CS480
Translators

Introduction to Parsing
Finish Chap. 2
Revisit SDT

• Syntax-directed translation
  – Translation Scheme/Semantic Actions
    • Ex. $E \rightarrow E_1 + T \quad \{ \text{print} \, ‘+’ \}$
  – During/After parsing

• When you see 10.0, do you print this in gforth?

\[ \text{print (10.0)} \]
What is Parsing?

- Determine how string is generated
- Methods
  - Top-down
  - Bottom-up
for (; expr ; expr ) other

stmt → expr ;
| if ( expr ) stmt
| for ( optexpr ; optexpr ; optexpr ) stmt
| other

optexpr → ε
| expr

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

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
  table-driven
Figure 2.18: Top-down parsing while scanning the input from left to right
void stmt() {
  switch (lookahead) {
    case expr:
      match(expr); match(';'); break;
    case if:
      match(if); match('('); match(expr); match(')'); stmt();
      break;
    case for:
      match(for); match('(');
      optexpr(); match('); optexpr(); match('); optexpr();
      match(')'); stmt(); break;
    case other:
      match(other); break;
    default:
      report("syntax error");
  }
}

void optexpr() {
  if (lookahead == expr) match(expr);
}

void match(terminal t) {
  if (lookahead == t) lookahead = nextTerminal;
  else report("syntax error");
}
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 $A \rightarrow \alpha \mid \beta$, then $\text{FIRST}(\alpha)$ and $\text{FIRST}(\beta)$ disjoint

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

• Left Recursion

• Rewrite:
  – expr -> expr + term | term
  – A -> Aα | β

Vs.
  – A -> βR
  – R -> αR | ε

• What is the A, α, and β?

need to make left/head recursive

new production
Infix to Postfix/What’s wrong?

\[ \text{expr} \rightarrow \text{expr} + \text{term} \quad \{ \text{print('+'}) \} \\
\left| \quad \text{expr} - \text{term} \quad \{ \text{print('-')} \} \\
\left| \quad \text{term} \quad \right. \]

\[ \text{term} \rightarrow 0 \quad \{ \text{print('0')} \} \\
\left| \quad 1 \quad \{ \text{print('1')} \} \\
\left| \quad \ldots \quad \right. \]

\[ \left| \quad 9 \quad \{ \text{print('9')} \} \right. \]

Figure 2.21: Actions for translating into postfix notation
How is this different/same?

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

Figure 2.23: Translation scheme after left-recursion elimination
void expr() {
    term(); rest();
}

void rest() {
    if (lookahead == '+') {
        match('+'); term(); print('+'); rest();
    } else if (lookahead == '-') {
        match('-'); term(); print('-'); rest();
    } else { /* do nothing with the input */ }
}

void term() {
    if (lookahead is a digit) {
        \texttt{t} = \texttt{lookahead}; match(lookahead); print(t);
    } else report("syntax error");
}
Milestone II - Lexical Analysis

• What is its purpose?
• What is the difference in a token vs. lexeme?
• Example:
  – The Brown Fox
  – if (i != 32) then j := 12
• Are spaces important?
Quiz #2

Question 1
• What is the language of the following CFG:
  \[ S \rightarrow bSbb \mid A \]
  \[ A \rightarrow aA \mid \varepsilon \]
• Provide the parse tree for bbaaabbbb.

Question 2
• Provide the abstract syntax tree for the following:
  \[-1 + 2 * 3.0^{4.7} / 6\]
• What is the post-order traversal of the tree.
• Explain how you would implement this in gforth.
• How would the tree look if you embed the gforth?
The Role of the Scanner...

Figure 3.1: Interactions between the lexical analyzer and the parser
while (i > 0)
i = i - 2;

Lexical Analyzer

Source Language

Error Messages

Token Stream

T_WHILE
T_LPAREN
T_IDENTIFIER
T_LESSTHAN
T_INTCONSTANT
T_RPAREN
T_IDENTIFIER
T_EQUALS
T_MINUS
T_EQUALS
T_INTCONSTANT
T_SEMICOLON
What’s new in this grammar?

\[
\begin{align*}
expr & \rightarrow \ expr + \ term \quad \{ \ \text{print('+' )} \} \\
& \quad | \ expr - \ term \quad \{ \ \text{print('-') } \} \\
& \quad | \ term \\

term & \rightarrow \ term * \ factor \quad \{ \ \text{print('*') } \} \\
& \quad | \ term / \ factor \quad \{ \ \text{print('/') } \} \\
& \quad | \ factor \\

factor & \rightarrow \ ( \ expr ) \\
& \quad | \ num \quad \{ \ \text{print(num.value) } \} \\
& \quad | \ id \quad \{ \ \text{print(id.lexeme) } \}
\end{align*}
\]

Figure 2.28: Actions for translating into postfix notation