Parsing from Grammar

Syntax Directed Left Recursive Grammar

Syntax directed translation adds semantic rules to be carried out when syntactic rules are applied. Let's do conversion of infix to postfix.

Expr → Expr Plus Term	{out(" + ");}
| Term	
Term → Term Times Factor	{out(" * ");}
| Factor	
Factor → Lparen Expr Rparen	
| Int	{out(" ", Lex.value, " ");}

How It Works

Examples of applying previous syntax directed translation

Input: 15 + 20 + 7 * 3 + 2

Output: 15 20 + 7 3 * + 2 +

Input: 15 + 20 + 7 + 3 * 2

Output: 15 20 + 7 + 3 2 * +

Removing Left Recursion

Given left recursive and non left recursive rules

$A \rightarrow A\alpha_1 \mid \ldots \mid A\alpha_n \mid \beta_1 \mid \ldots \mid \beta_m$

Can view as

$A \rightarrow (\beta_1 \mid \ldots \mid \beta_m)\ (\alpha_1 \mid \ldots \mid \alpha_n)^*$

Star notation is an extension to normal notation with obvious meaning

Now, it should be clear this can be done right recursive as

$A \rightarrow \beta_1\ B \mid \ldots \mid \beta_m\ B$

$B \rightarrow \alpha_1 B \mid \ldots \mid \alpha_n B \mid \lambda$

Treat Actions from Left Rec as Terminals

Expr → Term ExprRest

ExprRest → Plus Term {out (“ + “);} ExprRest

| λ

Term → Factor TermRest

TermRest → Times Factor {out(“ * “);} TermRest

| λ

Factor → Lparen Expr Rparen

| Int {out(“ “,Lex.value,” “);}

Recursive Descent

```
Expr() {
    Term();
    ExprRest();
}

ExprRest() {
    if (token == Plus) {
        nextsy();
        Term();
        out(" + ");
        ExprRest();
    }
}
```

```
Term() {
    Factor();
    TermRest();
}

TermRest() {
    If (token == Times) {
        nextsy();
        Factor();
        out(" * ");
        TermRest();
    }
}
```

```
F() {
    switch (token) {
        case Lparen:
            nextsy();
            call E
            if (token == Rparen)
              nextsy();
            else
              ERROR();
            break;
        case Id:
            out( Lex.value );
            nextsy();
            break;
        default:
            ERROR();
    }
}
```

Process

- Write left recursive grammar with semantic actions.
- Rewrite a right recursive with actions treated as terminals in original rules.
- Develop recursive descent parser.