Type Inference

Luis Alejandro Lara
Carlos Alejandro Arrieta
Grupo C-312

1 Instrucciones de ejecución

Realizamos el proyecto en *Python*, utilizamos *Streamlit* como herramienta para el visual.
Para correr la aplicación hay que ejecutar $streamlit\ run\ VisualTypeinference.py$ en la terminal de *Linux* en la carpeta del proyecto.
Se debe escribir el código del lenguaje dejando una separación entre cada *token* del lenguaje, por ejemplo:
Se debe escribir *class A { } ;* en lugar de *class A{};*, de lo contrario no se efectuará correctamente el proceso de tokenización e informaremos un error.

2 Implementación

Para efectuar todo el proceso de compilación de código de *COOL* hacemos todos los procesos estudiados en clase, dígase *Tokenizacion*, *Parsing*, y luego hacemos los recorridos de *TypeCollector*, *TypeBuilder* y *TypeChecker* como aparece en las clases prácticas y conferencias estudiadas.
Para poder inferir los tipos no especificados ($a : AUTOTYPE$, por ejemplo), utilizamos un algoritmo en el que realizamos varias veces el recorrido *TypeChecker*, en cada recorrido tenemos que inferir al menos un tipo que no haya sido inferido en recorridos anteriores, en caso de que luego de un recorrido no lo hayamos hecho, significa que:

- Ya todos los tipos fueron inferidos
- Hay algún o algunos tipos que no pudimos inferir

En el 2do caso reportaremos un error informando que existen tipos que no se pudieron inferir
¿Cómo inferimos los tipos?
En nuestro proyecto se cumple la siguiente propiedad: $< type > .conforms_to(AUTOTYPE)$ *and* $AUTOTYPE.conforms_to(< type >)$ para cualquier tipo, para poder seguir lo mejor posible el recorrido *TypeChecker* a pesar de que, en cierto momento, no hayamos inferido el tipo de alguna variable, o retorno de un método.
Algunos casos de inferencia que realizamos:

- $a : AUTOTYPE < -- 1 + 1$, este caso es bastante sencillo ya que la expresión de asignación $1 + 1$ tiene un tipo bien definido, por lo que se le asigna el tipo *int* (en este caso) a la variable a.

- $a : AUTOTYPE$; $b : int <-- a + 1$;
  En este caso la variable "a" se define pero no se le asigna valor en un primer momento, pero luego a es usada en una suma, por lo que debe ser de tipo int. Para lograr asignarle este tipo, en el momento que se está evaluando la suma $a + 1$ se manda a evaluar la parte izquierda (a) y la parte derecha (1) informando que el tipo sugerido es int, de esta manera, cuando la variable a pasa a ser evaluada se entera de que el tipo sugerido para ella es int, en ese momento nos damos cuenta de que a es $AUTOTYPE$ por lo que le asignamos su tipo, int en este caso.

- $a : AUTOTYPE$; $if\ a\ then\ 1\ else\ 2\ fi$; en este caso pasa igual que en el anterior, lo mismo pasaría si a es usada como condición de un $while$, en estos casos su tipo deberá ser $bool$, pasa lo mismo si a es usada en una expresión como $a < 1$ (debe ser int).

```
f (n : AUTOTYPE) : AUTOTYPE {
        if n < 1
        then n
        else
        f( n + 1 )
}
```

En este caso intentamos utilizar el tipo de retorno de if para inferir el tipo de retorno del método f, y para inferir el tipo de n utilizamos la información de las operaciones donde es utilizada.
Como vimos anteriormente, cuando se está evaluando la cláusula del 'if' inferimos que el tipo de 'n' es 'int', por lo que el tipo de la cláusula then es 'int', pero no podemos inferir el tipo de la cláusula else
Tenemos hasta ese momento:

- Tipo de n: int
- Tipo de then: int
- Tipo de else: $AUTOTYPE$

El tipo de retorno de la expresión if tiene que ser el primer ancestro común de int y $AUTOTYPE$, en este caso podría ser cualquiera de los dos (recuerde que $int.conforms_to(AUTOTYPE)\ and\ AUTOTYPE.conforms_to(int)$), pero le damos más prioridad a los tipos distintos de $AUTOTYPE$, en este caso int, por lo que expresión if retornaría tipo int y podríamos inferir que el tipo de retorno de f es int
¿Por qué hacemos varios recorridos? ¿Sería suficiente con uno solo?
No, lo explicamos con este ejemplo:

```
class A {
        a : AUTOTYPE = f() ;
        f () : AUTOTYPE {
                1
        } ;
} ;
```

En este caso, una vez comenzamos con el primer recorrido del $TypeChecker$, no podemos inferir el tipo de a puesto que el método f tiene como tipo de retorno $AUTOTYPE$ y aún no ha sido visitado.
Una vez se visita el método f podemos inferir su tipo de retorno fácilmente puesto que la expresión que retorna es claramente un int.
Luego, en un 2do recorrido inferiremos el tipo de a a partir del tipo de f que fue inferido anteriormente.
¿Qué no podemos inferir?
Ejemplos:

```
class A {
        a : int = f(1)

        f(n : AUTOTYPE) : AUTOTYPE{
        n
        }
}
```

n tiene que ser int, pero solo utilizamos información brindada dentro del *scope* del método para inferir estas variables.

```
class A {

        F():int
        { 1}

};
class B {
        a : AUTOTYPE

        g() : int{

                a.F()
        }
}
```

En este caso a tiene que ser de tipo A, pero tampoco lo inferimos debido a que en un futuro pudiera existir otro tipo con un método con la misma signatura y nombre del método f de A.
La gramática que creamos para parsear el lenguaje se encuentra en el archivo *Grammar.py*, y utilizamos el parser creado por nosotros en el primer proyecto de la asignatura.