2. \[ S \rightarrow bS \mid aT \]
\[ T \rightarrow \varepsilon \mid bT \mid aS \]
Start state: S

3. No there is no type T. A typing attempt fails in the rule for conditionals, since the two branches have types ref int and int, respectively.

\[
\begin{align*}
\Gamma', x : \text{int} & \vdash \Gamma''', x : \text{int} \\
\Gamma''', x + 1 : \text{int} & \vdash \Gamma''', \text{alloc}(x + 1) : \text{ref int} \\
\Gamma''', \text{alloc}(x + 1) & \vdash \Gamma''', x : \text{int}
\end{align*}
\]
where \( \Gamma''' = \Gamma[x \mapsto \text{int}] \).

The rule for conditionals requires the two branches to yield the same type.

4. An example grammar is
\[ S \rightarrow S + S \mid S - S \mid a \]
Start symbol S
String \( a + a - a \) has two parse trees

[Diagram of parse trees]
Problem 5.

1. \( S' \rightarrow S \$
2. \( S \rightarrow Y \)
3. \( Y \rightarrow Ya \)

- Need to prove this is SLR grammar, but isn't LR(1) and LL(K) grammar.

For LR(1):

1. \( S' \rightarrow S.S \)
2. \( S \rightarrow Y \)
3. \( Y \rightarrow Ya \)

<table>
<thead>
<tr>
<th>A</th>
<th>$</th>
<th>S</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$</td>
<td>g_2</td>
<td>g_3</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>g_2</td>
<td>g_3</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>g_2</td>
<td>g_3</td>
</tr>
<tr>
<td>4</td>
<td>a</td>
<td>g_2</td>
<td>g_3</td>
</tr>
</tbody>
</table>

There is a duplicated entry, so it is not in LR(1).

With SLR, \( a \notin \text{Follow}(S) \), then

<table>
<thead>
<tr>
<th>A</th>
<th>$</th>
<th>S</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$</td>
<td>g_2</td>
<td>g_3</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>g_2</td>
<td>g_3</td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>g_2</td>
<td>g_3</td>
</tr>
<tr>
<td>4</td>
<td>a</td>
<td>g_2</td>
<td>g_3</td>
</tr>
</tbody>
</table>

No duplication, thus it is in SLR.

Finally, this grammar has left recursion, that cannot be LL(K).

Problem 6

- Need to prove it is LL(1), i.e. no look ahead.

\( S' \rightarrow a.\$

Two cases: input a, accept

Others: reject \( \Rightarrow \) no look ahead