Analizador Sintáctico

Un analizador sintáctico (o parser) es un programa informático que analiza una cadena de símbolos de acuerdo a las reglas de una gramática formal. El término proviene del Latín pars, que significa parte (del discurso). Usualmente hace parte de un compilador, en cuyo caso, transforma una entrada en un árbol de sintáctico de derivación.1​2​  
  
El análisis sintáctico convierte el texto de entrada en otras estructuras (comúnmente árboles), que son más útiles para el posterior análisis y capturan la jerarquía implícita de la entrada. Un analizador léxico crea tokens de una secuencia de caracteres de entrada y son estos tokens los que son procesados por el analizador sintáctico para construir la estructura de datos, por ejemplo un árbol de análisis o árboles de sintaxis abstracta.  
  
El análisis sintáctico también es un estado inicial del análisis de frases de lenguaje natural. Es usado para generar diagramas de lenguajes que usan flexión gramatical, como los idiomas romances o el latín. Los lenguajes habitualmente reconocidos por los analizadores sintácticos son los lenguajes libres de contexto. Cabe notar que existe una justificación formal que establece que los lenguajes libres de contexto son aquellos reconocibles por un autómata de pila, de modo que todo analizador sintáctico que reconoce un lenguaje libre de contexto es equivalente en capacidad computacional a un autómata de pila.  
  
Los analizadores sintácticos fueron extensivamente estudiados durante los años 1970, detectando numerosos patrones de funcionamiento en ellos, cosa que permitió la creación de programas generadores de analizadores sintácticos a partir de una especificación de la sintaxis del lenguaje en forma Backus-Naur por ejemplo, tales como yacc, GNU bison y javaCC.

Lenguajes de Programación

El uso más común de los analizadores sintácticos es como parte de la fase de análisis de los compiladores. De modo que tienen que analizar el código fuente del lenguaje. Los lenguajes de programación tienden a basarse en gramáticas libres de contexto, debido a que se pueden escribir analizadores rápidos y eficientes para estas.  
  
Las gramáticas libres de contexto tienen una expresividad limitada y sólo pueden expresar un conjunto limitado de lenguajes. Informalmente la razón de esto es que la memoria de un lenguaje de este tipo es limitada, la gramática no puede recordar la presencia de una construcción en una entrada arbitrariamente larga y esto es necesario en un lenguaje en el que por ejemplo una variable debe ser declarada antes de que pueda ser referenciada. Las gramáticas más complejas no pueden ser analizadas de forma eficiente. Por estas razones es común crear un analizador permisivo para una gramática libre de contexto que acepta un superconjunto del lenguaje (acepta algunas construcciones inválidas), después del análisis inicial las construcciones incorrectas pueden ser filtradas.  
  
Normalmente es fácil definir una gramática libre de contexto que acepte todas las construcciones de un lenguaje pero por el contrario es prácticamente imposible construir una gramática libre de contexto que admita solo las construcciones deseadas. En cualquier caso la mayoría de analizadores no son construidos a mano sino usando generadores automáticos.

Idioma

Los sistemas más exitosos usan estadísticas léxicas, es decir obtienen la categoría gramatical de las palabras, estos sistemas son vulnerables debido a que terminan por tener una cantidad excesiva de parámetros y finalmente requieren simplificaciones.

Los algoritmos de análisis de idioma natural no se pueden basar en gramáticas que tengan unas buenas características como se hace con las gramáticas diseñadas, por ejemplo para los lenguajes de programación. Algunos formalismos gramaticales son muy difíciles de analizar computacionalmente, por lo que, en general se usa una aproximación libre de contexto incluso si la estructura en sí no es libre de contexto para obtener una primera simplificación.

Los algoritmos que usan gramáticas libres de contexto se suelen basar en alguna variante del algoritmo Cocke-Younger-Kasami (CYK) y heurística para la poda de análisis infructuosos. En todo caso algunos enfoques sacrifican la velocidad por la precisión usando, por ejemplo, versiones lineales del algoritmo «shift-reduce». Enfoques recientemente desarrollados utilizan un algoritmo que genera de múltiples análisis y otro que escoge la mejor opción.

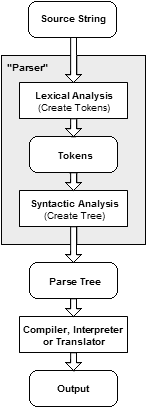
Clasificación

La tarea esencial de un analizador es determinar si una determinada entrada puede ser derivada desde el símbolo inicial, usando las reglas de una gramática formal, y como hacer esto, existen esencialmente dos formas:

* *Analizador sintáctico descendente (Top-Down-Parser):* ..un analizador puede empezar con el símbolo inicial e intentar transformarlo en la entrada, intuitivamente esto sería ir dividiendo la entrada progresivamente en partes cada vez más pequeñas, de esta forma funcionan los analizadores LL, un ejemplo es el javaCC. Una mejora en estos parsers se puede lograr usando GLR (Generalized Left-to-right Rightmost derivation).
* *Analizador sintáctico ascendente (Bottom-Up-Parser):* un analizador puede empezar con la entrada e intentar llegar hasta el símbolo inicial, intuitivamente el analizador intenta encontrar los símbolos más pequeños y progresivamente construir la jerarquía de símbolos hasta el inicial, los analizadores LR funcionan así y un ejemplo es el Yacc. También existen SLR (Simple LR) o los LALR (Look-ahead LR) como también de los GLL(Generalized Left-to-right Leftmost derivation).

Otros tipos de analizadores son:

* Analizador sintáctico descendente recursivo
* Chart parser
* Left corner parser
* Analizador sintáctico LR
* Analizador sintáctico LALR



Resumen del proceso

El siguiente ejemplo demuestra el caso común de analizar un lenguaje de computadora con dos niveles de gramática: léxico y sintáctico. La primera etapa es la generación de tokens, o análisis léxico, mediante la cual el flujo de caracteres de entrada se divide en símbolos significativos definidos por una gramática de expresiones regulares. Por ejemplo, un programa de calculadora vería una entrada tal como "12 \* (3 + 4) ^ 2" y dividirlo en los tokens 12, \*, (, 3, +, 4,), ^, 2, cada uno de que es un símbolo significativo en el contexto de una expresión aritmética.

El lexer contendrá reglas para decirle que los caracteres \*, +, ^, (y) marcan el comienzo de un nuevo token, por lo que no se generarán fichas sin sentido como "12 \*" o "3". análisis gramatical o análisis sintáctico, que consiste en comprobar que los tokens forman una expresión permisible, lo que suele hacerse con referencia a una gramática sin contexto que define recursivamente componentes que pueden formar una expresión y el orden en que deben aparecer. todas las reglas que definen los lenguajes de programación pueden ser expresadas sólo por gramáticas sin contexto, por ejemplo, la validez de tipo y la declaración adecuada de identificadores.Las reglas finales pueden ser formalmente expresadas con gramáticas de atributos.La fase final es el análisis semántico o análisis, de la expresión recién validada y tomando la acción apropiada

En el caso de una calculadora o intérprete, la acción es evaluar la expresión o programa, un compilador, por otra parte, generaría algún tipo de código Las gramáticas de atributo también se pueden usar para definir estas acciones.