CS480
Translators

Introduction to Parsing
Chap. 2

Revisit SDT

• Syntax-directed translation
  — Attribute grammars/SDD
    • Synthesized
    • Inherited
    • Ex. \( E \rightarrow E_1 + T \quad E.s=E_1.s|T.s|'+' \)
  — Translation Scheme/Semantic Actions
    • Compliment SDD (allow SDD to contain code fragments)
    • Ex. \( E \rightarrow E_1 + T \ {\text{print}} ' + ' \)
  — During/After parsing

What is Parsing?

• Determine how string is generated
• Methods
  — Top-down
  — Bottom-up
Top-down Parsing

- Recursive Descent
  - What is it?
  - What do you need?
  - Issues?
- Predictive Parsing
Predictive Parsing

- Relies on:
  - $\alpha$ is string in grammar
  - $\text{FIRST}(\alpha)$ is set of terminals that appear first
  - If $\alpha$ generates $\varepsilon$, then $\varepsilon$ is in $\text{FIRST}(\alpha)$
  - If $A \rightarrow \alpha | \beta$, then $\text{FIRST}(\alpha)$ and $\text{FIRST}(\beta)$ disjoint
- Example:
  - $\text{FIRST}(\text{stmt}) = \{\text{expr, if, for, other}\}$
  - $\text{FIRST}(\text{expr };) = \{\text{expr}\}$

Looping Forever

- Left Recursion
- Rewrite:
  - $\text{expr} \rightarrow \text{expr} + \text{term} | \text{term}$
  - $A \rightarrow A\alpha | \beta$
  - $A \rightarrow \beta R$
  - $R \rightarrow aR | \varepsilon$
- What is the $A$, $\alpha$, and $\beta$?
Infix to Postfix/What’s wrong?

\[
\begin{align*}
expr & \rightarrow expr + term \ {\small \{ \ print(\star) \ \}} \\
& \ | \ expr - term \ {\small \{ \ print(-\star) \ \}} \\
& \ | \ term
\end{align*}
\]

\[
\begin{align*}
term & \rightarrow 0 \ {\small \{ \ print(0) \ \}} \\
& \ | \ 1 \ {\small \{ \ print(1) \ \}} \\
& \ | \ \ldots \\
& \ | \ 9 \ {\small \{ \ print(9) \ \}}
\end{align*}
\]

Figure 2.21: Actions for translating into postfix notation

How is this different/same?

\[
\begin{align*}
expr & \rightarrow \ term \ rest \\
rest & \rightarrow + \ term \ {\small \{ \ print(\star) \ \}} \ rest \\
& \ | \ - \ term \ {\small \{ \ print(-\star) \ \}} \ rest \\
& \ | \ \epsilon
\end{align*}
\]

\[
\begin{align*}
term & \rightarrow 0 \ {\small \{ \ print(0) \ \}} \\
& \ | \ 1 \ {\small \{ \ print(1) \ \}} \\
& \ | \ \ldots \\
& \ | \ 9 \ {\small \{ \ print(9) \ \}}
\end{align*}
\]

Figure 2.23: Translation scheme after left-recursion elimination

```c
void expr() {
    term(); rest();
}

void rest() {  
    if (lookahead == \star) {  
        match(\star); term(); rest(); 
    }  
    else if (lookahead == -\star) {  
        match(-\star); term(); rest(); 
    }  
    else { \small /* do nothing with the input */ \}
}

void term() {  
    if (lookahead is a digit) {  
        t = lookahead; match(lookahead); print(t); 
    }  
    else {report("syntax error"); 
}
```

Figure 2.25: Pseudocode for nonterminals \textit{expr}, \textit{rest}, and \textit{term}. 
Tail Recursion/While loop

```c
void rest() {
    while (true) {
        if (lookahead == '+') {
            match('+'); form('+'); print('+'); continue;
        } else if (lookahead == '-') {
            match('-'); form('-'); print('-'); continue;
        }
        break;
    }
}
```

Figure 2.26: Eliminating tail recursion in the procedure rest of Fig. 2.25.

---

Quiz #2

**Question 1**
- Remove the ambiguity from $S \rightarrow S (S) S | e$
- Derive $\{ \{ ( \{ ) \} \} \} \} \}$ w/ the unambiguous grammar.

**Question 2**
- What is the language of the following CFG:
  $S \rightarrow bSb | A$
  $A \rightarrow aA | e$
- Provide the parse tree for bbaaabb.

**Question 3**
- Provide the abstract syntax tree for the following:
  $-1 + 2 * 3.0^2 + 6$
- What is the post-order traversal of the tree.
- Explain how you would implement this in gforth.