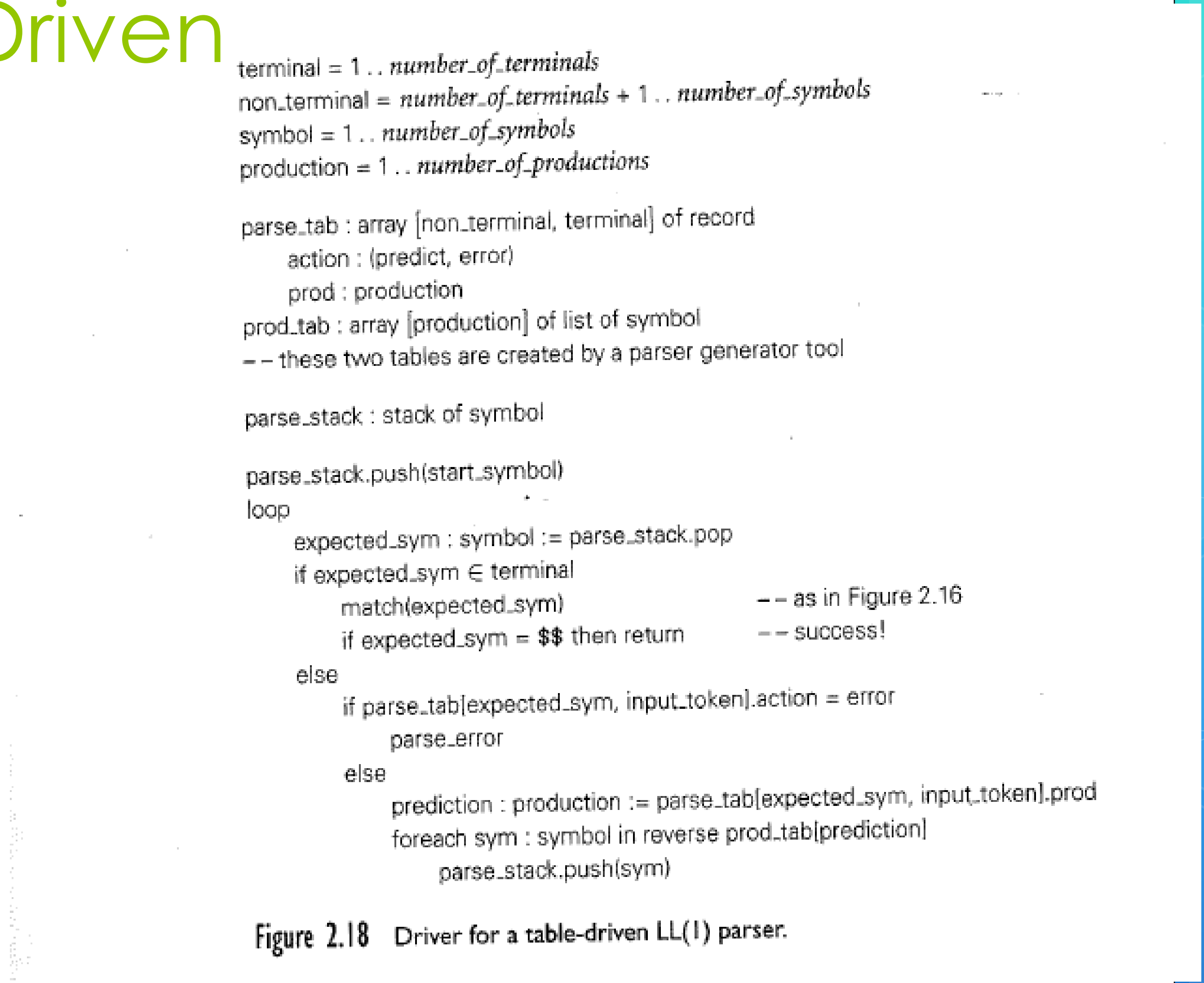
Recursive Descent Parsing
Table Driven



```
terminal = 1 . . number_of_terminals
non_terminal = number_of_terminals + 1 . . number_of_symbols
symbol = 1 . . number_of_symbols
production = 1 . . number_of_productions

parse_tab : array [non_terminal, terminal] of record
    action : (predict, error)
    prod : production
prod_tab : array [production] of list of symbol
-- these two tables are created by a parser generator tool

parse_stack : stack of symbol

parse_stack.push(start_symbol)
loop
    expected_sym : symbol := parse_stack.pop
    if expected_sym ∈ terminal
        match(expected_sym)                      -- as in Figure 2.16
        if expected_sym = $$ then return         -- success!
    else
        if parse_tab[expected_sym, input_token].action = error
            parse_error
        else
            prediction : production := parse_tab[expected_sym, input_token].prod
            foreach sym : symbol in reverse prod_tab[prediction]
                parse_stack.push(sym)
```

Figure 2.18 Driver for a table-driven LL(1) parser.