Parsing

Simple Precedence Parsing Algorithm

Input: a simple precedence matrix or functions
       grammar
       sentence with \( \downarrow \) appended to the end

Assumptions: \( \downarrow < V_i \) \( \downarrow \) where \( V_i \in V, \downarrow \notin V \)

Algorithm:

1. Push \( \downarrow \) onto stack
2. Read first input symbol and push it onto stack
3. Do
   3.1 Obtain SP relation between the symbol on top of the stack and the next input symbol
   3.2 If SP relation is \( < \text{ or } = \)
       Then
       3.2.1 Stack input symbol.
   Else \{ relation is \( > \} \}
       3.2.2 Pop top of stack into handle
       3.2.3 Obtain the relation between the top of the stack and the leftmost symbol in the handle
       3.2.4 While SP relation is \( = \) Do
           3.2.4.1 Pop top of stack into handle
           3.2.4.2 Obtain the SP relation between the top of the stack and the leftmost symbol
                      in the handle
           3.2.5 Match the handle against the RHS of all productions
           3.2.6 Push the LHS of the matched RHS onto the stack
4. Until end-of-file and only \( \downarrow \) and the start symbol are on the stack

Note: The algorithm above does not detect any syntax errors.

Def: A simple precedence parse configuration triple is \((S, \alpha, \pi)\) where \(S\) is the stack, \(\alpha\) is the input string, and \(\pi\) is the action sequence.
Initially, we have \((\bot, \alpha, \lambda)\) where \(\bot\) represents the empty stack and \(\alpha\) is the input string. We terminate when we have \((S, \lambda, \pi)\) where \(S\) is the start symbol and \(\pi\) is the action sequence. Using the grammar and simple precedence matrix below, we construct the simple precedence parse configuration sequence for the input string \(V^* (V + V)\)

\[
\begin{align*}
(\bot, & \ V^* (V + V) \bot, \lambda) \\
1. & E \to E + T - \\
2. & T - \\
3. & T - \to T \\
4. & T \to T * P \\
5. & P \\
6. & P \to (E -) \\
7. & V \\
8. & E - \to E
\end{align*}
\]

\[
\begin{array}{cccccccc}
| E | T | P | - | + | * | ( | ) | V |
\hline
E & \ldots & \ldots & = & \ldots & G & . \\
T^- & \ldots & \ldots & G & . & G & . \\
T & \ldots & \ldots & G & = & \ldots & G & . \\
P & \ldots & \ldots & G & G & . & G & . \\
E^- & \ldots & \ldots & \ldots & \ldots & = & \ldots & . \\
+ & \ldots & \ldots & = & \ldots & L & L & \ldots & . \\
\times & \ldots & \ldots & \ldots & \ldots & \ldots & L & L & \ldots & . \\
( & \ldots & \ldots & \ldots & \ldots & \ldots & L & L & \ldots & . \\
) & \ldots & \ldots & G & G & . & G & . \\
V & \ldots & \ldots & G & G & . & G & .
\end{array}
\]
Simple Precedence Parsing Errors

In all of our examples above we have only had sentences in the language of the simple precedence grammar to parse. This situation will not always be the case. There are three types of simple precedence parsing errors, namely character pair errors, reducibility errors, and stackability errors. We will examine each of these errors next.

Def: A character pair error occurs when there is no simple precedence relation between pairs of symbols in the grammar.

A character pair error is identified when you go the simple precedence matrix, and there is no simple precedence relation between the two symbols. In general, this type of error is most common. In an experiment that was previously conducted, character pair errors occur approximately 85% of the time that there are syntax errors in the string that you are parsing.

Def: A reducibility error occurs when you cannot reduce the handle to the left hand side of some production.

A reducibility error is identified when you take the potential handle and cannot find a matching right hand side of a production in the grammar. This is the second most common parsing error and occurs approximately about 9% of the time in that same experiment.

Def: A stackability error occurs when the relation between the top of the stack and the reduced left hand side is either a > or there is no relation.

A stackability error occurs when you have identified the left hand side of the handle. You go to push that left hand side onto the stack and discover that the relationship between the top of the stack and the left hand side is either a > or there is no relation. This type of simple precedence parsing errors occurs approximately 6% of the time in that same experiment.

Modify the Simple Precedence Parsing Algorithm above to include the detection of the three types of simple precedence parsing errors.
The simple precedence parse configuration for the above sentence is:

<table>
<thead>
<tr>
<th>Parse Sequence</th>
<th>Relations</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>((⊥, V*(V+V)⊥, -)</td>
<td>⊥ &lt; V</td>
<td>S</td>
</tr>
<tr>
<td>((⊥ V, *(V+V)⊥, -)</td>
<td>V &gt; *</td>
<td>R 7</td>
</tr>
<tr>
<td>((⊥ P, *(V+V)⊥, 7)</td>
<td>P &gt; *</td>
<td>R 5</td>
</tr>
<tr>
<td>((⊥ T, *(V+V)⊥, 5)</td>
<td>T = *</td>
<td>S</td>
</tr>
<tr>
<td>((⊥ T*(V, +V)⊥, -)</td>
<td>* &lt; (</td>
<td>S</td>
</tr>
<tr>
<td>((⊥ T*(P, +V)⊥, 7)</td>
<td>V &gt; +</td>
<td>R 7</td>
</tr>
<tr>
<td>((⊥ T*(T, +V)⊥, 5)</td>
<td>P &gt; +</td>
<td>R 5</td>
</tr>
<tr>
<td>((⊥ T*(T-, +V)⊥, 3)</td>
<td>T &gt; +</td>
<td>R 3</td>
</tr>
<tr>
<td>((⊥ T*(E, +V)⊥, 2)</td>
<td>T- &gt; +</td>
<td>R 2</td>
</tr>
<tr>
<td>((⊥ T*(E+, V)⊥, -)</td>
<td>E = +</td>
<td>S</td>
</tr>
<tr>
<td>((⊥ T*(E+V)⊥, -)</td>
<td>+ &lt; V</td>
<td>S</td>
</tr>
<tr>
<td>((⊥ T*(E+P)⊥, 7)</td>
<td>V &gt; )</td>
<td>R 7</td>
</tr>
<tr>
<td>((⊥ T*(E+T)⊥, 5)</td>
<td>P &gt; )</td>
<td>R 5</td>
</tr>
<tr>
<td>((⊥ T*(E+T-)⊥, 3)</td>
<td>T &gt; )</td>
<td>R 3</td>
</tr>
<tr>
<td>((⊥ T*(E-)⊥, 1)</td>
<td>T- &gt; )</td>
<td>R 1</td>
</tr>
<tr>
<td>((⊥ T*(E-)⊥, 8)</td>
<td>E &gt; )</td>
<td>R 8</td>
</tr>
<tr>
<td>((⊥ T*(E-)⊥, -)</td>
<td>E- = )</td>
<td>S</td>
</tr>
<tr>
<td>((⊥ T*P, ⊥, 6)</td>
<td>) &gt;</td>
<td>R 6</td>
</tr>
<tr>
<td>((⊥ T, ⊥, 4)</td>
<td>P &gt;</td>
<td>R 4</td>
</tr>
<tr>
<td>((⊥ T-, ⊥, 3)</td>
<td>T &gt;</td>
<td>R 3</td>
</tr>
<tr>
<td>((⊥ E, ⊥, 2)</td>
<td>T- &gt;</td>
<td>R 2</td>
</tr>
</tbody>
</table>

The reduction sequence (reverse rightmost derivation sequence) for the above parse is:
7, 5, 7, 5, 3, 2, 7, 5, 3, 1, 8, 6, 4, 3, 2