RECURSIVE DESCENT PARSER

Recursive Descent Parsing

In a recursive-descent parser, **the production information is embedded in the individual parse functions for each nonterminal and the run-time execution stack is keeping track of our progress through the parse.** There is another method for implementing a predictive parser that uses a table to store that production along with an explicit stack to keep track of where we are in the parse.

This grammar for add/multiply expressions is already set up to handle precedence and associativity:
E –> E + T | T
T –> T * F | F
F –> (E) | int

After removal of left recursion, we get:
E –> TE'
E' –> + TE' | ε
T –> FT'
T' –> * FT' | ε
F –> (E) | int

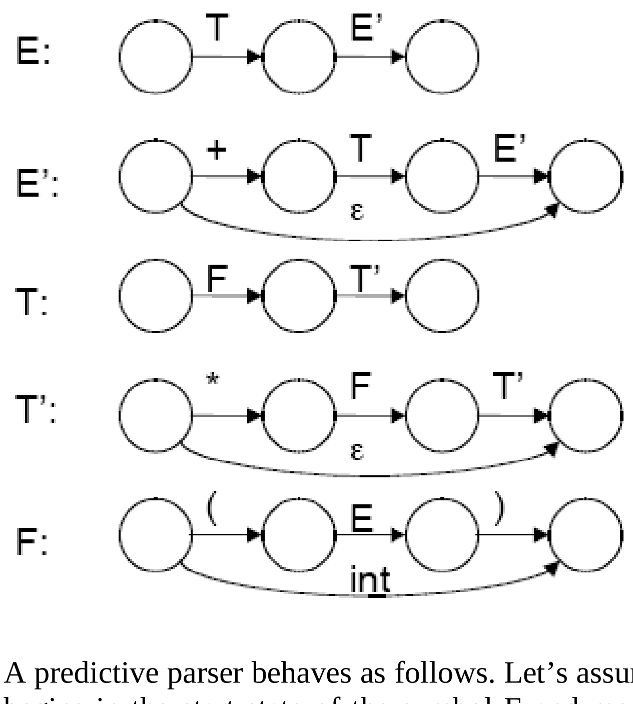
The following shows the mutual recursive procedures for all nonterminals to recognize arithmetic expression grammar.

```
procedure E()
begin
        T()
        EPRIME()
end

procedure EPRIME()
if input symbol=’+’ then
begin
        ADVANCE()
        T()
        EPRIME()
end

procedure T()
begin
        F()
        TPRIME()
end
```

```
procedure TPRIME()
if input-symbol=’*’ then
begin
    ADVANCE()
    F()
    TPRIME()
end
```

```
procedure F()
if input-symbol=’id’ then
    ADVANCE()
else if input-symbol=’(‘ then
begin
    ADVANCE()
    E()
    if input symbol=’)’ then
        ADVANCE()
    else ERROR()
end
else ERROR()
```

Another way to illustrate the process is to study some transition graphs that represent the grammar:



A predictive parser behaves as follows. Let’s assume the input string is **3 + 4 * 5.** Parsing begins in the start state of the symbol E and moves to the next state. This transition is marked with a T, which sends us to the start state for T. This in turn, sends us to the start state for F. F has only terminals, so we read a token from the input string. It must either be an open parenthesis or an integer in order for this parse to be valid. We consume the integer token, and thus we have hit a final state in the F transition diagram, so we return to where we came from which is the T diagram; we have just finished processing the F nonterminal.

We continue with T', and go to that start state. The current lookahead is + which doesn’t match the * required by the first production, but + is in the follow set for T' so we match the second production which allows T' to disappear entirely. We finish T' and return to T, where we are also in a final state. We return to the E diagram where we have just finished processing the T. We move on to E', and so on.