Compiladores

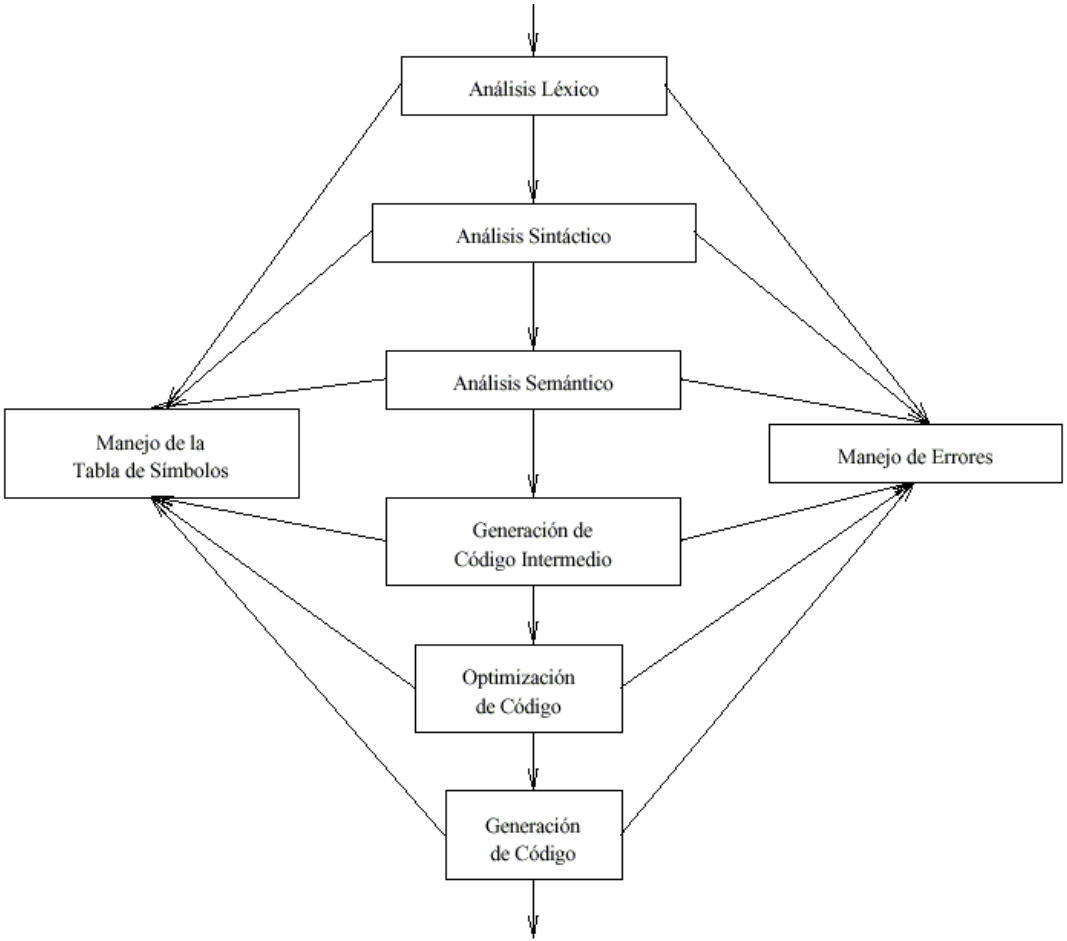
Ing José Jesús Ambriz Meza
jambriz@gmail.com
https://www.youtube.com/channel/UC0IAqjAAFAzxgcJC_YoiUQA
http://deprofesoramaestro.blogspot.com/

Índice

Etapas de un traductor

La tabla de símbolos inicia su construcción con el lexicográfico, pero se continúa enriqueciendo en las siguientes etapas.

El manejo de errores es el mismo caso.



Analizador léxico

Recibe el archivo que se quiere traducir y genera la secuencia de los tokens que lo integran. Un token es la unidad mínima con significado (es un terminal de la gramática).

El lexicográfico reconoce un token independientemente del contexto (no valida si el token está ubicado en el lugar que marca la gramática). Cada token comúnmente inicia con dos valores (puede tener más): valor y tipo.

(identificador, “u8numero”)

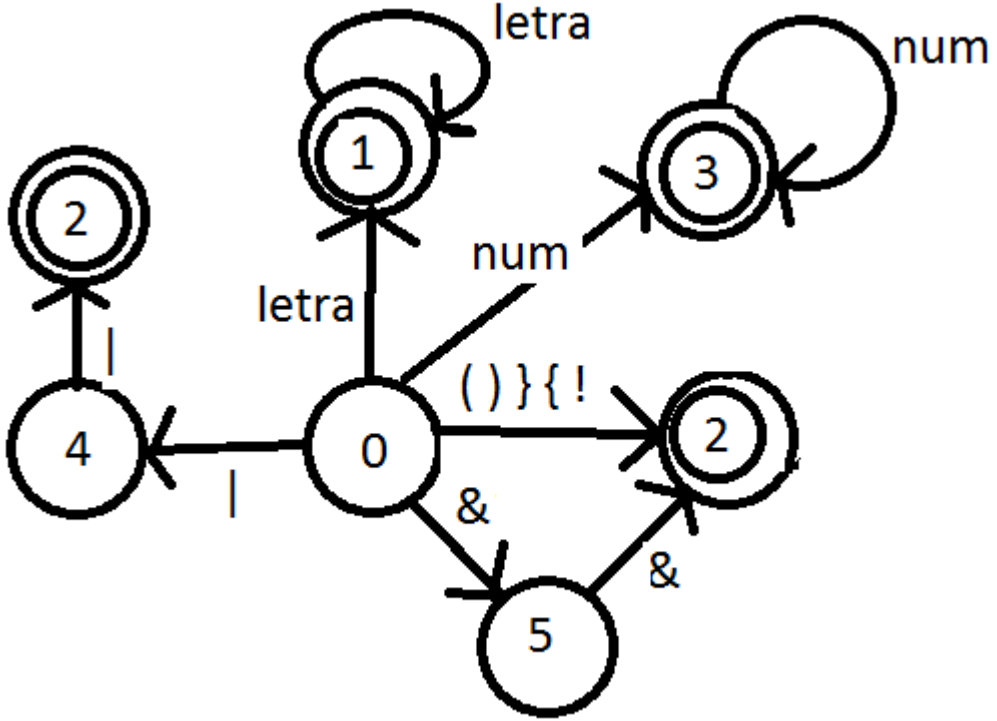
(simbolo, “+”)

Analizador léxico

Un lexicográfico puede implementarse por medio de un autómata finito determinístico.

El siguiente autómata reconoce los tokens de la gramática de karel vistos en la gramática dada para la clase.

Nota: 0 es el estado inicial



Analizador léxico

La implementación del autómata anterior puede realizarse con una tabla como la siguiente.

Una variable “estado” va cambiando su valor procesando uno a uno los símbolos del archivo de entrada.

status= Automata[status][simb];

simb= columna del símbolo

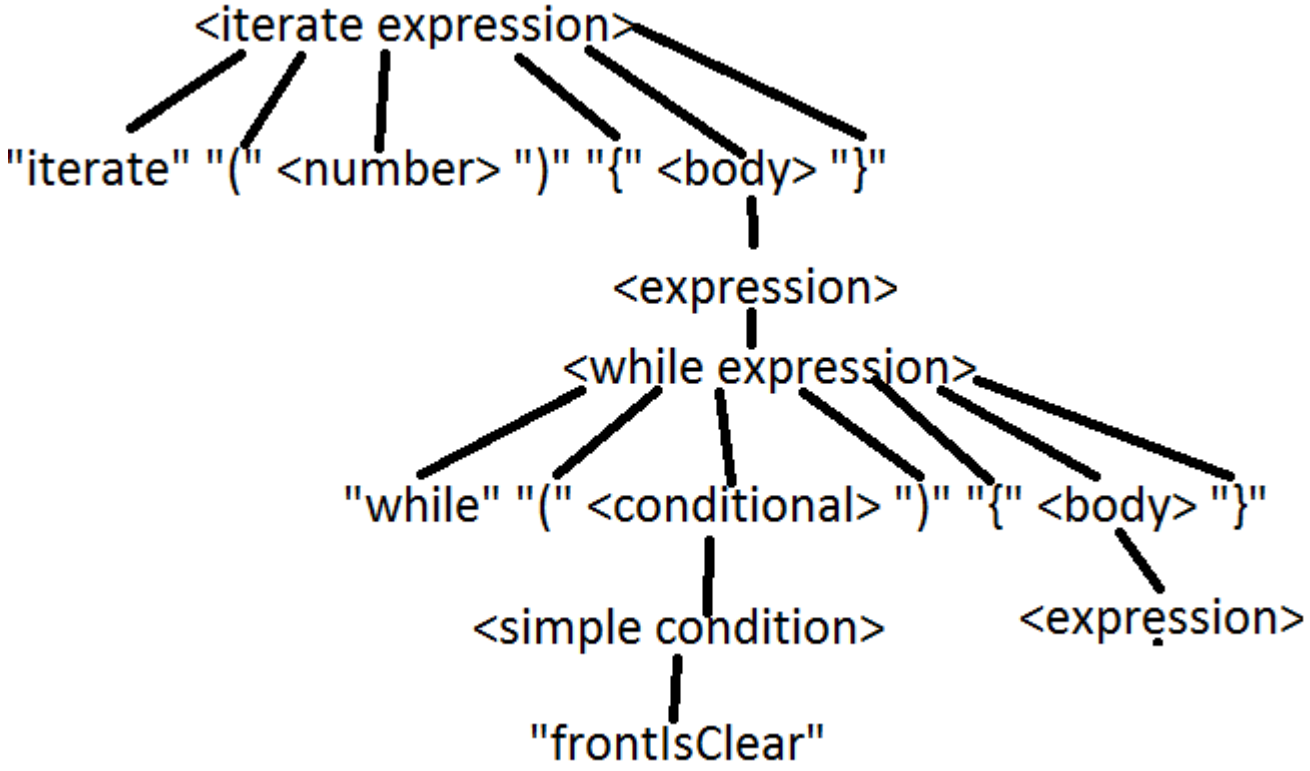
estados/ simbolos	letra	digito	({) } !	|	&	otro
0	1	3	2	4	5	6
1	1	6	6	6	6	6
2	6	6	6	6	6	6
3	6	3	6	6	6	6
4	6	6	6	2	6	6
5	6	6	6	6	2	6
6	6	6	6	6	6	6

Analizador sintáctico

Recibe la secuencia de tokens procesada por el lexicográfico, y genera una estructura en memoria que representa la gramática que procesa a ese lenguaje indicando los tokens leídos.

El sintáctico, usando la gramática, válida la sintaxis del archivo recibido. Para hacerlo genera un árbol sintáctico. Es una estructura en memoria que representa los elementos no terminales en la gramática usada.

Analizador sintáctico



Analizador sintáctico

Un árbol sintáctico se puede construir en base a dos estrategias:

- LL (de arriba hacia abajo, desde la izquierda)
- LR (de abajo hacia arriba, desde la derecha)

La primera procesa gramáticas más sencillas que la segunda.

Analizador sintáctico

LL (de arriba hacia abajo, desde la izquierda)

Una estrategia de implementación es generar un recursivo descendente. Consta de un software que tiene una función por cada no terminal.

El programa tiene la “misma” estructura que la gramática.

Analizador sintáctico

LL consta de 2 funciones básicas:

boolean = exigir(token) - devuelve true si el token que recibe como parámetro es el siguiente en la secuencia que recibe. Continua con el siguiente token

boolean = verifica(token) - lo mismo que la anterior PERO no pasa al siguiente token.

Analizador sintáctico

```
void iterate_expression() {
  if ( exigir("iterate") ) {
        if ( exigir("(") ) {
          if ( exigir( NUMBER ) {
            if ( exigir(")") ) {
              if ( exigir("{") ) {
                 body();
                 if ( !exigir("}") ) {   /*error */              }
              } else { /*error */ }
            } else { /*error */ }
          } else { /*error */ }
        } else { /*error */ }
  } else { /*error */ }
}
```

Analizador sintáctico

De recibir un terminal no esperado, el sintáctico tiene dos opciones:

a. Continuar
b. Romper la generación del árbol sintáctico.

De no poder continuar debido a la cantidad de errores debe de mostrar los lugares dentro del código y el error cometido. De no tener errores el árbol sintáctico, éste será usado después.

Referencias

[1] Aho, Sethi & Ullman. Compilers Principles, Techniques and Tools.

[2] Superhacker77. Compiladores. http://www.monografias.com/trabajos11/compil/compil.shtml