El lenguaje de programación `fuu`: un análisis de diseño

Diego Munguía Molina
Ingeniería en Computación, TEC
Alajuela
dmunguia@itcr.ac.cr

Resumen

Se describen los principales componentes de diseño del lenguaje de programación ***fuu****. Un lenguaje funcional, subconjunto del español e inspirado en la notación matemática tradicional para definición de funciones.*

1. Introducción

En este artículo se documenta un análisis del diseño del lenguaje de programación **fuu**. Se analiza el lenguaje desde tres perspectivas fundamentales: estrategias de asignación de identificadores y alcances, control de flujo y tipos de datos. Finalmente se categoriza el lenguaje dentro de un conjunto de paradigmas y se recomienda una serie de escenarios de aplicación en los que el lenguaje brinda algún tipo de ventaja expresiva.

fuu es un lenguaje subconjunto del español y totalmente orientado a la notación matemática de funciones. Su nombre es un derivado del nombre `foo` clásicamente utilizado para identificar funciones de ejemplo.

2. Identificadores y alcances

fuu permite la declaración de funciones y constantes.

```
definición PI:R 3.14159.

función inc(n:Z):Z = { n + 1. }
```

Los identificadores del lenguaje pueden representar constantes, funciones o argumentos.

Los comentarios se delimitan con los caracteres “ y ” y pueden ser multilínea.

```
“este es un comentario”
“este también es
 un comentario”
```

2.1. Alcances

El alcance es léxico. Presenta dos niveles de alcance: el global, donde se definen constantes y funciones, y el local donde se definen argumentos y está delimitado por { y } en la definición de una función cualquiera. Desde cualquier alcance local es posible accesar identificadores definidos en el alcance global.

3. Control de flujo

fuu no presenta el concepto de *statement*, todos los cálculos se realizan a través de expresiones.

3.1. Expresiones

Los valores literales permiten representar números enteros y flotantes.

```
definición MEDIO:R 0.5 .
definición CERO:Z 0 .
```

También es posible representar estructuras de datos complejas: vectores, matrices y conjuntos. Las estructuras de datos son inmutables.

```
definición ORIGEN:V(Z,2) [0,0] .
definición ID:M(Z,2,2) [[1,0],[0,1]].
definición PRIMOS:C() {1,3,5,7}.
```

Los valores se pueden operar utilizando operadores aritméticos, relacionales y lógicos.

```
+ - * cociente residuo elevado = /= >
< >= <=
```

Estos operadores están sobrecargados, dependiendo del tipo de datos de los operadores realizarán diferentes operaciones. Por ejemplo:

```
definición A:C() {1,3}.
definición B:C() {2,4}.

A + B = {1,2,3,4} “verdadero”
```

En este caso el operador + representa la operación de unión de conjuntos. Si los operandos fueran enteros o reales representaría más bien la operación de adición.

En este sentido, los operadores +, – y * requieren de especial atención por su polimorfismo.

+	Adición, unión de conjuntos, disjunción lógica, construcción de conjuntos.
-	Substracción, diferencia de conjuntos, negación lógica.
*	Multiplicación, intersección de conjuntos, conjunción lógica.

El símbolo es un valor especial, es una etiqueta textual que se representa así misma, no tiene un valor o significado más allá de su propio signo o imagen.

```
definición ETIQUETA:S $uno.
definición Ns:C() {$uno,$dos,$tres}.
```

Los valores lógicos de verdad se representan con símbolos especiales.

```
(1 > 0) = $v “es verdadero”
(0 > 1) = $f “es verdadero”
```

Es posible accesar los valores de vectores y matrices utilizando notación de subíndice.

```
[7,13][1] = 13 “$v”
[[2,4],[6,8]][0][1] = 4 “$v”
```

3.2. Control de flujo

La definición de una función debe contener una sola expresión que la calcule.

```
función ident(x:Z):Z = { x. }
```

Es posible incluir diferentes expresiones en una definición de función con ramificaciones condicionales.

```
función clasificar(n:Z):S {
   $par   ; si n residuo 2 = 0.
   $impar ; si n residuo 2 /= 0.
}
```

La repetición se logra con recursividad. Esta puede ser de pila o de cola.

```
función fact(n:Z):Z = {
   1               ; si n = 0.
   n * fact(n – 1) ; si n > 0.
}

función factC(n:Z, acum:Z):Z = {
   acum                   ; si n = 0.
   factC(n – 1, acum * n) ; si n > 0.
}
```

3.3. Funciones de alto órden

Es posible definir funciones que reciban otras funciones como argumentos como mecanismo para crear complejidad.

```
función suma(a:Z,b:Z):Z = { a + b. }
función operar(a:Z,b:Z,op:(Z,Z):Z):Z=
{
   op(a,b).
}
```

El lenguaje implementa las funciones `mapear`, `reducir` y `filtrar` para trabajar sobre conjuntos.

```
función duplicar(n:Z):Z = { n * 2. }
mapear(duplicar, {1,2}) = {2,4} “$v”

reducir(+, {1,2}) = 3 “$v”

función es_par(n:Z):L = {
   n residuo 2 = 0.
}
filtrar(par, {1,2,3,4}) = {2,4} “$v”
```

4. Tipos de datos

`fuu` es un lenguaje fuertemente y estáticamente tipado. Los tipos se representan de la siguiente manera:

```
Z “enteros”
R “reales aprox. con punto flotante”
L “valores lógicos de verdad”
C() “conjunto”
V(tipo,dimensión) “vector”
M(tipo,filas,cols) “matriz”
(T1,T2,…,Tn):Tr “función”
```

Los conjuntos son colecciones desordenadas heterogéneas. En contraste, los vectores y matrices son colecciones secuenciales homogéneas, es necesario especificar el tipo de dato de los valores que contendrán.

El tipo de dato de una función se especifica con la tupla de tipos de sus argumentos y el tipo de retorno de la función.

El operador binario + se puede utilizar en un contexto donde uno de los operandos es un conjunto y el otro es un valor de cualquier tipo excepto conjunto. En este contexto la operación que se aplica es la construcción de un nuevo conjunto que contiene todos los elementos del conjunto original y el valor del operando que no es conjunto.[1]

Los vectores y matrices deben especificarse literalmente, no pueden ser construidos por partes.

5. Paradigmas

Por sus características **`fuu`** puede ser clasificado como un lenguaje funcional puro.

- Carece de efectos secundarios.
- No presenta statements.
- No presenta variables.
- Las funciones son valores de primer orden.

6. Aplicaciones

`fuu` es un lenguaje experimental, su propósito es servir de ejemplo pedagógico en el contexto de diseño de lenguajes de programación e implementación de compiladores e intérpretes.

También es experimental en el sentido de que permite explorar sus efectos como herramienta expresiva al salirse de la media en dos aspectos particulares: el uso de español como lenguaje natural del que se deriva y el uso de una notación para la definición de funciones que se acerca mucho a la notación matemática tradicional.

7. Referencias

Es importante que liste las referencias que utilizó. Recuerde que cuando no se listan referencias el trabajo se puede clasificar como plagio.

1En este caso la definición se queda corta pues no permitiría el concepto matemático de un cojunto de conjuntos.

[1] A.B. Smith, C.D. Jones, and E.F. Roberts, “Article Title”, *Journal*, Publisher, Location, Date, pp. 1-10.