Chapter 7
Syntax Directed Translation

Lexical Structure --> Systematic algorithms exist --> Finite State Automata.

Syntax Structure --> Systematic algorithms exist --> Push Down Automata.

Semantic Structure --> Unfortunately, no systematic algorithm.

However, there is a framework for **intermediate code generation**, which is an extension of the context-free grammar called **syntax-directed translation**.

In syntax-directed translation, the algorithm allows what is called a **semantic action**, which is simply a subroutine(procedure/function) attached to some of the production rules of the context-free grammar.

A semantic action or a semantic rule is simply an output action added(associated) to the production rule of the grammar. For example, given the production

> A --> α

the semantic action is simply

> A --> α #B

where B is the semantic action.

Take the production

> A --> XYZ #α

assume that α is the semantic action/rule, then in the syntax-directed translation scheme, the semantic action α is called/executed whenever the parser recognizes or accepts a sentence w derived from A in top-down parsing.

> A --> XYZ -*-> w ∈ L(G)

In bottom-up parsers, the semantic action α is called whenever XYZ is reduced.

Generally, compilers generate/translate source code into another format which is easier for the compiler to understand(evaluate). This code is called **intermediate code**. There are different kinds of intermediate code :

1. Postfix Code : for example,

   `A + B * ( C - D ) + E`

   becomes

   `A B C D - * + E +`

   in postfix.

   Another example :

   ```
   if a
       x
   else
       y
   ```

   would become

   ``` a x y ? ```

   in postfix.

   Or another example :

   ```

```
      If a
        if(c - D)
          a + c
        else
          a * c
    else
        a + b
```

becomes

``` a c D - a c + a c * ? a b + ? ```

in postfix.

2. Three Address Code (TAC) : Each instruction has at most 3 components.

   for example :

   ```
   - w * x + ( y + z )
   ```

   would be

   ```
   (1) - , w
   (2) *,(1),x
   (3) +,y,z
   (4) +,(2),(3)
   ```

   in TAC.

   Another Example :

   ```
   if (x > y)
           z = x
   else
           z = y + 1
   ```

   would be

   ```
   (1) -,x,y
   (2) JGZ,(1),(6)
   (3) +,y,1
   (4) =,z,(3)
   (5) JMP,(7)
   (6) =,z,x
   (7) ......
   ```

   in TAC.

3. Quadruples : Another form of intermediate code, which has at most 4 components. for example : `- w * x + ( y + z )` would be :

operation	operand 1	operand 2	result
-	w	_	R1
*	R1	x	R2
+	y	z	R3
+	R2	R3	R4
   ```