# Instituto Tecnológico de Costa Rica

Ingeniería en Computación


# Compiladores e intérpretes

## Tarea I: Análisis contextual

**Profesor: Óscar Víquez**

**Estudiantes:**

**Kenneth Sancho, carnet 200941125**

**Luis Alonso Vega, carnet 201042592**

**Domingo, 27 de mayo 2012**

## Análisis del contexto

Para este proyecto se esperaba agregar un analizador contextual al compilador que ya se tenía, el cual realizaba un análisis sintáctico. De esta manera se obtiene un árbol decorado que permita reconocer los tipos de algunas de sus partes y que sea posible realizar luego una generación de código “con sentido”. El programa está escrito en Java, y además el lenguaje que se evalúa es un subconjunto de este mismo lenguaje.

En este caso, se desarrolló un método de revisión del árbol abstracto que permitiera hacer una comprobación de los dos principales conceptos de esta fase: alcances y tipos. Para implementar el mencionado método, se debía aplicar la estrategia de *Visitor*, de abajo hacia arriba, al igual que en la implementación del imprimir árbol en la primera parte del proyecto.

Al final se deseaba crear una tabla de símbolos que permitiera saber qué identificadores existen (se han declarado) en el programa escrito, y qué tipos tienen para asegurarse que no se asignen tipos distintos. Se debía considerar otra serie de puntos adicionales, incluyendo:

- Arreglos de tipos simples, no de clases
- La operación suma en enteros o strings
- Identificadores heredados
- Chequeo de arreglos según su subtipo
- Métodos predefinidos (*ord, chr, len*)
- Declaración única de identificador dentro de un mismo *scope*
- Prohibido el uso de identificadores no declarados
- Chequeo de tipos en expresiones y algunos *statements*
- Revisión de número y tipos de parámetros al invocar métodos
- El retorno de función debe ser del mismo tipo que su declaración

## Soluciones e implementación

### Alcance

Este chequeo se realiza en básicamente todos los *visit* que trabajen con o referencien a un identificador. En el caso de las declaraciones se pregunta si no está declarado el identificador, tomando en cuenta el nivel actual, pues sí es posible declarar un mismo identificador pero en dos scopes del mismo nivel (o más bajos) pero en lugares distintos. Si el identificador está declarado, no se inserta y se muestra el error. Cualquier referencia a este se hace como si fuera el que ya estaba declarado desde antes.

Cuando no existe el identificador, se inserta en la tabla con la referencia a la rama del árbol, de modo que pueda ser localizado luego junto con su tipo (almacenado dentro de esta rama).

Para el caso de las asignaciones, o en el uso de expresiones, se realiza también la comprobación. En estas ocasiones, se envía un error si el identificador no está declarado. Se puede enviar que su tipo es desconocido o indeterminado, lo que por lo general causa otros errores en la evaluación de expresiones en cadena. Cuando sí está declarado el identificador usado, se busca en la tabla y se busca su tipo, para hacer la revisión que veremos en el siguiente título.

### Tipos

En cada asignación, y en otros casos (mencionados adelante) se realiza un chequeo de tipos, con el fin de asegurar que todas las variables reciban valores del tipo que fueron declaradas. Para esto se realizó un método que obtuviera el tipo de cualquier puntero relevante del árbol, por ejemplo, declaraciones de variables, declaraciones de métodos, de clases, parámetros, arreglos, tipos simples y cualquier identificador en general. De esta manera, cuando se busca un identificador, se localiza su puntero en la tabla, y luego se busca su tipo.

En los visitadores de las expresiones también se realiza este chequeo, utilizando como denotadores de tipos cadenas de caracteres de las formas “int”, “char[]”, “Fac”, etc. Las expresiones de comparaciones y aritméticas con dos operandos comprueban que las expresiones recibidas sean numéricas, con la única excepción de la suma que permite usar strings.

Al chequear los *switch*, se revisa el tipo de su expresión y se envía hacia abajo como parámetro para revisar cada *case* dentro de este. De esta forma se asegura que se pueda comparar que los tipos sean los mismos.

### Tabla de identificadores

Ya mencionada atrás, esta tabla es una implementación del modelo visto en clase, en que se utilizaba una lista simple para almacenar los identificadores declarados, con su puntero hacia el elemento del árbol donde se declaró. Se maneja con un nivel actual, el cual puede aumentar o disminuir usando los métodos de *openScope* y *closeScope*. Un método *enter* permite realizar

las inserciones, y un método *retrieve* se utiliza para las búsquedas. Se realizó una modificación para poder buscar del *scope* hacia arriba, o en toda la tabla, esto con el fin de poder hacer búsquedas desde cualquier lugar (como en el caso de búsquedas de métodos en los *message send*). Se creó un método *exists* basado en el *retrieve*, que permite saber si está declarado un identificador o no. Finalmente en vez de una función imprimir se creó un método para obtener un modelo de tabla, el cual permitirá ver los resultados de revisar el código fuente en el editor.

Cuando el programa finaliza la revisión, se muestra la tabla en el editor (ver siguiente título). De esta manera se genera ya un árbol decorado, con la suma del árbol abstracto AST y la tabla de identificadores que muestra los tipos de cada elemento.

### Editor

No hay mucho por mencionar en este apartado. El editor es el mismo utilizado en la primera fase del desarrollo del compilador, pero con la agregación de una tabla de idenficadores que muestra cada uno de ellos, junto a su nivel y su tipo.

**Nota**: Los métodos predefinidos de ord, len y chr se insertan al inicio en la tabla. Sin embargo para usarlos es necesario llamarlos desde algún identificador. Esto se debe a la declaración de la sintaxis que no permite llamadas a métodos sin instancia. Se recomienda utilizar “this” para llamarlos. Por ejemplo:

```
this.len(arr);
this.chr(239);
```

## Resultados obtenidos

- Revisión de alcances desarrollado con éxito.
- Chequeo de tipos desarrollado con éxito.
- Árbol decorado desarrollado con éxito.
- Funciones de la tabla desarrolladas con éxito.
- Mostrar la tabla en el editor desarrollado con éxito.

## Conclusiones

Lo más importante de este proyecto fue un buen análisis. Los métodos que se debían implementar no eran tan difíciles pues consistían en nada más que realizar comprobaciones. Sin embargo, se volvía complicado reconocer los tipos de datos que se manejaban, ya fuera con los tipos del AST o con nombres de clases. Al final, fue necesario implementar un único método que permitiera reconocer estos tipos y unificarlos.

La etapa escencial fue el análisis y diseño. Y de todas formas se tuvieron que corregir varios métodos que quedaron sin “calzar”, pero al final se lograron unir y quedaron funcionando bien, o al menos bastante bien.

Los chequeos que se debían hacer fueron fáciles de comprender pues el lenguaje era similar a Java, lo que permitió mayor velocidad en esta parte. Implementarlos fue cuestión de revisar los visitors y poner el código para revisar sus partes, ya fueran declaraciones, asignaciones o chequeo de tipos específicos (como lógicos o enteros).

Los dos problemas más grandes para realizar el árbol decorado fueron los envíos de mensajes, pues requerían comprobar que existiera la instancia, que el método llamado perteneciera a esa clase, que podía ser heredado, también. Los parámetros debían ser de la misma cantidad y del mismo tipo, así que todo esto provocó que se volviera la comprobación más grande. El otro problema fue el uso de *this* y los *switch*. Y los mencionamos juntos porque utilizamos la misma estrategia para resolverlos, la cual fue pasar como parámetros hacia abajo el tipo y la expresión evaluada, respectivamente.

## Bibliografía


- Oracle Documentation. Recuperado el 26 de Mayo de 2012, de HashMap: http://docs.oracle.com/javase/6/docs/api/java/util/HashMap.html
- Michael Weber. Contextual analysis. Recuperado el 20 de Mayo de 2012, en http://fmt.cs.utwente.nl/courses/vertalerbouw/sheets/vb-03-contextual-analysis-4up.pdf
- Georgia Institute of Technology. (Julio de 1999). Recuperado el 28 de Marzo de 2012, de CUP User's Manual: http://www.cs.princeton.edu/~appel/modern/java/CUP/manual.html
- SCRIBD, (sf). Swing y JFC (Java Foundation Classes), Visitado el 20 de abril del 2012 de http://es.scribd.com/trukly/d/7545519-Manual-JAVA-Swing