}[r_1] \cup \text{last}[r_2] \\
\text{last}[r_1 \cdot r_2] &= \{ \text{last}[r_1] \cup \text{last}[r_2] \} \quad \text{if empty}(r_2) = t \\
\text{last}[r_1?] &= \text{last}[r_1] \\
\text{last}[r_1?] &= \text{last}[r_1]
\end{align*}
\]

Berry-Sethi Approach: (sophisticated version)

Implementation:

DFS pre-order traversal

For the root, we find:

\( \text{next}[] = \emptyset \)

Apart from that we distinguish, based on the context:

<table>
<thead>
<tr>
<th>( r )</th>
<th>Equalities</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_1 \cdot r_2 )</td>
<td>( \text{next}[r_1] = \text{next}[r_2] )</td>
</tr>
<tr>
<td>( r_1 \cdot r_2 )</td>
<td>( \text{next}[r_2] = \text{next}[r_1] )</td>
</tr>
<tr>
<td>( r_1 \cdot r_2 )</td>
<td>( \text{first}[r_1], \text{next}[r_2] ) if ( \text{empty}(r_1) = t )</td>
</tr>
<tr>
<td>( r_1 \cdot r_2 )</td>
<td>( \text{first}[r_2], \text{next}[r_1] ) if ( \text{empty}(r_2) = f )</td>
</tr>
<tr>
<td>( r_1 )</td>
<td>( \text{last}[r_1] = \text{next}[r_1] )</td>
</tr>
<tr>
<td>( r_1 )</td>
<td>( \text{last}[r_1] = \text{next}[r_1] )</td>
</tr>
</tbody>
</table>

Construction (sophisticated version):

Create an automaton based on the syntax tree's new attributes:

- States: \( \{ \epsilon \} \cup \{ i \mid i \text{ a leaf} \} \)
- Start state: \( \epsilon \)
- Final states: \( \text{last}[\epsilon] \) if \( \text{empty}[\epsilon] = f \)
- \( \{ \epsilon \} \cup \{ \text{last}[\epsilon] \} \) otherwise
- Transitions:
  - \( \{ \epsilon, a, \epsilon \} \) if \( a \in \text{first}[r] \) and \( a \) labeled with \( o \) and \( \text{next}[r] \) labeled with \( w \)

We call the resulting automaton \( A_e \).
Berry-Sethi Approach

... for example:

Remarks:
- This construction is known as Berry-Sethi- or Glushkov-construction.
- It is used for XML to define Content Models
- The result may not be, what we had in mind...