Comprobaciones semánticas para otras expresiones

Introducción

- El no terminal **exp** tiene varias producciones además de las relativas a expresiones aritméticas y lógicas cuyas comprobaciones semánticas ya han sido descritas. Estas producciones son:
  - (80) exp: TOK_IDENTIFICADOR
  - (81) exp: constante
  - (82) exp: '(' exp ')'
  - (83) exp: '(' comparacion ')'
  - (85) exp: elemento_vector
  - (88) exp: TOK_IDENTIFICADOR '(' lista_expresiones ')'
- Las comprobaciones semánticas de cada una de las producciones se describe a continuación.

Comprobaciones semánticas para otras expresiones

Implementación

- En la producción siguiente:

  (80) exp : TOK_IDENTIFICADOR

  Se comprueba que:
  - El identificador ($1.lexema) existe en la tabla de símbolos (en cualquiera de los ámbitos abiertos). Si el identificador no existe, se genera un error semántico y se termina el proceso de compilación con error.
  - El identificador no es ni de categoría función (es decir, es una variable o un parámetro), ni de clase vector (es decir, es un escalar).

  Se calculan los atributos:
  - El tipo del símbolo exp de la parte izquierda ($$.tipo) se establece haciendo uso de la información almacenada en la tabla de símbolos para el identificador de la parte derecha de la producción. (IMPORTANTE, $1.tipo no almacena el tipo del identificador, el tipo está en la tabla de símbolos).
  - El símbolo de la parte izquierda representa una dirección de memoria ($$.es_direccion=1).

Comprobaciones semánticas para otras expresiones

Implementación

- Las producciones (81) a (85) únicamente contienen en su acción semántica las siguientes instrucciones para propagar atributos:
  - $$.tipo = $1.tipo en las producciones 81 y 85.
  - $$.tipo = $2.tipo en las producciones 82 y 83.
  - $$.es_direccion = $1.es_direccion en las producciones 81 y 85.
  - $$.es_direccion = $2.es_direccion en las producciones 82 y 83.

Comprobaciones semánticas para otras expresiones

Implementación

- La producción 88 define la sintaxis de la llamada a una función:

  (88) exp: TOK_IDENTIFICADOR '(' lista_expresiones ')'

- La descripción de la semántica es la siguiente:
  - En las sentencias de llamadas a funciones, sólo es necesario comprobar la corrección del número de argumentos. No es necesario realizar ninguna comprobación de la correspondencia de tipos.
  - En las llamadas a funciones, no se puede escribir, en un parámetro, una llamada a otra función.

- En primer lugar, se comprueba que el identificador existe en la tabla de símbolos y además tiene categoría de función.

Comprobaciones semánticas para otras expresiones

Implementación

- Para comprobar la corrección del número de argumentos es necesario un mecanismo que permita "contar" el número de elementos de la lista de expresiones para poder compararlo con el número de parámetros con el que se definió la función y que está almacenado en la tabla de símbolos.

- Un posible mecanismo sería usar una variable global **num_parametros_llamada_actual** para contar el número de parámetros de la llamada. La inicialización a 0 de esta variable se haría antes de comenzar a procesar el no terminal lista_expresiones, y se incrementaría en 1 cada vez que se procesara uno de los elementos de la lista.

- Para comprobar que no haya una llamada a función en el lugar de un parámetro, se puede utilizar otra variable global **en_explist** que se inicialice a 1 antes de procesar el no terminal lista_expresiones. Y posteriormente, en cualquier llamada a función se comprobará que en_explist no sea 1. Por último, al terminar la llamada a función se reseteará a 0 esta variable.

Comprobaciones semánticas para otras expresiones

Implementación

- Agrupando todas las tareas necesarias para implementar la semántica de la llamadas a funciones, se deduce que hay dos puntos en los que se ubican las acciones:

  (88) exp: TOK_IDENTIFICADOR ❶ '(' lista_expresiones ')' ❷

- Se modifica la gramática para situar la acción del punto ❶ al final de una producción, añadiendo la siguiente regla:

  idf_llamada_funcion : TOK_IDENTIFICADOR ❶

  Por lo tanto, la producción 88 de la gramática original ahora quedaría de la siguiente manera:

  (88) exp: idf_llamada_funcion '(' lista_expresiones ')' ❷

Comprobaciones semánticas para otras expresiones

Implementación

- La acción del punto ❶ implica las siguientes tareas:
  - Buscar el identificador en la tabla de símbolos. Generar un error semántico si no existe y terminar el proceso de compilación con error.
  - Comprobar que el identificador es de categoría función.
  - Como después del identificador empieza la lista de expresiones correspondiente a los parámetros de la llamada, hay que comprobar que la variable en_explist no valga 1 para asegurar que la actual llamada a función no es un parámetro de llamada a función.
  - Inicializar a 0 la variable num_parametros_llamada_actual.
  - Inicializar a 1 la variable en_explist ya que después del identificador empieza la lista de expresiones correspondiente a los parámetros de la llamada.
  - Propagar el lexema del identificador ya que se va a necesitar en el punto ❷.

- La acción del punto ❷ implica las siguientes tareas:
  - Comprobar que el número de parámetros de la llamada es correcto comparando el valor de la variable global num_parametros_llamada_actual con la información almacenada en la tabla de símbolos.
  - Resetear a 0 la variable en_explist.
  - El tipo del símbolo de la parte izquierda exp ($$.tipo) se iguala al tipo de retorno de la función que está almacenado en la tabla de símbolos.
  - El símbolo dela parte izquierda no representa una dirección, es un valor ($$.es_direccion=0).

Comprobaciones semánticas para otras expresiones

Implementación

- Por último, sólo faltaría decidir los puntos en los que se debe actualizar la variable global num_parametros_llamada_actual. Las producciones indicadas para ello son las siguientes:

  (89) lista_expresiones: exp resto_lista_expresiones

  (91) resto_lista_expresiones: ',' exp resto_lista_expresiones

  En ambas producciones únicamente hay que realizar la acción:

  num_parametros_llamada_actual++

Comprobaciones semánticas en sentencias de asignación

Comprobación de tipos

- **Descripción de la semántica:** Para que una sentencia de asignación sea válida, los tipos de dato de la parte izquierda y derecha deben ser iguales.

- **Comprobaciones semánticas:** hay que comprobar que son iguales los tipos de la parte izquierda y derecha de la asignación en las siguientes producciones:

  - (43) asignacion : TOK_IDENTIFICADOR '=' exp
  - (44) asignacion : elemento_vector '=' exp

Comprobaciones semánticas en sentencias de asignación

Implementación

- La primera producción corresponde a la asignación de una expresión a un identificador:

  (43) asignacion: TOK_IDENTIFICADOR '=' exp

  Se comprueba que:
  - El identificador existe en la tabla de símbolos.
  - El identificador no es de categoría función (es decir, es una variable o un parámetro) y no es de clase vector (es decir, es un escalar).
  - Los tipos de ambas partes de la asignación son iguales.

  No hay propagación de atributos, no es necesario.

- La segunda producción corresponde a la asignación de una expresión a un elemento de un vector:

  (44) asignacion: elemento_vector '=' exp

  Se comprueba que:
  - Los tipos de ambas partes de la asignación son iguales.

  No hay propagación de atributos, no es necesario.

Comprobación del tamaño

- **Descripción de la semántica:** El tamaño de los vectores no podrá exceder nunca el valor de 64.

- **Comprobaciones semánticas:** hay que comprobar que el tamaño declarado para un vector no excede el límite permitido (64). También hay que comprobar que el tamaño declarado es mayor que 0 (esta restricción no se especifica explícitamente en la descripción de la semántica del lenguaje). Estas comprobaciones se pueden realizar la siguiente producción:

  - (15) clase_vector : TOK_ARRAY tipo '[' TOK_CONSTANTE_ENTERA ']'

(VER EPÍGRAFE "Acciones semánticas de las producciones 15 y 16")

Comprobaciones semánticas para vectores

Comprobaciones en el indexado de vectores

- **Descripción de la semántica:** Para indexar los elementos de un vector se utiliza la cadena "[exp]"a continuación del nombre del vector. Los corchetes deberán contener en su interior una expresión de tipo **int**.

- **Comprobaciones semánticas:** la producción indicada para comprobar la semántica descrita es la siguiente:

  - (48) elemento_vector : TOK_IDENTIFICADOR '[' exp ']'

  Las comprobaciones que hay que realizar son:

  - El identificador que aparece ($1.lexema) existe en la tabla de símbolos y además corresponde a la declaración de un vector.
  - La expresión que actúa como índice es de tipo INT ($3.tipo == INT).

  Se propagan los atributos:

  - $$.tipo = el tipo que aparece para el lexema $1.lexema en la tabla de símbolos.
  - $$.es_direccion = 1.

Comprobaciones semánticas para vectores

Comprobaciones en el indexado de vectores de una dimensión

- **Descripción de la semántica:** Los corchetes deberán contener en su interior una expresión de tipo **int**, con un valor entre 0 y el tamaño definido para el vector menos 1 (ambos incluidos).

- **Comprobaciones semánticas:** en una operación de indexado de un vector de una dimensión, la comprobación de que el valor del índice está dentro del rango permitido sólo se puede realizar en **tiempo de ejecución**, no en tiempo de compilación. Para implementar esta restricción semántica se genera el código ensamblador adecuado para realizar la validación del valor del índice en tiempo de ejecución. La producción indicada para la generación del código es:

  - (48) elemento_vector : TOK_IDENTIFICADOR '[' exp ']'

Comprobaciones semánticas para las condiciones

- **Descripción de la semántica:** Las expresiones que aparecen como condiciones en las sentencias **if**, **if-else** y **while** deben ser de tipo boolean.

- **Comprobaciones semánticas:** hay que comprobar que las condiciones de los bucles while y de las sentencias condicionales son de tipo boolean.
  Las producciones implicadas en la comprobación de esta restricción semántica son las siguientes:

  - (50) condicional : TOK_IF '(' exp ')' '{' sentencias '}'
  - (51) condicional : TOK_IF '(' exp ')' '{' sentencias '}' TOK_ELSE '{' sentencias '}'
  - (52) bucle : TOK_WHILE '(' exp ')' '{' sentencias '}'

  Las comprobaciones se realizan sobre el no terminal exp que es el que representa la condición.

Comprobaciones semánticas para las condiciones

- **Modificación de la gramática:** los puntos adecuados para realizar las comprobaciones sobre el símbolo exp se sitúan después de él y antes del no terminal sentencias. Por ejemplo, para la sentencia condicional simple, los puntos posibles son los indicados con **{acción}** :
  - (50) condicional : TOK_IF '(' exp **{acción}** ')' '{' sentencias '}'
  - (50) condicional : TOK_IF '(' exp ')' **{acción}** '{' sentencias '}'
  - (50) condicional : TOK_IF '(' exp ')' '{' **{acción}** sentencias '}'

  Una vez elegido el punto, para poder situar la acción semántica al final de la regla, sería necesario modificar la gramática. Por ejemplo si se elige el primer punto, la gramática se manipula de la siguiente manera:
  - Se añade un nuevo símbolo no terminal **if_exp** para situar la acción semántica al final de una regla:

    if_exp : TOK_IF '(' exp **{acción}**

  - La producción original (50) quedaría de la siguiente manera:

    (50) condicional : if_exp ')' '{' sentencias '}'

Comprobaciones semánticas para las condiciones

- En la nueva regla,

  if_exp : TOK_IF ‘(‘ exp **{acción}**

  Se comprueba que:
  - La expresión es de tipo boolean, es decir, $3.tipo == BOOLEAN

- Se aplica el mismo razonamiento para la sentencias **if-else** y **while**.

Comprobaciones para sentencias de entrada

- **Descripción de la semántica:** Las operación de entrada **scanf** se efectúa sobre elementos de clase escalar que son identificadores.

- **Comprobaciones semánticas:** La producción de entrada de datos es:

  - (54) lectura : TOK_SCANF TOK_IDENTIFICADOR

  Solamente es necesario realizar la siguiente comprobación:

  - El identificador ($2.lexema) existe en la tabla de símbolos, y es de clase escalar.

Comprobaciones para sentencias de salida

- **Descripción de la semántica:** La operación de escritura **printf** trabaja con expresiones.

- **Comprobaciones semánticas:** La producción correspondiente a la operación de escritura **printf** es la siguiente:
  - (56) escritura : TOK_PRINTF exp

    En esta producción no hay que realizar ninguna comprobación semántica.

Otras comprobaciones semánticas

- **En la descripción de la semántica de ALFA proporcionada en el enunciado de la práctica sólo se mencionan los aspectos más relevantes. Además de éstos, hay un conjunto de restricciones semánticas que también deben estar presentes en el compilador de ALFA, como por ejemplo:**
  - Una sentencia de retorno de función solamente debe aparecer en el cuerpo de una función.
  - En el cuerpo de una función obligatoriamente tiene que aparecer al menos una sentencia de retorno.
  - En una sentencia de retorno el tipo de la expresión debe de coincidir con el tipo de retorno de la función.
  - etc