Un analizador sintáctico (o *parser*) es una de las partes de un compilador que transforma su entrada en un árbol de derivación.

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 reconozca un lenguaje libre de contexto es equivalente en capacidad computacional a un autómata de pila.

Los analizadores sintácticos fueron extensivamente estudiados durante los años 70 del siglo XX, detectándose numerosos patrones de funcionamiento en ellos, cosa que permitió la creación de programas generadores de analizadores sintáticos a partir de una especificación de la sintaxis del lenguaje en forma Backus-Naur por ejemplo, tales como yacc,GNU bison y javaCC.

Algunos sistemas de traducción o procesamiento de idioma natural son léxicamente analizados por programas informáticos. Las frases no son fácilmente analizables debido a la carga de ambigüedad que existe en la estructura del idioma humano. Para procesar el idioma humano los investigadores deben antes ponerse de acuerdo en la gramática a utilizar y esta decisión está influenciada por criterios lingüísticos y computacionales, por ejemplo algunos sistemas de análisis usan gramáticas léxico-funcionales. Pero en general el análisis de gramáticas de este tipo es un NP-completo.

El «Head-driven phrase structure grammar» es otro formalismo que ha sido popular en la comunidad, pero los esfuerzos en investigación se han centrado en algoritmos menos complejos como el de Penn Treebank. El análisis ligero «Shallow parsing» se encarga sólo de encontrar los componentes principales de la frase como nombres o verbos. Otra estrategia popular para evitar la controversia lingüística es la gramática de dependencias.

La mayoría de los analizadores modernos son al menos en parte estadísticos, esto quiere decir que se basan en unos datos de entrenamiento que han sido analizados a mano. Este enfoque permite al sistema reunir información sobre la frecuencia con que ocurren ciertas construcciones en un contexto específico. Algunos de estos enfoques han incluido gramáticas libres de contexto probabilísticas, sistemas de máxima entropía y redes neuronales.

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.

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 éstas.

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.

Visión general del proceso

El siguiente caso demuestra un caso común de análisis de un lenguaje de programación con dos niveles de gramática, léxica y sintáctica.

El primer estado es la generación de tokens o análisis léxico, en este proceso la cadena de entrada se parte en símbolos con significado definidos por una gramática de expresiones regulares, por ejemplo un programa calculadora con la siguiente entrada: «12*(3+4)^2», la dividiría en los siguientes tokens 12, *, (, 3, +, 4,), ^ y 2, cada uno de estos símbolos tiene un significado en el contexto de la expresión aritmética. El analizador contendrá reglas para

indicar que los símbolos *, +, ^, (y) indican el comienzo de un nuevo token, de modo que otros tokens que no tendrían sentido como 12 o 13 no se generarán.

El siguiente estado es el análisis sintáctico lo que significa comprobar que los tokens forman una expresión válida, esto se hace usualmente usando una gramática libre de contexto que define recursivamente componentes que pueden aparecer en una expresión y el orden en que estos deben aparecer. Las reglas que definen un lenguaje de programación no siempre se pueden expresar usando únicamente una gramática libre de contexto, por ejemplo la validación de tipos y la declaración correcta de identificadores. Estas reglas pueden expresarse formalmente usando gramáticas de atributos.

La fase final es el análisis semántico, que trabaja en las implicaciones de la expresión ya validada y realiza las actuaciones pertinentes. En el caso de la calculadora, la acción es evaluar la expresión. Un compilador por el contrario generará código. Las gramáticas de atributos pueden ser usadas también para definir estas acciones.

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.
- 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.

Otros tipos de analizadores son:

- Analizador sintáctico descendente recursivo
- Chart parser
- Left corner parser
- Analizador sintáctico LR