MODELOS FORMALES DE COMPUTACIÓN

Semántica de continuaciones para Programación Lógica

---

1. ¿Cómo usar la semántica de continuaciones?

En PoliformaT tenéis disponibles distintos ficheros que corresponden a las semánticas de continuaciones descritas en teoría. Cada una de estas semánticas se han implementado como un programa en el lenguaje de programación Maude.

Por ejemplo, el tema 5 de programación lógica tiene asociado un fichero "Tema 5.maude" donde se incluye la implementación en Maude de la semántica descrita en teoría.

1.1. Estructura de cada fichero de Maude

Cada fichero en Maude que describe una semántica de continuaciones tiene una estructura muy simple:

- Hay diversos módulos funcionales de Maude, descritos como "`fmod Nombre is … endfm`". Pero hay un primer grupo de módulos funcionales que definen la sintaxis y un segundo grupo que define la semántica.

- La gramática BNF vista en teoría para cada lenguaje se codifica como un conjunto de definiciones de símbolos en Maude. Por ejemplo, la definición del operador de asignación en la gramática BNF

```
Exp ::= Name := Exp
```

donde `Exp` y `Name` son símbolos no terminales de la gramática y ":=" es un símbolo terminal (una palabra reservada) da lugar a la siguiente definición en Maude

```
op _:=_ : Name Exp -> Exp .
```

donde `Exp` y `Name` son tipos de datos definidos al principio del fichero con la palabra `sort` y "`_:=_`" es un símbolo en Maude.

- Cada una de las definiciones de la semántica por continuaciones se codifica también en Maude usando ecuaciones. Por ejemplo, la definición semántica de qué hacer cuando nos encontramos un número entero, es decir,

$$\text{exp}(I,\text{Env}) \curvearrowright K \rightleftharpoons \text{val}(I) \curvearrowright K$$

se codifica en Maude como una ecuación

```
eq k(exp(I, Env) -> K) = k(val(int(I)) -> K) .
```

con la pequeña diferencia de incluir el símbolo k que identifica la componente asociada a la pila de continuaciones.

1.2. Ejecutar una semántica descrita en Maude

No es necesario ser un experto programador en Maude para ser capaz de entender cómo se describe una semántica en Maude y cómo podemos escribir programas aceptados por esa semántica y ejecutarlos.

Cada fichero de Maude asociado a un tema de la asignatura incluye varios programas de ejemplo. La forma de cargar uno de esos ficheros en el entorno Maude sería

```
                     \||||||||||||||||||/
                   --- Welcome to Maude ---
                     /||||||||||||||||||\
           Maude 2.7.1 built: Jun 27 2016 16:43:23
            Copyright 1997-2016 SRI International
                  Mon Oct 29 17:27:56 2018
Maude> load /usr/usuario/MFC/Tema\ 5.maude
```

mostrando por pantalla la ejecución de todos los programas de ejemplo que incluye. Se puede cambiar de directorio con el comando "`cd`" y se puede conocer el directorio actual con el comando "`pwd`". Si haces las prácticas con Maude para Windows, las rutas de directorio son de la forma "`/cygwin/c/Users/usuario`".

El programa Maude asociado a la semántica de continuaciones del Tema 5 incluye una función `evalLog` donde se le pasa justamente el programa que deseamos ejecutar. Esta función `evalLog` se encarga de generar una configuración inicial del lenguaje con todos los componentes inicializados de forma apropiada y con el programa a ejecutar situado en la cabeza de la pila de continuaciones. Esta función `evalLog` nos devuelve las distintas sustituciones computadas. Hay que tener en cuenta que la función `evalLog` recibe en el primer argumento una lista con las variables frescas que puede utilizar durante el proceso de unificación; de esta forma se limita la búsqueda de soluciones.

Por ejemplo, dado una simple función f que recibe una lista y calcula el número de elementos de esa lista, la codificación y ejecución usando la semántica de continuaciones del Tema 5 descrita en Maude sería la siguiente:

```
Maude> search evalLog(('x1,'x2,'x3,'x4),
  let rec f(l) = if null?(l) then 0 else 1 + f(cdr(l)) in f(lvar(l))
) =>! X:[Value] .

Solution 1 (state 3)
X:[ValueList] --> (l |-> {nil}),int(0)

Solution 2 (state 8)
X:[ValueList] --> ...(l |-> {int(1),lval('x1)}),int(1)

Solution 3 (state 13)
X:[ValueList] --> ... (l |-> {int(1),lval('x1)}),int(2)

Solution 4 (state 18)
X:[ValueList] --> ... (l |-> {int(1),lval('x1)}),int(3)

Solution 5 (state 22)
```

```
X:[ValueList] --> ... (l |-> {int(1),lval('x1)}),int(4)
```

1.3. Notación

Cada semántica viene predefinida con identificadores de una única letra. Si queréis escribir un identificar (nombre de función o variable o parámetro) de más de una letra, p.ej. máximo ó fact, hay que utilizar el símbolo del acento que se encuentra al lado de la tecla del cero en el teclado español, es decir, ‘maximo ó ‘fact.

Puede ser más fácil escribir las funciones en notación infija con paréntesis “f(x,y)” en vez de “(f x y)” aunque acepta ambas.

2. Ejercicios básicos a resolver

Dada la codificación de la función de longitud de una lista mostrada anteriormente, indica con detalle el resultado de la ejecución de cada uno de los siguientes programas.

2.1. ¿Qué devuelve el siguiente programa y por qué?

```
Maude> search evalLog(('x1,'x2,'x3,'x4),
  let rec f(l) = if null?(l) then 0 else 1 + f(cdr(l))
  in f([])
) =>! X:[Value] .
```

2.2. ¿Qué devuelve el siguiente programa y por qué?

```
Maude> search evalLog(('x1,'x2,'x3,'x4),
  let rec f(l) = if null?(l) then 0 else 1 + f(cdr(l))
  in f(cons(1,cons(2,[])))
) =>! X:[Value] .
```

2.3. ¿Qué devuelve el siguiente programa y por qué?

```
Maude> search evalLog(('x1,'x2,'x3,'x4),
  let rec f(l) = if null?(l) then 0 else 1 + f(cdr(l))
  in f([1,2])
) =>! X:[Value] .
```

2.4. ¿Qué devuelve el siguiente programa y por qué?

```
Maude> search evalLog(('x1,'x2,'x3,'x4),
  let rec f(l) = if null?(l) then 0 else 1 + f(cdr(l))
  in f([ivar(a),ivar(b)])
) =>! X:[Value] .
```

2.5. ¿Qué devuelve el siguiente programa y por qué?

```
Maude> search evalLog(('x1,'x2,'x3,'x4),
  let rec f(l) = if null?(l) then 0 else 1 + f(cdr(l))
  in f(lvar(l))
) =>! VL:ValueList,int(0) .
```

2.6. ¿Qué devuelve el siguiente programa y por qué?

```
Maude> search evalLog(('x1,'x2,'x3,'x4),
  let rec f(l) = if null?(l) then 0 else 1 + f(cdr(l))
  in f(lvar(l))
) =>! VL:ValueList,int(4) .
```

2.7. ¿Qué devuelve el siguiente programa y por qué?

```
Maude> search evalLog(('x1,'x2,'x3,'x4),
  let rec f(l) = if null?(l) then 0 else 1 + f(cdr(l))
  in f(lvar(l))
) =>! VL:ValueList,int(5) .
```

2.8. ¿Qué devuelve el siguiente programa y por qué?

```
Maude> search evalLog(('x1,'x2,'x3,'x4),
  let rec f(l) = if null?(l) then 0 else 1 + f(cdr(l))
  in f(cons(1,lvar(l)))
) =>! VL:ValueList,int(5) .
```

2.9. ¿Qué devuelve el siguiente programa y por qué?

```
Maude> search evalLog(('x1,'x2,'x3,'x4),
  let rec f(l) = if null?(l) then 0 else 1 + f(cdr(l))
  in f(cons(ivar(a),lvar(l)))
) =>! VL:ValueList,int(5) .
```

2.10. ¿Qué devuelve el siguiente programa y por qué?

```
Maude> search evalLog(('x1,'x2,'x3,'x4),
  let rec f(l) = if null?(l) then 0 else 1 + f(cdr(l))
  in f(cons(1,lvar(l)))
) =>! VL:ValueList,int(6) .
```