Práctico 1 - Evaluador de Chi en Haskell

Teoría de la Computación
Universidad ORT Uruguay

Abril 2025

El objetivo de esta tarea es codificar[1] en Haskell el *lenguaje* χ estudiado en el curso como modelo funcional de computabilidad. Ello incluye:

- sintaxis abstracta, y
- reglas de evaluación débil,

tales como han sido descriptas en la especificación del lenguaje publicada.

Se pide, concretamente:

1. Declarar un tipo inductivo (`data`) apropiado para representar las expresiones (sintaxis abstracta) de χ.
2. Declarar tipos inductivos (`data`) apropiados para representar a los valores y fórmas canónicas débiles de χ.
3. Definir el tipo de las `sustituciones`, así como el efecto de ellas sobre expresiones χ (para esto se requiere también definir las operaciones de búsqueda y bajas).
4. ¿Es necesaria la sustitución multiple o podríamos haber usado la sustitución simple iterada múltiples veces en la definición de `sustituciones`? Encontrar un ejemplo en χ donde ambas no coinciden para convencerse de que es necesario.
5. Definir la función (parcial[2]) de `evaluación débil`.
6. Para la evaluación completa:
   (a) Definir en deducción natural el conjunto que reglas que la caracterizan.

[1] Otro término técnico utilizado es *embeber*. En inglés se usan *to encode* y *to embed*.
[2] Cuando indicamos *parcial*, nos referimos a que no actúa sobre valores y además falla en los casos así indicados en la especificación.

(b) Definir la función de `evaluación completa` en base a dichas reglas.

7. Codificar en χ puro y en χ embebido en Haskell las funciones:

   - `or`: la disyunción booleana.
   - `triple`: que dado un natural n, retorna el triple n.
   - `duplicar`: que dada una lista l, retorna l con todos sus elemtos duplicados. `Ejemplo. duplicar [1,2,3] = [1,1,2,2,3,3]`
   - `ramaC`: dado un árbol ternario, con información en los nodos, y las hojas, retorna una lista con todos los elementos de la rama central del árbol.
   - `zeros`: La lista que contiene infinitos ceros: `[0,0,0,...]`
   - `takes`: dado un natural n y una lista l, devuelve los primeros n elementos de l

8. Para la función `not` que implementa la negación booleana:

   (a) Codificarla en *chi* puro.

   (b) Escribir explicitamente la derivación en deducción natural del juicio de evalución débil para las expresiones de χ:

   - `not (False [])`
   - `not (not (True []))`

9. Extender la sintaxis BNF de χ con un nuevo caso para permitir expresiones de la forma *if e then e else e* . Luego de eso:

   (a) Escribir la (o las) nuevas reglas para la semántica en deduccion natural.

   (b) Ajustar el data de expresiones definido para la representación de χ en Haskell.

   (c) Ajustar la funcion `evaluación débil` para que contemple las nuevas reglas.

   (d) ¿La (o las) reglas definidas aplican para tres expresiones cualesquiera? Si la respuesta es negativa, justifique por qué.