IIC2323 — Construcción de Compiladores
Entrega 3

Primer Semestre 2014

Descripción general

La tercera entrega tiene dos partes: terminar el análisis semántico, y comenzar con la generación de código.

Parte 1 (60 puntos): Análisis semántico II

En esta parte, se pide utilizar el AST, la tabla de símbolos, y la relación de subtipos para calcular los tipos estáticos de cada expresión. Las expresiones son los elementos sintácticos que se pueden derivar de la variable 'expression', por ejemplo las expresiones literales, llamadas a métodos e iteradores, acceso a variables, etc.

Tipos de expresiones

Atributos

El tipo de un atributo es el tipo declarado.

Variables

Si se declaró un tipo, es decir se definió de la forma
var:INT
ó
var:INT:=4,
entonces el tipo es el declarado.
Por otra parte, si se declaró con tipo implícito, o sea de la forma:
var::=33
el tipo es el tipo de la expresión que se asigna.

Métodos e iteradores

Si tiene definido un tipo de retorno, su tipo es el de retorno. De lo contrario, no tiene tipo.

new y self

El tipo es el de la clase donde se encuentra.

Create expressions, operadores binarios y unarios

Estas son llamadas a métodos, por lo que aplica el mismo criterio.

and y or

El tipo es BOOL.

Expresiones literales

El tipo es el que corresponda. Notar que a partir de esta entrega se agrego la clase predefinida STRING. Los strings literales tienen ese tipo.

Evaluación

Se debe imprimir el AST, con el mismo formato de la entrega 2, pero agregando los tipos junto a las expresiones, entre < y >. Por ejemplo, una parte del output podría ser:

```
call is_prime <BOOL>
|-- caller
    |-- binary <INT>
        |-- local a <INT>
        |-- local b <INT>
```

Además, en las asignaciones con tipo implícito se deberá agregar el tipo, por ejemplo,

```
a::=4
```

deberá imprimirse

```
assign a : INT
|-- literal INT 4 <INT>
```

En las llamadas a métodos o iteradores sin tipo de retorno, se debe imprimir VOID.

Errores semánticos

Además de lo anterior, se deben detectar errores semánticos, e imprimir mensajes indicativos. Los errores a considerar son:

Variable no existe

Si se accede a una variable que no está en el scope.
Mensaje:

```
lineno: error: ‘NAME’ undeclared.
```

donde ID se reemplaza según corresponda.

Método o iterador no definido

Si se intenta llamar a un método o iterador que no está definido en la clase que corresponda, o bien si se intenta acceder a un feature privado desde otra clase.
Mensaje:

```
lineno: error: ‘ID’ not defined.
```

Demasiados parámetros en método/iterador

Si se intenta llamar a un método o iterador con más parámetros de los definidos.
Mensaje:

```
lineno: error: too many arguments to method/iterator ‘NAME’
```

Muy pocos parámetros en método/iterador

Si se intenta llamar a un método o iterador con menos parámetros de los definidos.
Mensaje:

```
lineno: error: too few arguments to method/iterator ‘NAME’
```

Tipo incompatible en parámetro

Si el tipo de un parámetro no es conformante con el declarado.
Mensaje (2 líneas):

```
lineno: error: incompatible type for argument 2 of ‘NAME’
lineno: note: expected ‘A’ but argument is of type ‘B’
```

Tipo incompatible en asignación

Si se asigna a una variable un objeto de un tipo no conformante.
Mensaje:

```
lineno: error: incompatible types when assigning to type ‘A’ from type ‘B’
```

Tipo incorrecto en if, and, u or

Si el tipo de la expresión en un if, and, u or no es BOOL.
Mensaje:

```
lineno: error: used ‘TYPE’ type value where BOOL is required
```

Tipo de retorno incompatible

Si el tipo de la expresión que se retorna no es subtipo del declarado, o bien si se intenta retornar un valor cuando el método/iterador se definió sin retorno.

En los métodos, se deben analizar los return_statements, mientras que en los iteradores, son los yield_statements.
Mensaje:

```
lineno: error: incompatible types when returning type ‘TIPO1’ but ‘TIPO2’ was expected
```

Método puede no retornar valor (bonus)

Si un método tiene un statement_list que no termina en un return, ni en un salto incondicional, y que no es seguido por el comienzo de otro bloque básico.

Para detectar este error, se necesita construir y analizar el grafo de flujo del método.

Este error no sucederá en ningún ejemplo de evaluación, por lo que si se implementa se debe indicar en el readme.

Mensaje:

```
lineno: error: control reaches end of non-void function
```

Otros (bonus)

Cualquier otro tipo de error semántico que se considere importante. Se deben incluir los detalles en el readme.

En los mensajes, lineno corresponde al número de línea.

Si sucede un error semántico en una expresión, al imprimir el AST no se debe imprimir el tipo (ya que hay un error), y en su lugar se debe imprimir INVALID, sin < >. Por ejemplo:

```
call is_prime INVALID
|-- caller
    |-- literal STRING "no soy un número!" <STR>
```

Este mensaje se propaga, y se debe imprimir en todas las expresiones que correspondan.

Tipo SAME

Las variables, y tipos de retorno pueden tener declarado el tipo especial SAME. Este corresponde al tipo de la clase donde se utiliza, y es relevante principalmente en los subtipos e inclusiones. Por ejemplo, si una clase A define un método:

```
copy:SAME
```

y una B es subtipo de A, entonces al ejecutar el método copy de un objeto de tipo B se obtiene un objeto de tipo B. El tipo SAME no se puede utilizar para los tipos de argumentos, ni para definir subtipos o inclusiones, es decir sólo es válido para tipos de atributos, de retorno, y de variables.

Conformancia de tipos

Para algunos errores, se necesita una definición de conformancia de tipos. Para esto, se utilizarán las siguientes reglas:

- A una variable de tipo A se puede asignar un objeto de tipo B, si $A = B$, o si B es subtipo de A, por ejemplo, como INT es subtipo de STR, la asignación:

  a:STR:=5

  es correcta.

- Si un método/iterador es definido con un parámetro de tipo A, entonces puede recibir un objeto de tipo B, si B es subtipo de A.

Parte 2 (40 puntos): Generación de código

En esta parte se comenzará a generar código intermedio de CLISP. Por el momento, se considerarán las siguientes restricciones:

- Cada archivo sólo tendrá una clase.
- Las expresiones serán de los siguientes tipos: INT, STR, OUT, BOOL y CHAR.
- Los únicos statements a utilizar son declaraciones, asignaciones y retorno.
- Se utilizarán los siguiente tipos de expresiones: todas las literales, local, call, and, or y sugar (operadores binarios).

Se proveerán implementaciones de las funciones correspondientes a las clases predefinidas, y se publicará un documento con más detalles del bytecode y la máquina virtual.

Las funciones predefinidas son funciones, y no métodos, por lo que reciben como primer parámetro al "objeto" que corresponda. Por ejemplo, en la clase STRING, el método length no tiene parámetros: str.length es el largo de str, pero la función predefinida `|COMMON-LISP-USER|::|STRLEN|` requiere el string como parámetro. Similarmente, en todas las funciones se agrega el objeto como primer parámetro. En el caso de la clase OUT, por esta entrega el primer parámetro es ignorado, por lo que se recomienda utilizar NIL (o más precisamente `|COMMON-LISP-USER|::|NIL|`).

Para ejecutar el código, cada archivo compilado deberá incluir la función:

```
|COMMON-LISP-USER|::|START|
```

sin parámetros, que corresponda al método "main" del archivo original.

Ejecución del código

Para ejecutar el código con CLISP, se debe:

- Cargar las funciones predefinidas: `(load ‘‘/path/to/sather.fas’’)`,
- Cargar el archivo compilado (el output del compilador): `(load ‘‘/path/to/output/file.fas’’)`
- Ejecutar start: `(start)`

Evaluación y entrega

Al igual que las entregas anteriores, se revisará el contenido de sus repositorios al momento de la entrega, es decir el día 30 de Mayo a las 23:59.

En el directorio base del repositorio se debe incluir un script que permita compilar y ejecutar la entrega, para lo que se publicará un archivo de ejemplo.

La corrección será automática, en base a comparar el output de sus programas con los ejemplos que serán publicados en el repositorio: `https://bitbucket.org/jsnavar1/iic2323-14-examples`, utilizando un corrector similar a los usados anteriormente.

Se recomienda clonar y mantener actualizado el repositorio.