Universidad Simón Bolívar

Departamento de Computación y Tecnología de la Información

CI3725 - Traductores e Interpretadores

Abril - Junio 2015

Lanscii - Etapa III

Tabla de símbolos y chequeo de tipos (7%)

Especificación de la entrega

Hasta este momento hemos construido un analizador lexicográfico y sintáctico para el lenguaje de programación Lanscii. Esta siguiente etapa del desarrollo corresponde a la verificación del uso correcto de tipos dentro del lenguaje además de la implementación y uso de una Tabla de Símbolos.

Al finalizar esta etapa, su interpretador debe de reportar -*si existen*- los siguientes errores:

- Errores de redeclaraciones de variables dentro de un mismo alcance.
- Utilización de variables no declaradas en ese alcance.
- Modificación de variables de iteración.
- Errores de tipos. (por ejemplo: Intentar sumar un entero y un booleano).

La tabla de símbolos a utilizar debe ser implementada con una estructura eficiente, como una tabla de hash. Por motivo de las instrucciones de incorporación de alcance, la tabla de símbolos debe ser jerárquica. Esto es, si un nombre **no** se encuentra en la tabla de símbolos local, debe buscarse en la tabla de símbolos inmediatamente superior. Solo si, al recorrer todas las tablas de símbolos visibles en ese alcance no se encuentra el nombre, es que debe reportarse que no ha sido declarado. Los errores de redeclaración de variables solo pueden ocurrir en un mismo nivel (donde no se han incorporado nuevos alcances). Si un nombre es usado en diferentes niveles, el del nivel más interno esconde la definición de el del nivel más externo en la extensión de su alcance.

Recuerde que las variables de iteración (`[ <ident> : <expr 0> .. <expr 1> | <instr> ]`, en este caso `<ident>`) también se considera para la tabla de símbolos, aunque no se encuentre definida en un alcance.

La implementación debe estar en un módulo o archivo aparte y **debe incluir** como mínimo los siguientes procedimientos (si está programando en castellano, puede colocar los equivalentes):

- Constructor de una tabla de símbolos vacía:
  - Esto depende del lenguaje utilizado.
- Insertar un elemento en una tabla de símbolos:
  - Lenguajes de programación orientados a objetos: `table.insert(...)`
  - Haskell: `insert :: ... -> SymbolTable -> SymbolTable`
- Eliminar un elemento de una tabla de símbolos:
  - Lenguajes de programación orientados a objetos: `table.delete(...)`
  - Haskell: `delete :: ... -> SymbolTable -> SymbolTable`
- Actualizar la información de un elemento de una tabla de símbolos:
  - Lenguajes de programación orientados a objetos: `table.update(...)`
  - Haskell: `update :: ... -> SymbolTable -> SymbolTable`
- Determinar si un elemento se encuentra dentro de una tabla de símbolos:
  - Lenguajes de programación orientados a objetos: `table.contains(...)`
  - Haskell: `elem :: ... -> SymbolTable -> Bool` (siguiendo estándares de Haskell)
- Obtener la información de un elemento que se encuentra en la tabla de símbolos:
  - Lenguajes de programación orientados a objetos: `table.lookup(...)`
  - Haskell: `lookup :: ... -> SymbolTable -> ...`

Ejecución

Luego de realizar el análisis lexicográfico y sintáctico, de no presentarse errores relacionados con éstos, se procede a verificar errores de alcance, redeclaración y no-declaración utilizando la tabla de símbolos. Finalmente se chequea la correctitud en el uso de tipos.

En caso de encontrar errores lexicográficos y sintácticos, deben reportar sólo estos. Los errores relacionados a la verificación haciendo uso de la tabla de símbolos y errores de mal utilización de tipos deberán ser reportados juntos.

Por salida, se debe mostrar la tabla de símbolos. En caso de encontrar errores, ésta no debe ser impresa. El formato de impresión debe ser similar al ejemplo mostrado a continuación:

```
{ %a @b !c |
    { @c d |
        write c;
        (c |
            a = 10 - 4 / 2
```

```
        )
    }

    { !a |
        write b;
        { %a c d | write b }
    }
}
```

Resultado:

```
Tabla de símbolos:
En línea 1, columna 1: %a, @b, !c
    En línea 2, columna 5: @c, @d
    Em línea 9, columna 5: !a
        En línea 11, columna 9: %a, %c, %d
```

Otro ejemplo:

```
{ %i @c !b|
    (i ?
        write b;
        [i..b | write c - i; write n]
    )
}
```

Resultado:

```
# aquí puede estar la tabla de símbolos a medias hasta le primer error.
En línea 2, columna 4 instrucción condicional espera tipo `!` y obtuvo `%`.
En línea 3, columna 14 instrucción `write` espera tipo `@` y obtuvo `!.
En línea 4, columna 23 operador `-` no funciona con operadores `@` y `%`.
En línea 4, columna 36 variable `n` no existe en este alcance.
```

Análisis Teórico-Práctico

Durante la segunda etapa de desarrollo de `Lanscii`, se estudió cómo manejar gramáticas ambiguas (*no acepta la crema en la* ***u***) utilizando las propiedades que la herramienta de desarrollo escogida ofrece. Lo anterior se basa en las normas de precedencia y asociatividad de operandos. Ahora profundisaremos el estudio de estos conflictos en el contexto del análisis sintáctio -*teórico*- LR(1).

1. Sea *Gi* la gramática recursiva-izquierda `({S},{a},{S -> Sa, S ->`*`lambda`*`},S)` y sea *Gd* la gramática recursiva-derecha `({S},{a},{S -> aS, S ->`*`lambda`*`},S)`. Ambas generan el lenguaje denotado por la expresión regular *a**, i.e. el lenguaje *L(a*)*.

    a. Muestre que ambas gramáticas son LR(1) y construya sus analizadores sintácticos respectivos: Sus *autómatas de pila*.

    b. Explique brevemente la diferencia existente entre ambos analizadores en términos de espacio, esto es: el tamaño de sus tablas, la cantidad de pila utilizada para reconocer cada frase de *L(a*)* y de tiempo (cantidad de movimientos realizados por el autómata de pila para reconocer cada frase)

2. Sea *G* la gramática ({Instr},{;,IS},{Instr -> Instr ; Instr, Instr -> IS},Instr).

    a. Muestre que *G* no es una gramática LR(1), intentando construir el *autómata de prefijos viables* y determinando en cuál estado existen conflictos.

    b. Considere las dos alternativas para la eliminación de conflictos (en favor del *shift* o en favor del *reduce* en caso de un conflicto *shift-reduce*, o en favor de una producción o de otra en caso de un conflicto *reduce-reduce*). Para cada una de las alternativas de eliminación de conflicto, muestre la secuencia de reconocimiento de la frase *IS;IS;IS* indicando la secuencia de producciones reducidas. ¿A qué corresponde cada una de las alternativas: a asociar el operador **;** hacia la izquierda o hacia la derecha?

3. ¿Existe alguna forma, diferente a la de utilizar las propiedades de precedencia de operadores de la herramienta escogida, para definir la precedencia de operadores del lenguaje `Lanscii`? Explique.

Entrega

Formato de Entrega

Deben enviar un correo electrónico a **todos los preparadores** con el asunto: `[CI3725]eXgY` donde `X` corresponde al número de la entrega e `Y` al número del equipo. El correo debe incluir lo siguiente:

Un archivo **.zip** con el nombre `eXgY` siguiendo las mismas instrucciones del asunto del correo referentes a los valores de `X` e `Y`, que contenga:

- Código fuente debidamente documentado.
- En caso de utilizar *Haskell*, deben incluir un archivo Makefile o un archivo de configuración para *Cabal*. En caso de utilizar *C++* se debe incluir un archivo Makefile. Si su proyecto no compila, el proyecto no será corregido.

- Un archivo de texto con el nombre `LEEME.txt` donde **brevemente** se expliquen:
  - Decisiones de implementación
  - Estado actual del proyecto
  - Problemas presentes
  - Cualquier comentario respecto al proyecto que consideren necesario
  - Este archivo **debe** estar identificado con los nombres, apellidos y carné de cada miembro del equipo de trabajo.
- Un archivo de texto con el nombre `analisistp.txt` con la respuesta a cada una de las preguntas correspondientes a la sección de **Análisis Teórico-Práctico**. En caso de ser necesario adjuntar imágenes para contestar alguna de las preguntas, éstas deben estar en formato **.jpg** o **.png** debidamente identificadas tanto en el archivo **.zip** como en el archivo `analisisTP.txt`.

Fecha de entrega

La fecha límite de entrega del proyecto es el día **domingo 14 de junio** de 2015 (semana 10) *hasta* las **11:00pm**, entregas hechas más tarde tendrán una **penalización del 20%** de la nota, esta penalización aplica por cada día de retraso.