Gramáticas y Lenguajes Formales
Práctica 3

Juan González Caminero

Mayo 2019

Contents

1 ppc.l

He mantenido la estructura general del programa de Dusan, haciendo los cambios necesarios para adaptarlo a la especificación del lenguaje P.

He añadido las palabras clave if, while, else, cos, asin, acos, atan, log, log10, exp, ceil, floor, read y write, y eliminado let y print ya que las asignaciones ahora se realizan sólo con un =, y la palabra clave para escribir por pantalla es write.

Para el resto de la parte de definiciones, mantengo gran parte de lo que usaba Dusan, las expresiones regulares para detectar letras, números, identificadores y operadores, aunque en el caso de los operadores añado los nuevos que no existían en el lenguaje original.
Añado dos expresiones para detectar los comentarios y así poder ignorarlos al leer el fichero. También una para detectar los caracteres (,), { y }, y otra que detecta los ;.

En la sección de órdenes, hago que se ignoren espacios en blanco y comentarios. Cuando encuentro un cambio de línea, no devuelvo nada, ya que en nuestro lenguaje los cambios de línea no terminan las sentencias, pero le sumo 1 a la variable yylineno, que está declarada como externa en otros ficheros como astree.c para poder acceder desde allí. En caso de encontrar un ;, no le sumo al número de línea pero sí devuelvo el caracter.

Al encontrar un operador aritmético, símbolo =, o un paréntesis o corchete devuelvo el símbolo leído. Sin embargo, el el caso de los operadores lógicos devuelvo el tóken correspondiente, ya que la mayor parte están compuestos por dos caracteres y no podemos poner un string en la gramática de Yacc.

Cuando encuentro un caracter (entre comillas simples) reservo espacio para 1 caracter y almaceno en yylval.s.u.vStr lo leído. En caso de leer un caracter escapado, hago un switch para determinar de cuál de los posibles caracteres escapados se trata y lo almaceno en el mismo lugar. Es necesario hacer esto ya que de otra forma lex interpretaría estos caracteres como una contrabarra y una letra en vez de un caracter escapado.

Por último, mantengo en la última sección del fichero las funciones de Dusan, ya que aunque son sencillas de implementar no veo el motivo de reescribirlas.
Sin embargo, en el caso de la función que lee un string sí hago cambios. En primer lugar corrijo un bug que tenía la versión original, que hacía que no se leyeran correctamente los caracteres escapados. Cambio c == input() por c = input() dentro del if que comprueba si se ha leído una contrabarra.
A continuación, para poder imprimir correctamente los caracteres escapados, no podemos guardar los dos caracteres que los componen, la contrabarra y la letra, por separado, sino que debemos almacenarlos como un sólo caracter. Por tanto, dentro del if creo un switch que detecta si el caracter siguiente a la

contrabarra completa uno de los caracteres de escape de ASCII, en cuyo caso guardo el caracter correspondiente como un char en la variable c, que después se almacenará en el buffer que se devuelve. En caso de no corresponder a ningún caracter de escape, devuelvo lo leído a la entrada y almaceno sólamente una contrabarra en el buffer.

2 ppc.y

Hago un pequeño cambio en el tipo union, llamando a la variable que almacena el valor numérico del nodo, en caso de tenerlo, vNumber en vez de vFloat.
Declaro la asociatividad de los operadores fijándome en cómo lo hace Dusan, añadiendo también los nuevos.

La gramática tiene más diferencias con el programa anterior:

-La variable inicial es program, que puede derivar en program o en program progelement.
Al igual que Dusan, cuando leo un progelement añado el subárbol que corresponde a esa línea después del último nodo que haya en el árbol de análisis del programa.

-Progelement puede derivar en una sentencia, una sentencia condicional o bien un ";", que es una sentencia vacía pero es válida en nuestro lenguaje.
En caso de derivar en un ;, ponemos a NULL el campo ast de la estructura devuelta, de forma que esta línea no aparezca en el árbol de análisis, ya que ocupará su lugar la próxima línea con contenido, al leer NULL la función appR de astree.c.

-Una sentencia condicional puede ser o bien un If seguido de una expresión entre paréntesis y después un bloque entre corchetes, o un while seguido de lo mismo. En el caso del if el bloque puede ir seguido también de un else y un bloque.
En el caso del if añadimos al nodo la etiqueta IF y dos hijos, uno con la expresión de la concidión, y otro con un nodo auxiliar, que tendrá un hijo con el subárbol correspondiente al bloque del if y otro NULL, o bien un hijo con el subárbol del bloque del if y otro con el subárbol del else.
En el caso del while añado al nodo correspondiente a la línea la etiqueta WHILE, y dos hijos. El izquierdo correspondiente a la expresión entre paréntesis después del if o while, y el otro que será la raíz del subárbol asociado al bloque de código que sigue al while.

-Una sentencia puede ser una asignación, que consiste en un identificador, un =, y una expresión, o un "read" o "write", seguidos de uno o dos argumentos entre paréntesis, que serán un string o un identificador, en el caso de un sólo argumento, o un string y un identificador si son dos argumentos.

En cualquier caso, se añade al nodo la etiqueta correspondiente a lo encontrado, =, READ o WRITE, dos hijos en el caso de la asignación y las funciones con dos argumentos, y un sólo hijo en las funciones con un sólo argumento, que estará a la izquierda si es un String, y a la derecha si es un identificador.

-Una expresión puede ser un identificador, un número, una expresión entre paréntesis, una o dos expresiones junto a un operador, tanto aritmético como lógico, o una de las funciones matemáticas existentes en el lenguaje seguidas de una expresión entre paréntesis.
Si la expresión es un número o un identificador creamos un nodo especial que no tiene hijos y sólo almacena el número en un float o el nombre de la variable en un string, respectivamente.
Si la expresión está entre paréntesis, símplemente toma el valor de la expresión entre paréntesis.
En caso de haber un operador, se añade al nodo la etiqueta correspondiente al operador y uno o dos hijos en función del operador, lo mismo ocurre con las funciones, que sólo tienen un hijo cada una.

-Por último, un bloque puede ser un progelement, o un bloque seguido de un progelement. En los bloques seguimos el mismo proceso que se sigue desde el nodo raíz del árbol de análisis, leyendo sentencias y añadiéndolas a la raíz del bloque, hasta que no encontramos más sentencias.

3 astree.c

Las funciones para la creación de los nodos del árbol siguen igual que en el programa de Dusan. La mayoría de cambios están en la sección de ejecución del programa.

expr() recibe el nodo correspondiente a una expresión y devuelve el valor asociado después de ejecutar la operación correspondiente. El mayor cambio aquí es la adición de los nuevos operadores y funciones.

proc() lee la etiqueta asociada a una sentencia y hace las llamadas correspondientes. Un pequeño cambio es usar printf en la función write en vez de puts, como hacía Dusan, ya que puts siempre pone un cambio de línea al final de lo que imprime y con printf puedo evitarlo, de forma que el programador siempre tenga el control de dónde se introducen los cambios de línea al imprimir por pantalla.

El mayor cambio es la adición de las sentencias If y While. Al entrar a ejecutar un if se declara un ast_t, block_root, que toma el valor del hijo izquierdo del árbol auxiliar a la derecha del nodo del if.
Si la expresión a la izquierda del nodo del if evalua a algo mayor que 0, entramos

en un while igual al que se usa para recorrer el árbol de análisis del programa, que recorre el subárbol del bloque asociado al if y llama a proc() sobre cada línea.
Si la expresión es menor o igual a 0, y el hijo derecho del árbol auxiliar asociado al if no es NULL, se ejecuta el else. Para ello, se asigna a block_root el hijo derecho del árbol auxiliar, que corresponde al bloque del else, y se ejecuta igual que el if.

El while es esencialmente igual, con la diferencia de que el hijo derecho del nodo del while ya es la raíz del subárbol asociado al bloque, y que el código se ejecuta mientras la expresión evalue a algo mayor que 0.

4 Implementación del for

Para empezar añado el token for en el yacc y la palabra clave for en el lex.
Añado el for en el lex como una expresión condicional. Como sólo queremos permitir sentencias que modifiquen una variable, en vez de permitir cualquier sentencia sólo permito la asignación. La parte derecha de la regla queda:

FOR '(' IDENTIFIER '=' expression ';' expression ';' IDENTIFIER '=' expression ')' '{' block '}'

El nodo tiene una etiqueta FOR y dos hijos auxiliares, el de la izquierda tiene como hijos la primera asignación y la expresión a su derecha, el de la derecha, la asignación de la derecha y la raíz del subárbol que contiene las sentencias del bloque asociado al for.

Para ejecutarlo, ejecuto primero la asignación de la izquierda, y después, mientras la expresión central evalue a algo mayor que 0, ejecuto las sentencias del bloque de código y la asignación de la derecha.

5 Test

Para probar el intérprete incluyo los dos programas que aparecen en el enunciado de la práctica, test_program y test_program_2, y varios programas que prueban funcionalidades específicas, aunque dentro de estos he intentado combinar otras funcionalidades para que se pueda comprobar que también funcionan correctamente juntas.

6 Referencias

Ficheros asociados a la charla del profesor Dusan Kolar