Procesadores del Lenguaje

PECL3: Construcción de un analizador semántico

Mario Ríos Muñoz
09047759L
29 Noviembre 2014

Indice:

Abstracto:

El objetivo de esta práctica es implementar el análisis semántico, partiendo de los analizadores léxico y sintáctico creados en las prácticas anteriores.

Modificaciones respecto a la PECL2

El código generado en la práctica anterior ha sido reutilizado prácticamente en su totalidad. Solamente se han añadido las piezas de código necesarias para implementar las reglas semánticas solicitadas, y se ha modificado ligeramente la clase storeClass para poder implementar las mismas.

Acerca de la modificación de storeClass:

Con el fin de tener los datos necesarios para implementar la regla número 3, se ha decidido añadir un atributo a la clase storeClass de tipo Hashtable que contiene un mapa entre nombre y tamaño de todos los registros existentes en el fichero analizado.

También se ha implementado un método llamado isCorrect, que devuelve verdadero si la regla 3 se cumple, y falso si no lo hace, ayudándose de la estructura anteriormente mencionada.

Implementación de las reglas semánticas

Regla 1, ProgramPC y FlagVector:

Esta regla especifica que si un registro tiene como uso ProgramPC o FlagVector, no puede tener ningún tipo de uso.

Para comprobar esta condición, en la condición contenido_registro se hace una búsqueda de los valores en cuestion en la Stack que representa los usos. Si alguno de estos valores existe en la pila, y el tamaño de la pila es mayor que uno, quiere decir que la regla no se cumple, por lo que se reporta el error.

Regla 2, referencia a registros no declarados:

Si a la hora de definir los registros que conforman un grupo, se usan nombres de registros que no han sido declarados, se debe lanzar un error.

Para conseguir esto, se ha creado un Hashtable en la sección de action code, de nombre registros, en el que se introducen los usos de cada registro declarado, dándole como clave el nombre de ese registro.

De esta forma, en la producción de los no terminales ‘registros_aux’ y ‘sub_registros’, se mira que el nombre de registro reconocido exista en la tabla, y si no es así se lanza un error.

Regla 3, integridad en el tamaño del grupo:

El tamaño de un grupo no puede ser mayor que la suma de los tamaños de los registros que lo forman, aunque si puede ser más pequeño.

Para comprobar esta condición, se ha usado la tabla mencionada anteriormente en la clase storeClass. Esta estructura es una tabla que mapea nombre de registros con el tamaño del

mismo. Cuando se crean registros, se añaden a esta tabla con su valor del tamaño correspondiente.

Cuando se crea un grupo, se pasa a la clase storeClass que lo representa esta tabla, de manera que el objeto creado pueda acceder a la misma.

Seguidamente, se llama a la función isCorrect. Esta función, recorre la pila que acumula los subregistros del grupo. Por cada registro en la pila, hace una búsqueda en la tabla, y si el registro existe, recupera su tamaño. Se suman todos los tamaños, y al final se comparan con el tamaño del registro. Si es menor, se devuelve falso.

A la hora de llamar a la función isCorrect, si esta devuelve falso, se reporta un error.

Un efecto colateral de infringir la regla 2, es que a la hora de de comprobar la regla 3, ninguno de los subregistros existe, su suma total es 0, por lo que también se infringiría la regla 3.

Regla 4, registro de destino no puede ser FlagVector:

En la definición de instrucciones, si el registro en el que se guarda el resultado es de tipo FlagVector, se lanza un error.

Para implementar esta comprobación, se usa la Hashtable 'registros' mencionada anteriormente, que relaciona nombre de registro con usos del mismo.

Cuando se reduce el no terminal 'salida', se busca en la tabla registros el registro reconocido. Si no se encuentra se lanza un error. Si se encuentra, se recupera la pila con los usos de este registro. Se busca entre los contenidos de la pila el valor 'FlagVector' y si se encuentra, se lanza un error.

Tratamiento de errores:

En la sección de action code, se ha implementado un método 'error_semántico' que recibe como parámetro el mensaje de error y la línea en la que tiene lugar el mismo. La función de este método es dejar constancia en la salida del programa, de la existencia de un error semántico.

Casos de prueba:

Se entregan tres fichero de prueba .upd: correcto.upd, incorrecto1.upd e incorrecto2.upd.

A continuación se describen los casos de prueba que se han analizado en cada uno:

correcto.upd: Este archivo se ha redactado siguiendo las normas de la gramática especificada. El objetivo de este fichero de pruebas, es que al analizarse, no genere ningún error.

incorrecto1.upd: En este archivo se han probado las reglas 1, 3 y 4.

- Línea 12: Se han dado los usos ProgramPC y FlagVector, para probar que teniendo los dos se lanza el error 'Usos de registro incompatibles'
- Línea 19: Similar al anterior.
- Línea 26: Se dan los valores Accumulator, FlagVector, StackPointer para demostrar que se identifica la presencia de FlagVector entre usos que sí pueden estar juntos. El error reportado es 'Usos de registro incompatibles'.
- Línea 43: Similar al anterior pero usando ProgramPC en vez de FlagVector.
- Línea 52: Se prueba que la regla 2 se analiza correctamente. Para ello se da un valor de bitsize mayor a la suma de subregistros. El error obtenido es 'Tamano de grupo incoherente'
- Línea 86: Se da como salida un registro con uso FlagVector. El error resultante es 'El registro de destino no puede ser de tipo FlagVector'

- Línea 93: Se da como salida un grupo con uso FlagVector: El error resultante es ‘El registro de destino no puede ser de tipo FlagVector’

incorrecto2.upd: En este archivo se ha probado la regla 2. Además, se ha dado como registro de salida de una instrucción, registros que no han sido declarados.

- Línea 21: Se dan como subregistros uno que existe y otro que no existe. El que no existe genera un error de tipo ‘Registro 'AH' no existe’
- Línea 29: Se dan como subregistros dos registros que no existen. Esto genera dos errores similares al anterior.
- Línea 27: Como los subregistros no existen, su tamaño asociado es 0, por lo que la regla 3 se incumple. Esto genera un error de tipo ‘Tamano de grupo incoherente’
- Lína 37 y 53: Se dan como salida registros que no existen. El resultado de esto es un error de tipo ‘registro no existe’.

Comentarios sobre el entorno de desarrollo:

Para la realización de esta práctica se ha usado netbeans, con una particularización: La creación de la implementación de los analizadores a partir de los ficheros de especificación .lex y .cup se ha realizado mediante un makefile que automatiza la tarea.

Es muy probable que dicho makefile no funcione en el ordenador del profesor, dado que para llamar a los comandos se utiliza una ruta absoluta, y el profesor puede haber instalado las herramientas en directorios diferentes. Aún así, dicho makefile es entregado junto con el resto de ficheros.

En netbeans, se ha tenido que añadir los directorios de JLex y JCup a la fuente del proyecto, para que los import de las clases de cada uno sean válidos. Es probable que el profesor tenga que repetir esta tarea en su IDE a la hora de corregir la práctica.