Segundo Proyecto de Compilación - Inferencia de Tipos para COOL

La inferencia de tipos es la capacidad de deducir, ya sea parcial o totalmente, el tipo de una expresión en tiempo de compilación. El objetivo de este proyecto es la implementación de un intérprete de *COOL* que posea inferencia de tipos mediante la adición del tipo `AUTO_TYPE`.

Inferencia de Tipos

Un programa en *COOL* no necesita especificar todas las anotaciones de tipos si son inferibles dado el contexto. Ocasionalmente se necesitan anotaciones para la desambiguación, por ejemplo, la inferencia de tipos con recursión polimórfica es indecidible. Se muestran a continuación algunos ejemplos que muestran algunos casos en los cuales es posible inferir el tipo de las expresiones y en cuáles se lanzará un error semántico.

Ejemplos

El caso más sencillo, es cuando en una declaración de variable se omite el tipo. En este caso, el tipo se infiere de la expresión de inicialización:

```
class Main inherits IO {
    main() : AUTO_TYPE {
        let x : AUTO_TYPE <- 3 + 2 in {
            case x of
                y : Int => out_string("Ok");
            esac
        }
    };
};
```

De igual forma sucede con los atributos de una clase, cuando pueden ser inferidos por el tipo de la expresión de inicialización:

```
class Point {
    x : AUTO_TYPE;
    y : AUTO_TYPE;
    init(n : Int, m : Int) : SELF_TYPE {
    {
        x <- n;
        y <- m;
    }};
};
```

Un caso más complejo es cuando se deja sin especificar el tipo de retorno de una función, pero puede ser inferido por a partir de su cuerpo:

```
# ...

    succ(n : Int) : AUTO_TYPE { n + 1 };

# ...
```

En el caso anterior, es fácil inferir el tipo de retorno de `succ` porque la expresión retorna exactamente el mismo tipo que un argumento. En estos casos, es posible incluso no especificar el tipo del argumento, ya que el operador `+` solo está definido para `Int`:

```
# ...

    succ(n : AUTO_TYPE) : AUTO_TYPE { n + 1 };

# ...
```

Sin embargo, a veces no es posible inferir el tipo de un argumento a partir de su uso dentro del cuerpo de una función. En el caso siguiente, aunque sabemos que el tipo del argumento `p` debe ser `Point` para aceptar la invocación, no se garantiza que el mecanismo de inferencia de tipos deba deducirlo (ya que en el futuro puede haber otras clases con un método translate). Dependiendo de la implementación, en estos casos se permite lanzar error semántico indicando que no fue posible inferir el tipo del argumento `p`.

```
# ...

    step(p : AUTO_TYPE) { p.translate(1,1) };

    main() : Object {
        let p : AUTO_TYPE <- new Point(0,0) in {
            step(p) # Puede lanzar error semantico
        };
    }

# ...
```

Por último, especial complejidad acarrean las funciones recursivas:

```
# ...

    fact(n : AUTO_TYPE) {
        if (n<0) then 1 else n*fact(n-1) fi
    };

# ...
```

El ejemplo anterior permite inferir simultáneamente el tipo del argumento `n` y del retorno, ya que se usa el retorno de la función recursiva en una operación `+` que solo está definida para `Int`. Sin embargo, en el ejemplo siguiente:

```
# ...

    ackermann(m : AUTO_TYPE, n: AUTO_TYPE) : AUTO_TYPE {
        if (m==0) then n+1 else
            if (n==0) then ackermann(m-1, 1) else
                ackermann(m-1, ackermann(m, n-1))
            fi
        fi
    };

# ...
```

Como el tipo de retorno no se utiliza explícitamente en una operación matemática, no es trivial deducir que su tipo de retorno es `Int`, ya que `Object` funcionaría también como tipo de retorno. En estos casos, se desea que el mecanismo de inferencia deduzca el tipo más concreto para el retorno y el tipo más abstracto para los argumentos que sea posible. Finalmente, dos funciones mutuamente recursivas:

```
# ...

    f(a: AUTO_TYPE, b: AUTO_TYPE) : AUTO_TYPE {
        if (a==1) then b else
            g(a + 1, b/2)
        fi
    }

    g(a: AUTO_TYPE, b: AUTO_TYPE) : AUTO_TYPE {
        if (b==1) then a else
            f(a/2, b+1)
        fi
    }

# ...
```

En este caso, es posible teóricamente inferir que `f` y `g` deben ambos retornar tipo `Int`, pero dada la complejidad de manejar la inferencia de tipos en más de una función a la vez, no se garantiza que sea posible deducir los tipos en este caso.