Bottom-Up Parsing Basics

Example: Rightmost Derivation

Input String: $aaabffcc$

$$\begin{align*}
S &\rightarrow ABc \\
A &\rightarrow Aa \mid a \\
B &\rightarrow bF \mid b \\
F &\rightarrow fF \mid f
\end{align*}$$

Input String: $aaabffcc$

The symbol stack summarizes the consumed string.

Example: Symbol Stack

<table>
<thead>
<tr>
<th>Input String</th>
<th>Symbol Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>$aa$ $abffc$</td>
<td>$A$</td>
</tr>
<tr>
<td>$aaab$ $ffc$</td>
<td>$Ab$</td>
</tr>
<tr>
<td>$aaabff$ $c$</td>
<td>$AbF$</td>
</tr>
<tr>
<td>$aaabff$ $c$</td>
<td>$AB$</td>
</tr>
</tbody>
</table>

Example: Item (NFA state)

<table>
<thead>
<tr>
<th>Input String</th>
<th>Symbol Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>$aaabff$ $c$</td>
<td>$S \rightarrow AB.c$</td>
</tr>
</tbody>
</table>

The NFA state shows the part of a rule reachable using the consumed string.
Example: State transitions

Input String: \texttt{aaa | bffc}
\texttt{aaaab | ffc aaaab | ffc}

State Transition

Symbol stack and State stack

Input String: $\texttt{aaa b f c}$
Symbol Stack: $\texttt{A b f}$

Note 1: Redundancy

- The state stack can be generated from the sequence of symbols on the symbol stack, by tracing the derivation from the start symbol, using the productions.
- The state stack is used for efficiency.
- It avoids running the entire DFA (containing production fragments) to determine the next state each time.

Why trace the entire symbol stack?

It is not possible to determine the “current” production (state) by merely inspecting a fixed number of stack symbols.

Input 1:
Symbol Stack: $\{\texttt{aaa a}\}$
State: $A \rightarrow a, A$

Input 2:
Symbol Stack: $\{\texttt{aaa a}\}$
State: $B \rightarrow a, B$

\texttt{S -> [A] | {B}}
\texttt{A -> a A | a}
\texttt{B -> a B | a}$
DFA State: Input strings with eqvt. history
: RHS of rules sharing a prefix

Input String 1: ab ffc
Input String 2: ab ge
Symbol Stack: $ A b

The DFA state captures all possible rule prefixes reachable from the start symbol using the consumed input.

Stacking states is adequate

S => A B c => A b c
S => A D e => A d B c => A d b c

The effect of reduction by B -> b depends on the previous state.

Note 2: Redundancy

- The state stack contains enough information to regenerate the symbol stack. Note that all arcs into a DFA state carry the same symbol label.
- As a consequence, the symbol stack can be avoided by making the "reduce"-entry in the parse table specify the number of stack elements (length of RHS) to be popped and the (LHS) nonterminal to make the transition.
- In practice, the (synthesized) semantic attribute associated with a stack symbol is allocated and computed on a parallel stack.

Relation to Grammars

- The DFA naturally corresponds to a left-linear grammar.

• The DFA state is obtained from the NFA states using the standard construction.