Documentación Práctica 3 GLF

Intérprete del Lenguaje P

Adrián Arroyo Calle

El lenguaje P es un subconjunto del lenguaje C. El lenguaje debe soportar asignaciones, condicionales if-else, bucles while, aritmética y operaciones de entrada/salida.

El intérprete diseñado para este lenguaje, pcc, está programado en C, ayudándose de las herramientas Lex (concretamente Flex) y Yacc (concretamente Bison). El intérprete, primero analiza el código fuente en lenguaje P, detectando posibles errores de sintaxis, elaborando un árbol AST. Al finalizar la lectura del código sin errores, se procede a ejecutar el árbol AST generado.

Para el código se ha partido de la base del lenguaje de Dusan Kolar.

Analizador Léxico

En primera instancia, el código fuente es analizado por el analizador léxico, construido con Lex. El código fuente se encuentra en el fichero pcc.l. Se definen expresiones regulares para capturar el código “útil” y devolver tokens al analizador sintáctico. La mayoría de estas expresiones regulares son simples: detectar “identificadores” (palabras clave y nombres de variable, no se distingue en este punto), constantes de string, char, char especial (símbolos como \n que son dos caracteres, pero equivalen a un único caracter), los diferentes tipos de float (formatos aceptados por scanf), los operadores (de uno y dos caracteres), los comentarios y los espacios en blanco.

Cuando se encuentra un identificador (expresión regular IDENT), se comprueba que si el identificador es una palabra reservada, en cuyo caso se devuelve el token correspondiente a esa palabra reservada. En caso contrario, se devuelve el token IDENT con el nombre detectado.

Existen dos tipos de comentarios, los de línea y los multilínea. Los de línea funcionan gracias a que Lex busca siempre la mayor cadena de texto que haga match a la expresión regular. Cuando se encuentran dos barras, todo caracter a su derecha entra en la expresión, cuya acción es no hacer nada. Los multilínea necesitan de una condición de carrera. Esta es activada al detectar el comienzo del comentario /*, e ignora todo hasta encontrar el final */.

Para la lectura de strings se usa un método dinámico, que va leyendo una vez comenzada la cadena, y permite que la cadena de texto crezca indefinidamente. Aquí el código de Dusan tenía un bug, ya que ponía “c==input();” en vez de “c=input();”. Esto era usado en la parte necesaria para reconocer caracteres especiales. Actualmente solo se reconoce \\ y \n.

Los caracteres separados, con comillas simples, se identifican de forma simple, ya que siempre ocupan tres caracteres, y se toma el caracter de en medio. Los caracteres especiales, ocupan cuatro caracteres, contando la barra, pero la operativa es similar.

Análisis Sintáctico

El analizador sintáctico se ha realizado con Yacc, está implementado en el fichero pcc.y. En primer lugar se definen los tokens que va a poder reconocer. Para algunos de ellos se diseña una estructura de datos auxiliar, que contiene información extra.

A continuación se detalla la gramática. El axioma de partida es prog, y un prog puede estar hecho de una o más líneas (que no son exactamente las líneas del fichero fuente). Este elemento representa el nodo raíz en el árbol AST. Las líneas son en realidad sentencias o sentencias con salto de línea. También valen los saltos de línea vacíos. Esta gramática genera un conflicto shift/reduce. Como Yacc por defecto va a hacer shift, no hay problema, ya que tiene prioridad la gramática más larga.

Las sentencias por su parte son de muchos tipos, incluyen la asignación, los diferentes WRITE y READ, así como IF, IF-ELSE, WHILE y FOR. También se admite el punto y coma (la sentencia vacía).

De estas sentencias la más complicada es la FOR. Esta sentencia es parecida al for de C. Toma primero una asignación, después una expresión que se evaluará positiva o negativamente, y por último otra asignación. Entre llaves se ubica el cuerpo del bucle. El cuerpo se compone de líneas (tal y como se han definido arriba). Se ha tomado la decisión se solo permitir asignaciones en los huecos 1 y 3 del FOR ya que son las únicas operaciones que tienen sentido. Internamente se crea un nodo AST, con dos hijos, de tipo FOR1 y FOR2 respectivamente. FOR1 tiene otros dos hijos, la asignación de paso de bucle, y la expresión de control. FOR2 contiene la asignación inicial y el cuerpo del bucle.

Por último, las expresiones tienen las producciones más grandes. Una expresión siempre devuelve un valor numérico. Existen expresiones terminales: FLOAT e IDENT. FLOAT es una constante numérica mientras que IDENT obtendrá de la tabla de variables su valor. Solo existen variables numéricas, ya que las cadenas de caracteres y los caracteres solo pueden aparecer como constantes. También se definen los paréntesis para poder separar correctamente las operaciones aritméticas.

El resto de producciones son operaciones aritméticas, lógicas o funciones matemáticas. Todas son muy similares y su verdadera acción se produce en el árbol AST. Para evitar conflictos y asignar prioridades de forma correcta, en la primera parte del fichero Yacc se define la precedencia de muchos de estos operadores.

Por último, el fichero pcc.y contiene la función main. Esta indica a Yacc el fichero a leer y cuando el parseo ha sido correcto, ejecuta el árbol AST.

Árbol AST

El fichero astree.c contiene la definición de las estructuras del árbol AST, generado con un parseado exitoso de Yacc. Además de las funciones de creación, presentes en el código original de Dusan, se encuentran funciones para recorrer el árbol y así poder ejecutar el programa cargado. Se han añadido

las casuísticas de los nuevos operadores y funciones matemáticas (función expr), mientras que en la función proc se han añadido WHILE, IF y FOR. Además se ha modificado el comportamiento de WRITE y READ para que nunca añadan saltos de línea si no es especificado en la cadena de texto.

Tests

Existe una carpeta de tests en el código fuente. Se incluyen los dos ficheros de test provistos en la descripción de la práctica. Adicionalmente existen otros ficheros de test para probar diferentes elementos del lenguaje.

Otros ficheros

Por último, existen otros ficheros (error.c, keywords.c) que incluyen funciones de manejo de error y el listado de palabras reservadas.

Además, existe un fichero Makefile para automatizar el proceso de compilación. Es muy importante el orden de compilación. Primero Yacc, después Lex y después el resto del código C.