Parseo y Generación de Código – 2do semestre 2020
Licenciatura en Informática con Orientación en Desarrollo de Software
Universidad Nacional de Quilmes

Segundo parcial

Nota: este parcial es a libro abierto. Se permite tener cualquier material manuscrito o impreso, pero no se permite el uso de dispositivos electrónicos. El parcial se califica con una nota numérica de 1 a 10. Se requiere ≥ 4 en ambos parciales para aprobar la materia. Para promocionar se requiere nota ≥ 6 en ambos parciales y promedio ≥ 7.

Ejercicio 1. Considerar el siguiente lenguaje de expresiones, que denotan **listas de números enteros**:

```
data E = Nil        -- []               : denota la lista vacía
       | Cat E E    -- e1 ++ e2         : denota la concatenación de e1 y e2
       | Var Id     -- x                : denota la lista [v] donde v es el valor de una variable x
       | Const Int  -- n                : denota la lista [n] donde n es una constante entera
       | Add E E    -- e1 + e2          : denota la lista [n1 + n2 | n1 <- e1, n2 <- e2]
                                          es decir, elementos de la forma (n1 + n2)
                                          donde n1 viene de la lista e1 y n2 viene de la lista e2
       | For Id E E -- for x in e1 do e2 : denota la lista [e2 | x <- e1], donde x puede aparecer en e2,
                                          es decir, concatena todas las listas denotadas por e2
                                          haciendo variar x entre los elementos de e1
```

donde `type Id = String`. Por ejemplo el programa `for x in 1 ++ 2 do (for y in 10 ++ 20 do x + y)` denota la lista `[11, 21, 12, 22]`. Escribir un intérprete basado en pasaje de continuaciones con el siguiente tipo:

```
eval :: E -> Env Int -> ([Int] -> Result) -> Result
```

Una continuación es una función de tipo `[Int] -> Result`, que recibe una lista de enteros, hace algo con ella y devuelve un resultado de tipo `Result`. El intérprete recibe una expresión, un entorno que a cada variable le asigna un valor numérico y una continuación, y devuelve un resultado de tipo `Result`. No se puede asumir nada sobre la estructura del tipo `Result`.

Ejercicio 2. La siguiente gramática independiente del contexto $G = (\{S\}, \{\mathsf{reg}, :=, \text{“;”}, +\}, \mathcal{P}, S)$ representa programas escritos utilizando código de tres direcciones: $S \to \epsilon \mid \mathsf{reg} := \mathsf{reg} + \mathsf{reg}\,; S$ Extenderla con atributos heredados y/o sintetizados para calcular el conjunto de **definiciones de alcance** en cada punto del programa. Suponer que el símbolo terminal reg tiene un atributo nombre :: String que representa el nombre del registro en cuestión. Las instrucciones se numeran desde 0.

Ejercicio 3. En este ejercicio supondremos que un **polinomio** está representado con una lista de coeficientes, de tal modo que el n-ésimo coeficiente de la lista acompaña al término de grado n. Por ejemplo, la lista `[3, 2, -5, 0, 9, 7]` representa el polinomio $7X^5 + 9X^4 - 5X^2 + 2X + 3$. Escribir una función:

```
compilarPolinomio :: [Int] -> [Op]
```

que recibe una lista de coeficientes que representan un polinomio y produce código de tres direcciones para calcular el valor del polinomio *usando sólo dos registros* y almacenando el resultado en el registro r_0. Las operaciones son las siguientes:

```
type Reg = Int              -- los registros se identifican con un número entero (r0, r1, r2, ...)

data Op = Mov Reg Int       -- ri := c       : almacena una constante numérica en ri
        | Add Reg Reg Reg   -- ri := rj + rk : suma dos registros rj y rk, almacenando el resultado en ri
        | MulX Reg Reg      -- ri := rj * X  : multiplica rj por X y almacena el resultado en ri
```

Importante: se deben usar sólo dos registros (digamos `r0` y `r1`).

Ejercicio 4. Considerar el cálculo-λ extendido con una operación $\mu x.t$ (conocida como operador de punto fijo). Las reglas de tipado se extienden del siguiente modo:

$$\frac{\Gamma, x : A \vdash t : A}{\Gamma \vdash \mu x.t : A}\ \text{T-FIX}$$

Suponiendo que hay un tipo `Int` y una constante `F : Int -> Int -> Int`.

a. Proponer un tipo para completar el agujero **?1** y dar una derivación para el juicio: $\vdash \mu x.\lambda y.\, \mathtt{F}\, y\, (x\, y) : \boxed{\textbf{?1}}$.

b. Proponer un tipo para completar el agujero **?2** y dar una derivación para el juicio: $\vdash \lambda y.(\mu x.y\, (\mathtt{F}\, x)) : \boxed{\textbf{?2}}$.

Justificar todas las respuestas.