1 Augmented Grammars

Given a context free grammar \( G = (N, T, P, S) \) we augment this grammar as:
\[
G' = \{ N \cup \{ S' \}, T \cup \{ \vdash, \dashv \}, P \cup \{ S' \rightarrow \vdash S \dashv \}, S' \}
\]

2 Top Down parsing

2.1 Informal description

\( S' \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow w \)

- We start at the starting symbol and find a rule to apply
- This gives us \( \alpha_1 \)
  1. We scan \( \alpha_1 \) left to right
  2. While we encounter terminals during this scan, we match the terminals to the input string
  3. We stop when we encounter the first non-terminal (note: this is the leftmost non-terminal)
  4. We find a rule for this non-terminal and apply it i.e., replace the LHS with its RHS
- Repeat steps 1, 2, 3 and 4 until we have no non-terminals remaining in \( \alpha_i \)

2.2 Efficiency

- The informal algorithm discussed above is inefficient because after applying each rule we scan \( \alpha_i \) again
- Formally, \( \alpha_i = xA\beta \) (remember as per convention \( x \) is a string of terminals, \( A \) is one non-terminal and \( \beta \) is a string of terminals and non-terminals)
- We apply steps 1 and 2 on this \( \alpha_i \) EVEN IF we had matched a prefix of \( x \) when we got \( \alpha_{i-1} \)

IDEA:
- Match a terminal from \( \alpha_i \) to input string and if it matches discard it from \( \alpha_i \)
- Formally: if \( \alpha_i = xA\beta \) and \( w \) (input) = xyz (remember each of \( x, y \) and \( z \) represent strings of terminals)
- Then once we have matched \( x \) from \( \alpha_i \) to the prefix \( x \) from input, we only maintain \( A\beta \)
- We use a stack to store \( A\beta \) with \( A \) on top (so that we can access it quickly rather than having to find the leftmost non-terminal by traversing the stack)

2.3 Informal description of efficient algorithm

- Place starting symbol on the stack
- while Top of Stack (TOS) is a terminal, match it to input string symbol, pop it of the stack,
- once TOS is a non-terminal, FIND a rule to apply
- Pop the non-terminal, Push the RHS of found rule in reverse
- Algorithm terminates once we have consumed all input and stack is empty
- Only thing left is how to decide which rule to apply
2.4 Predictor Tables

- For a grammar that contains multiple rules for a given non-terminal, there is a choice of which rule to apply.
- We want to choose a rule that will likely lead to a successful derivation
- We can use the next input symbol (first symbol from the so far unmatched input) to choose a rule

Given a non-terminal (A) and the next unmatched input symbol (a), the predictor table specifies which rule to apply.

One can think of the Predict table as:

- A function, Predict(A,a)
- A 2D lookup table, Predict[A][a]

A grammar is LL(1) if \(| Predict[A][a] |\) for all A and a is at most 1 i.e. there is at most one applicable rule

We will look at how the Predict table is computed later.

3 LL(1) example using the efficient algorithm and pre-computed predict table

\[ S' \rightarrow S \rightarrow \]  
\[ S \rightarrow AyB \]  
\[ A \rightarrow ab \]  
\[ A \rightarrow cd \]  
\[ B \rightarrow z \]  
\[ B \rightarrow wx \]  

INPUT to parse: \( w = \rightarrow a \ b \ y \ w \ x \rightarrow \)

3.1 Top Down Parsing Invariant

- When we optimized the algorithm we chose to discard x from \( \alpha \beta \) once we had matched it to the input
- Let’s call x the consumed input (see example)
- Top Down Parsing always maintains the invariant: consumed input + stack contents (top to bottom) = \( \alpha_i \)
- Also remember that \( \alpha_i \) implies that \( S \Rightarrow^* \alpha_i \)

3.2 Errors

Assuming that we have a valid Predict table i.e. \( Predict[A][a] \) is at most 1 for any A and a, an error can occur in two places

- \( Predict[A][a] \) has NO rule for the non-terminal A and symbol a
- TOS is a terminal but it is NOT the next symbol in input

Both cases indicate that a derivation is NOT possible
<table>
<thead>
<tr>
<th>Consumed Input</th>
<th>Remaining Input</th>
<th>Stack</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-</td>
<td>abywx -l</td>
<td>S -l</td>
<td>While TOS is a terminal&lt;br&gt;Pop terminal&lt;br&gt;Match with input char&lt;br&gt;match l-</td>
</tr>
<tr>
<td>I-</td>
<td>abywx -l</td>
<td>A y B -l</td>
<td>TOS is a non-terminal&lt;br&gt;Pop S&lt;br&gt;Predict[S][a]= S -&gt; AyB&lt;br&gt;Push RHS in reverse</td>
</tr>
<tr>
<td>I-</td>
<td>abywx -l</td>
<td>a b y B -l</td>
<td>While TOS is terminal&lt;br&gt;Pop terminal&lt;br&gt;match with input char&lt;br&gt;Match a</td>
</tr>
<tr>
<td>I-</td>
<td>abywx -l</td>
<td>a b y B -l</td>
<td>Pop terminal&lt;br&gt;Match b</td>
</tr>
<tr>
<td>I-</td>
<td>abywx -l</td>
<td>a b y B -l</td>
<td>Pop terminal&lt;br&gt;Match y</td>
</tr>
<tr>
<td>I-</td>
<td>abywx -l</td>
<td>a b y B -l</td>
<td>Pop terminal&lt;br&gt;Match w</td>
</tr>
<tr>
<td>I-</td>
<td>abywx -l</td>
<td>x -l</td>
<td>Pop terminal&lt;br&gt;Match x</td>
</tr>
<tr>
<td>I-</td>
<td>abywx -l</td>
<td>-l</td>
<td>Pop terminal&lt;br&gt;Match -l</td>
</tr>
<tr>
<td>I-</td>
<td>abywx -l</td>
<td>ε</td>
<td>Input Consumed,&lt;br&gt;Stack empty, ACCEPT</td>
</tr>
</tbody>
</table>
4 Pseudo-code for Top Down Parsing

Input: string \( \rightarrow \) input \( \rightarrow \)
Input: Grammar \( G = N, T, P, S' \)

\[
\begin{align*}
\text{push } & \rightarrow \\
\text{push } & S \\
\text{push } & \vdash \\
& \text{for each ‘a’ in } \vdash \text{input } \rightarrow \text{left to right} \\
& \quad \text{while top of stack is } A \in N \\
& \quad \quad \text{pop } A \\
& \quad \quad \quad \text{if Predict}[A][a] \text{ gives } A \rightarrow \gamma \text{ then} \\
& \quad \quad \quad \quad \quad \text{push the symbols in } \gamma \text{ (right to left)} \\
& \quad \quad \quad \text{else} \\
& \quad \quad \quad \quad \quad \text{reject} \\
& \quad \quad \text{// top of stack is a terminal} \\
& \quad \quad \quad \text{reject if top of stack is not ‘a’} \\
& \quad \quad \quad \text{pop ‘a’} \\
& \text{end for} \\
& \text{accept (stack is necessarily empty)}
\end{align*}
\]

5 The Top Down Parsing Predict Set

Predict\([A][a]\) - rules that apply when \( A \) is on the stack and \( a \) is next input char

- \( \text{Predict}[A][a] = \{ A \rightarrow \beta \mid a \in \text{First}(\beta) \} \cup \{ A \rightarrow \beta \mid \text{Nullable}(\beta) \text{ and } a \in \text{Follow}(A) \} \)
- \( \text{First}(\beta) = \{ a \mid \beta \Rightarrow^* a\gamma \} \)
- \( \text{Nullable}(\beta) = \text{true if } \beta \Rightarrow^* \epsilon, \text{False otherwise} \)
- \( \text{Follow}(A) = \{ b \mid S' \Rightarrow^* \alpha Ab\beta \text{ for some } \alpha, \beta \} \)

5.1 Optional Material: Computing Nullable

\( \text{Nullable}(\beta) = \text{true if } \beta \Rightarrow^* \epsilon, \text{False otherwise} \)

\[
\begin{align*}
\text{initialize } & \text{Nullable}[A] = \text{false } \forall \ A \\
\text{repeat} \\
& \text{for each rule } B \rightarrow B_1\ldots B_k \\
& \quad \text{if } k = 0 \text{ or Nullable}[B_1] = \ldots = \text{Nullable}[B_k] = \text{true} \\
& \quad \quad \text{Nullable}[B] = \text{true} \\
\text{until nothing changes}
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th>Iteration 0</th>
<th>Iteration 1</th>
<th>Iteration 2</th>
<th>Iteration 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>S’</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>S</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>C</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
5.2 Optional Material: Computing First

\[ \text{First}(\beta) = \{a \mid \beta \Rightarrow^* a\gamma \} \]

**initialize** \( \text{First}[A] = \{ \} \ \forall \ A \)

repeat
  for each rule \( A \rightarrow B_1 \ldots B_k \)
    for \( i = 1 \ldots k \)
      if \( B_i \) is a terminal \( a \)
        \( \text{First}[A] \cup= \{a\}; \) break
      else
        \( \text{First}[A] \cup= \text{First}[B_i] \)
        if not Nullable\([B_i]\); break
  until nothing changes

<table>
<thead>
<tr>
<th>Iteration</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S' )</td>
<td>{}</td>
<td>{-}</td>
<td>{-}</td>
<td>{-}</td>
</tr>
<tr>
<td>( S )</td>
<td>{}</td>
<td>{b,p}</td>
<td>{b,p,l}</td>
<td>{b,p,l}</td>
</tr>
<tr>
<td>( C )</td>
<td>{}</td>
<td>{l}</td>
<td>{l}</td>
<td>{l}</td>
</tr>
</tbody>
</table>

5.3 Optional Material: Computing First*\((\beta)\) - first set for a string of symbols

\[ \text{First}^*(Y_1, \ldots, Y_n) \]

result = \( \emptyset \)

for \( i = 1 \ldots n \)
  if \( Y_i \notin T \) (i.e., \( Y_i \in N \))
    result \( \cup= \text{First}(Y_i) \)
  else (i.e., \( Y_i \in \Sigma \))
    result \( \cup= \{Y_i\} \)
    break (terminals are not nullable)
return result

5.4 Optional Material: Computing Follow

\[ \text{Follow}(A) = \{b \mid S' \Rightarrow^* aAb\beta \text{ for some } a,\beta \} \]

**initialize** \( \text{Follow}[A] = \emptyset \ \forall \ A \neq S' \) (don’t compute follow \( S' \))

repeat
  for each rule \( A \rightarrow B_1, \ldots, B_n \)
    for \( i = 1 \ldots n \)
      if \( B_i \in N \) (only do Follow sets for non-terminals)
        \( \text{Follow}[B_i] \cup= \text{First}^*(B_{i+1}, \ldots, B_n) \)
      if all of \( B_{i+1}, \ldots, B_n \) are nullable (including \( i == n \))
        \( \text{Follow}[B_i] \cup= \text{Follow}[A] \)
  until nothing changes

<table>
<thead>
<tr>
<th>Iteration</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S )</td>
<td>{}</td>
<td>{,&quot;d,q}</td>
<td>{,&quot;d,q}</td>
</tr>
<tr>
<td>( C )</td>
<td>{}</td>
<td>{,&quot;d,q}</td>
<td>{,&quot;d,q}</td>
</tr>
</tbody>
</table>
5.5 NOT OPTIONAL MATERIAL: Computing Predict

Given tables for Nullable, First and Follow, students must be able to compute the Predict Table.

\[
\text{Predict}(A, a) = \{ A \rightarrow \beta \mid a \in \text{First}(\beta) \} \cup \{ A \rightarrow \beta \mid \text{Nullable}(\beta), a \in \text{Follow}(A) \}
\]

Example: Predict[S][b] = ???

Look at all the rules for S one at a time

<table>
<thead>
<tr>
<th>Rule</th>
<th>Production</th>
<th>First(b S d)</th>
<th>Nullable(b S d)</th>
<th>Follow(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>S b S d</td>
<td>{b}</td>
<td>True</td>
<td>{b}</td>
</tr>
<tr>
<td>3</td>
<td>S p S q</td>
<td>{p}</td>
<td>False</td>
<td>{b}</td>
</tr>
<tr>
<td>4</td>
<td>S C</td>
<td>{l}</td>
<td>True</td>
<td>{b}</td>
</tr>
</tbody>
</table>

For rule 2, \(\beta\) is b S d, \(\text{first}(b S d)\) is \(\{b\}\).
Since the lookahead is b, this means rule 2 should be in the Predict[S][b]

For rule 3, \(\beta\) is p S q, \(\text{first}(p S q)\) is \(\{p\}\).
Since the lookahead is b, b \(\notin \{p\}\), so the first condition for being in the predict set does not match.

Let’s look at the second condition: \(\beta\) Nullable. \(\beta\) is p S q which cannot possibly be Nullable since it starts with a terminal. So the second condition to be in predict is also not true. Therefore rule 3 is NOT in Predict[S][b]

For Rule 4, \(\beta\) is C. First(C) is \{l\}. So since b \(\notin \{l\}\), the first condition for being in Predict does not match.
Is \(\beta\) Nullable. Yes it is. Is b in Follow(S). No it is not. So the second condition is also not met

<table>
<thead>
<tr>
<th></th>
<th>b</th>
<th>d</th>
<th>p</th>
<th>q</th>
<th>l</th>
</tr>
</thead>
<tbody>
<tr>
<td>S’</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>

Once we have the Predict table, we can see that this grammar is LL(1) since for no combination of non-terminal and terminal do we ever have more than 1 rule.