ANÁLISIS LÉXICO

FUNCION

Leer los caracteres de entrada y elaborar como salida una secuencia de componentes léxicos que utiliza el analizador sintáctico para hacer el análisis.

FUNCION PRINCIPAL

El analizador léxico lee los caracteres de entrada y genera una secuencia de componentes léxicos para el análisis sintáctico. Generalmente, funciona como una subrutina del analizador sintáctico, respondiendo a sus solicitudes de nuevos componentes léxicos.

FUNCION SECUNDARIA

Realiza funciones secundarias como eliminar comentarios y espacios en blanco, asociar mensajes de error con el código fuente y, en algunos casos, generar una copia del programa con errores marcados. También puede manejar funciones de preprocesamiento de macros si el lenguaje lo permite.

COMPONENTES

Un token es una unidad léxica formada por un nombre y, opcionalmente, un valor de atributo. Representa elementos como palabras clave o identificadores y es procesado por el analizador sintáctico.

Un patrón describe la forma de los lexemas de un token. Para palabras clave, es una secuencia fija de caracteres, mientras que para identificadores y otros tokens puede ser más complejo.

Un lexema es la secuencia de caracteres en el código fuente que coincide con un patrón y que el analizador léxico reconoce como una instancia de un token.

ATRIBUTOS DE LOS COMPONENTES

El analizador léxico asocia atributos a los componentes léxicos, los cuales influyen en el análisis sintáctico y en su posterior traducción. Generalmente, estos atributos son apuntadores a la tabla de símbolos, donde se almacena la información relevante.

Para diagnóstico, se pueden registrar el lexema y la línea en la que apareció por primera vez. Algunos componentes léxicos no requieren atributos adicionales, mientras que otros, como los números, pueden almacenar su valor en la tabla de símbolos y usar un apuntador como atributo.

ESPECIFICACIONES

Las expresiones regulares son una notación utilizada para definir patrones, representando conjuntos de cadenas con características comunes. En este contexto, un alfabeto es un conjunto finito de símbolos, como letras o números (ej., {0,1} es el alfabeto binario). Una cadena es una secuencia finita de símbolos de un alfabeto, mientras que un lenguaje es un conjunto de cadenas sobre un alfabeto fijo. Este concepto abarca desde lenguajes formales, como los programas en Pascal, hasta lenguajes naturales, como el inglés.

EXPRESIONES REGULARES

En Pascal, un identificador comienza con una letra y puede seguir con letras o dígitos, representado por la expresión regular:
letra (letra | dígito)*
Las expresiones regulares definen patrones en un lenguaje y se construyen mediante reglas:
€ representa la cadena vacía.
Un símbolo a representa el conjunto {a}.
r | s indica la unión de dos lenguajes.
rs representa la concatenación.
r* aplica la clausura de Kleene (cero o más repeticiones).
Los lenguajes descritos por expresiones regulares se llaman lenguajes regulares y siguen reglas de precedencia:
* tiene la mayor prioridad.
La concatenación es la segunda.
| tiene la menor precedencia.

DEFINICIONES REGULARES

Para facilitar la notación, se pueden asignar nombres a las expresiones regulares y usarlos como símbolos en definiciones posteriores.
Una definición regular es una secuencia de asignaciones:
d1 → r1
d2 → r2
…
dn → rn
Aquí, cada di es un nombre único y cada ri es una expresión regular basada en símbolos del alfabeto Σ y en nombres previamente definidos. Sustituyendo estos nombres repetidamente, se puede construir una expresión regular final.

Para evitar definiciones recursivas infinitas, un nombre di no debe depender de otro nombre dj con j ≥ i. Para distinguirlos de los símbolos, los nombres de las definiciones regulares suelen escribirse en negrita.

RECONOCIMIENTO DE LOS COMPONENTES

El reconocimiento de componentes léxicos se basa en una gramática que define las estructuras del lenguaje, donde las reglas principales son:
prop → if expr then prop | if expr then prop else prop | €
expr → término oprel término | término
término → id | núm
Los terminales incluyen palabras clave (if, then, else) y otros tokens definidos por expresiones regulares:
oprel → < | <= | = | <> | > | >=
id → letra (letra | dígito)*
núm → dígito+(. dígito+)?(E(+ | -)? dígito+)?
El analizador léxico reconoce estos tokens y asume que las palabras clave son reservadas.
Además, elimina espacios en blanco representados por la expresión regular:
eb → delim+ (donde delim es un espacio, tab o nueva línea).
Cuando encuentra eb, el analizador lo ignora y busca el siguiente lexema válido. Como salida, genera pares de componente léxico y su atributo, usando constantes simbólicas para operadores relacionales como MEN, MEI, IGU, MAY, MAI.

DIAGRAMA DE TRANSICIONES

Un diagrama de transiciones es un paso intermedio en la construcción de un analizador léxico, donde se representan las acciones realizadas cuando el analizador léxico es llamado para obtener el siguiente componente léxico. Los estados (círculos) indican las posiciones en el proceso de reconocimiento, y las aristas (flechas) etiquetadas con caracteres de entrada guían el flujo entre estados. El estado inicial es donde comienza el reconocimiento del token, y el estado de aceptación (doble círculo) señala que el token ha sido reconocido. El diagrama funciona leyendo un carácter de entrada, siguiendo la arista correspondiente y avanzando a nuevos estados, hasta que se llega a un estado de aceptación o se detecta un fallo. En caso de fallo, el analizador retrocede al estado inicial y prueba otro diagrama de transiciones; si todos fallan, se activa una rutina de recuperación de errores. Por ejemplo, para reconocer > y >=, el diagrama comienza en el estado 0, avanza a diferentes estados según los caracteres leídos, y si se lee un carácter extra no perteneciente al token, se retrocede un carácter. Cada diagrama maneja un conjunto de tokens, y el analizador sigue varios diagramas hasta encontrar una coincidencia o detectar un error léxico.

AUTOMATAS FINITOS

Un reconocedor de lenguaje verifica si una cadena es válida en un lenguaje mediante un autómata finito, que puede ser determinista o no determinista. Ambos pueden reconocer conjuntos regulares, pero los autómatas deterministas son más rápidos, aunque más grandes que los no deterministas. Se estudian métodos para convertir expresiones regulares en ambos tipos de autómatas, comenzando con los no deterministas, que son más simples.

NO DETERMINISTA

Un autómata finito no determinista (AFN) es un tipo de autómata en el que, para un estado y un símbolo de entrada, puede haber más de una transición posible. Esto significa que el autómata puede estar en múltiples estados a la vez, lo que le permite explorar varias rutas de manera simultánea. A pesar de ser más flexible que los deterministas, los AFN no son tan eficientes en cuanto a espacio y tiempo. Sin embargo, son más fáciles de construir y pueden ser convertidos a autómatas finitos deterministas (AFD) si es necesario.

DETERMINISTA

Un autómata finito determinista (AFD) es un tipo de autómata finito donde, para cada estado y símbolo de entrada, hay a lo sumo una transición posible. No permite transiciones € (vacías) y tiene una única transición desde cada estado para cada entrada. Esto hace que sea fácil determinar si acepta o no una cadena, ya que solo existe un camino desde el estado inicial.
Para simular su comportamiento con una cadena de entrada, se utiliza un algoritmo que recorre la cadena y sigue las transiciones del autómata, respondiendo "sí" si la cadena es aceptada y "no" si no lo es.
ENTRADA: Una cadena de entrada x, que se termina con un carácter de fin de archivo eof. Un AFDD con el estado inicial S0, que acepta estados F, y la función de transición mover.
SALIDA: Responde "sí" en caso de que D acepte a x "no" en caso contrario.
MÉTODO: Aplíquese el algoritmo siguiente a la cadena de entrada x. La función mueve(s, c) da el estado al cual hay una transición desde el estado S en un carácter de entrada c. La función sigtecar devuelve el siguiente carácter de la cadena de entrada x

HERRAMIENTAS

LEX es una herramienta utilizada para construir analizadores léxicos a partir de expresiones regulares. Su funcionamiento comienza con la especificación de un programa en el lenguaje LEX (archivo lex.l), que describe cómo se deben reconocer los componentes léxicos de un lenguaje. Este archivo se pasa al compilador LEX, el cual genera un programa en C (lex.yy.c). Este programa en C contiene una representación tabular de un autómata de transiciones basado en las expresiones regulares de lex.l, junto con rutinas que permiten reconocer los lexemas y ejecutar las acciones correspondientes.
Un programa en LEX consta de tres secciones:
Declaraciones, que incluyen definiciones regulares y variables necesarias.
Reglas de traducción, que son expresiones regulares acompañadas de acciones en C que se ejecutan cuando el analizador encuentra una coincidencia.
Funciones auxiliares, que contienen los procedimientos adicionales que puedan ser requeridos por las acciones.
El analizador léxico creado por LEX funciona en sincronía con el analizador sintáctico. Cuando el analizador sintáctico activa al analizador léxico, este lee los caracteres de entrada hasta encontrar el mayor prefijo que coincida con una de las expresiones regulares definidas. Luego, ejecuta la acción correspondiente y devuelve el control al analizador sintáctico. Si no encuentra coincidencias, el analizador léxico continúa buscando más lexemas.
Además, LEX permite el uso de la construcción lookahead, que es útil cuando se necesita mirar más allá del final de un lexema para determinar su validez, lo cual es importante para manejar contextos específicos de ciertas construcciones léxicas.
En cuanto a la recuperación de errores lexicográficos, cuando se detectan errores, como caracteres fuera del vocabulario o errores de tipeo, LEX puede intentar corregirlos de forma automática generando sinónimos válidos del token erróneo, como en los casos de omisión, cambio, o permutación de caracteres. Esto permite al analizador continuar el proceso de manera fluida, proporcionando advertencias al usuario cuando se corrigen los errores. Estos procedimientos de recuperación de errores son específicos para el lenguaje que se está analizando y dependen de las reglas del lenguaje en cuestión.