Tarea 2 - Intérprete de Imp en Haskell

Teoría de la Computación
Universidad ORT Uruguay

Marzo 2022

El objetivo de esta tarea es 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 la primera instrucción de un programa.
- Reglas de ejecución completa de programas.

tal como han sido descriptas en los repartidos publicados.

Se pide, concretamente:

1. Declarar un tipo inductivo (`data`) apropiado para representar las `instrucciones` y `expresiones` (sintaxis abstracta) de *Imp*.
2. Definir el tipo de los `programas` y la `memoria`.
3. Declarar un tipo inductivo (`data`) para representar los `valores` y definir la función (parcial[2]) de `evaluación` sobre las `expresiones` de *Imp*.
4. Definir la función (parcial) de `ejecución` de la primer `instrucción` de un `programa` de *Imp*.
5. Definir la función (parcial) de `ejecución completa` de un `programa` de *Imp*.
6. Codificar en *Imp* embebido en Haskell los `programas`:
   - `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.

[1] Otro término técnico utilizado es *embeber*. En inglés se usan *to encode* y *to embed*.
[2] Cuando indicamos parcial, nos referímos a que puede fallar y no devolver un resultado.

- `igualdadN`: que dados dos naturales `m` y `n`, determina si `m` es igual a `n`.
- `fibonacci`: que dado un natural `n`, calcula el `n`-ésimo número de Fibonacci.