Práctico 2 - Intérprete de Imp en Haskell

Teoría de la Computación
Universidad ORT Uruguay

Mayo 2025

En este práctico se busca codificar[1] en Haskell el *lenguaje* Imp estudiado en el curso como modelo imperativo de computabilidad. Ello incluye:

- Sintaxis abstracta.
- Reglas de evaluación de expresiones.
- Reglas de ejecución de programas.

tal como han sido descriptas en la especificación publicada.

Se pide, concretamente:

1. Definir tipos apropiados para representar los `programas`, `expresiones` y `valores` de Imp.
2. Definir el tipo de la `Memoria` y las funciones para operar sobre ella (búsqueda, actualización, alta y bajas).
3. Definir la función de evaluación de expresiones de Imp.
4. Definir la función (parcial) de ejecución de un programa de Imp.
5. Codificar en Imp embebido en Haskell los programas que se especifican a continuación, **incluyendo tests unitarios** que le permitan convecerse de que su codificación es correcta:
   - `par`: que determina si un natural dado es o no par.
   - `suma`: que calcula la suma de dos naturales.
   - `largo`: que calcula la cantidad de elementos de una lista dada.
   - `igualdadN`: que dados dos naturales `m` y `n`, determina si `m` es igual a `n`.

[1] Otro término técnico utilizado es *embeber*. En inglés se usan *to encode* y *to embed*.

- `concat`: que dadas dos listas `l1` y `l2`, retorna una nueva lista con los elementos de `l1` seguido de los elementos `l2`. (Lo que en Haskell sería la función (++).)

6. Seleccione dos programas de la parte anterior y escriba el arbol correspondiente en deducción natural para alguna entrada.

7. Se extiende la sintaxis de Imp con directivas para declaraciones e invocaciones a funciones.

$$\begin{array}{lcl} p & ::= & \cdots \mid \underline{def}\ f(\bar{x})\ \underline{returns}\ x\ \{p\} \mid f(\bar{e})\ \underline{on}\ x \\ \vdots & & \\ f & ::= & String \end{array}$$

   (a) Definiendo la categoría de los *environments* (entornos de funciones) $\Delta ::= \overline{(f, \bar{x}, x, p)}$, adaptar las reglas de la semántica actuales a efectos de contemplar la forma del nuevo juicio: $(\Delta, M) \triangleright p \triangleright (\Delta', M')$

   (b) Agregar reglas en deducción natural para contemplar las nuevas directivas de la parte (a) dentro de la semántica de Imp. Tomar en consideración que:

   - Los parámetros $\bar{x}$ de una declaración de función se consideran pasados por copia. Ejemplo:

     $$\underline{def}\ foo(x)\ \underline{returns}\ r\ \{x := 1, r := x\}; x := 0; foo(x)\ \underline{on}\ res$$

     La memoria resultante de la ejecución del programa anterior deberá ser $[(x, 0), (res, 1)]$
   - Al ejecutar $\Delta, M \triangleright f(e)\ \underline{on}\ x \triangleright \Delta, M'$, no es posible acceder a variables definidas en la memoria M dentro del programa p asociado a la función f en Δ.
   - No será posible ejecutar funciones que repitan variables (tanto de parámetros como de retorno). Ejemplos:
     $\underline{def}\ foo(x, x)\ \underline{returns}\ r\ \{\cdots\}$ repite variables en parámetros
     $\underline{def}\ foo(x, y)\ \underline{returns}\ x\ \{\cdots\}$ repite variables en parámetros y retorno.

   (c) Ajustar los data que amerite para poder extender la representación de Imp en Haskell con las nuevas directivas.

   (d) Ajustar su implementación en Haskell para la ejecución de un programa de Imp que utilice estas directivas.

   (e) Definir un programa `anyEven` que reciba como entrada tres números `x`, `y`, `z` y determine si alguno de ellos es par, utilizando funciones auxiliares adecuadas.

   (f) Comentar que cambiaría de las partes anteriores si ahora se busca que los parámetros de una función sean pasados por referencia.