Revisit SDT

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

Location of semantic action

$$rest \rightarrow + \ term \ \{ \text{print(' + ')} \} \ rest_1$$

Figure 2.13: An extra leaf is constructed for a semantic action

- What does this mean in terms of parsing?
What is Parsing?

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

for (; expr; expr) other

\[
\begin{align*}
slnt & \rightarrow \text{expr} ; \\
     & \quad | \text{if ( expr ) stmt} \\
     & \quad | \text{for ( optexpr ; optexpr ; optexpr ) stmt} \\
     & \quad | \text{other} \\
optexpr & \rightarrow \epsilon \\
     & | \text{expr}
\end{align*}
\]

Figure 2.16: A grammar for some statements in C and Java

\[
\begin{align*}
\text{for } \ ( \text{optexpr} ; \text{optexpr} ; \text{optexpr} ; \text{optexpr} ) \ \text{stmt} \\
\end{align*}
\]

Figure 2.17: A parse tree according to the grammar in Fig. 2.16

Top-down Parsing

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

• Relies on:
  – \( \alpha \) is string in grammar
  – FIRST(\( \alpha \)) is set of terminals that appear first
  – If \( \alpha \) generates \( \epsilon \), then \( \epsilon \) is in FIRST(\( \alpha \))
  – If \( A \rightarrow \alpha | \beta \), then FIRST(\( \alpha \)) and FIRST(\( \beta \)) disjoint

• Example:
  – FIRST(stmt) = \{ expr, if, for, other \}
  – FIRST(expr ;) = \{ expr \}
Looping Forever

- Left Recursion
- Rewrite:
  - \( expr -> expr + term | term \)
  - \( A -> Aα | β \)
  
  Vs.
  - \( A -> βR \)
  - \( R -> aR | ε \)

- What is the \( A, α, \) and \( β \)?

Infix to Postfix/What’s wrong?

\[
\begin{align*}
expr & \rightarrow \text{expr } \text{term} \\
& \quad \quad \{ \text{print}(\ast) \} \\
& \quad | \text{expr } \text{term} \\
& \quad \quad \{ \text{print}(\cdot) \} \\
& \quad | \text{term} \\
\text{term} & \rightarrow 0 \quad \{ \text{print}(0) \} \\
& \quad | 1 \quad \{ \text{print}(1) \} \\
& \quad | \ldots \\
& \quad | 9 \quad \{ \text{print}(9) \}
\end{align*}
\]

Figure 2.21: Actions for translating into postfix notation

How is this different/same?

\[
\begin{align*}
expr & \rightarrow \text{term } \text{rest} \\
\text{rest} & \rightarrow + \text{term} \quad \{ \text{print}(\ast) \} \quad \text{rest} \\
& \quad | - \text{term} \quad \{ \text{print}(\cdot) \} \quad \text{rest} \\
& \quad | \epsilon \\
\text{term} & \rightarrow 0 \quad \{ \text{print}(0) \} \\
& \quad | 1 \quad \{ \text{print}(1) \} \\
& \quad | \ldots \\
& \quad | 9 \quad \{ \text{print}(9) \}
\end{align*}
\]

Figure 2.23: Translation scheme after left-recursion elimination
Tail Recursion/While loop

```c
void rest() {
    while (true) {
        if ( lookahead == "x" ) {
            match("x"); term(); print("x");
        } else if ( lookahead == "-" ) {
            match("-"); term(); print("-");
        } else { /* do nothing with the input * */
        }
    }
}
```

Figure 2.26: Eliminating tail recursion in the procedure rest of Fig. 2.25.
Quiz #2

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

Question 2
• What is the language of the following CFG:
  $S \rightarrow bSb | A$
  $A \rightarrow aA | \varepsilon$
• Provide the parse tree for $bbbaabbb$.

Question 3
• Provide the abstract syntax tree for the following:
  $-1 + 2 * 3^{0.7} / 6$
• What is the post-order traversal of the tree.
• Explain how you would implement this in gforth.