Expresiones

Examen Extraordinario de Programación - Curso 2024

Día 2

> **Advertencia**
>
> Si usted está leyendo este documento si haberlo descomprimido, ciérrelo inmediatamente y descomprima el contenido del archivo zip que recibió en el escritorio. Una vez hecho esto, reanude su trabajo desde allí. Trabajar antes de descomprimir le hará perder sus cambios, y no tendrá oportunidad de reclamar.

Se desea implementar un evaluador de expresiones para un lenguaje matemático sencillo, basado completamente en funciones n-arias. Toda expresión en este lenguaje es una función, una constante numérica, o una variable.

Por ejemplo:

```
// 3+5
sum(3,5)

// 2*(5+7)
mul(2, sum(5,7))

// 3^log_2(10)
pow(3, log(2,10))
```

Además, en este lenguaje es posible definir variables a partir de una expresión, que serán usadas en otra expresión. Por ejemplo:

```
let(x, add(2,5), sum(3,x))

// equivalente en C# A
// int x = 2+5;
// return 3+x;
```

La lista de funciones definidas es:

- `sum(x,y)`: sumar x con y.
- `sub(x,y)`: restar x con y.
- `mul(x,y)`: multiplicar x con y.
- `div(x,y)`: dividir x con y.
- `pow(x,y)`: elevar x al exponente y.
- `log(x,y)`: calcular logaritmo en base x de y.

Y la función especial `let(x, init, expr)` que calcula el valor de `init`, lo guarda en una variable nueva llamada `x`, y devuelve el resultado de evaluar `expr` con el valor de `x` definido.

Sobre las variables

Es importante destacar que en cualquier sub-expresión de un `Let` se pueden otras expresiones `Let` que pudieran tener variables con el mismo nombre. Esto no es un error, cada variable es válida en la sub-expresión donde se define, y en ninguna otra. Si una variable con el mismo nombre existe en una expresión más arriba en el árbol, se debe usar el valor de la variable más cercana.

Por ejemplo:

```
let(x, 5, mul(x, let(x, 8, sum(x,3))))
```

En esta expresión, el valor de `x` para la expresión `sum(x,3)` será `8`, mientras en la expresión `mul(x, ...)` será `5`.

De la misma forma, es posible por supuesto usar una expresión `let` en la inicialización de una variable, por ejemplo:

```
let(x, let(y,4,y+1), x)
```

El valor de `x` en este caso sería igual al valor de `y+1` que sería finalmente `5`.

Implementación

Para representar en C# una expresión de este tipo, se define una jerarquía de clases con base en la clase abstracta `Expression`, y subclases para todos los tipos de expresiones disponibles.

Por ejemplo, 3+5 se representaría como:

```
var exp = new Sum(new Constant(3), new Constant(5));
```

Y para obtener su valor:

```
double value = exp.Evaluate();
```

Y para `let(x, add(2,5), sum(3,x))` el código equivalente en C# sería:

```
var exp = new Let(
        "x",
        new Sum(new Constant(2), new Constant(5)),
        new Sum(new Constant(3), new Variable("x"))
    );
```

Usted no tiene que definir niguna clase de esta jerarquía, ya que todas las clases están definidas de antemano. Usted solamente debe implementar el método abstracto `double Evaluate()` en todas las clases, de forma que evaluar cualquier expresión devuelva el valor correcto.