Parseo y Generación de Código – 2do semestre 2019
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 lenguaje cuyos programas están dado por el siguiente tipo de datos:

```
data Prog = Inc Id Int     -- x := x + k           : incrementa x en k
          | Seq [Prog]     -- p1; p2; ...; pn      : ejecuta secuencialmente p1, ..., pn
          | While Id Prog  -- while (x != 0) { p } : ejecuta p mientras x != 0
          | Yield Id       -- yield(x)             : produce x
```

donde `type Id = String`. Los comandos `Inc`, `Seq` y `While` tienen la semántica esperada. Las variables toman valores enteros y se suponen inicializadas en 0. A lo largo de la ejecución de un programa, cada vez que se invoca al comando `yield(x)`, el valor de la variable `x` se produce como resultado en la salida. Por ejemplo, el siguiente programa:

```
a := a + 3; while (a != 0) { a := a + (-1); b := b + 5; yield(b); yield(b) }
```

produce en la salida los valores $5, 5, 10, 10, 15, 15$. Definir la función: `eval :: Prog -> Int` que evalúa un programa y devuelve el **promedio** de los valores que produce dicho programa (haciendo la división entera). Por ejemplo, el resultado de evaluar el programa de arriba es $\frac{5+5+10+10+15+15}{6} = \frac{60}{6} = 10$.

Ejercicio 2. La siguiente gramática independiente del contexto $G = (\{S\}, \{\mathsf{reg}, :=, ";", +\}, \mathcal{P}, S)$ representa programas escritos utilizando código de tres direcciones: $S \to \epsilon \mid \mathsf{reg} := \mathsf{reg} + \mathsf{reg} \mid S; S$ Extenderla con atributos, ya sea heredados o sintetizados, para calcular el conjunto de variables vivas en cada punto del programa. Suponer que el símbolo terminal reg tiene un atributo nombre :: String que representa el nombre del registro en cuestión.

Ejercicio 3. Una variable se considera **insegura** en un punto del programa si su valor puede depender de datos provistos por el usuario. Los datos ingresados por el usuario se leen con la función `read()`. Consideraremos una variante del análisis de flujo de datos para aproximar el **conjunto de variables inseguras**. Notar que el análisis es de tipo *may–forward*. En una asignación de la forma "$x :=$ `read()`" la variable x pasa a ser insegura. En una asignación de la forma "$x := y \otimes z$" la variable x pasa a ser insegura si y es insegura o si z es insegura; en caso contrario x deja de ser insegura. Calcular el conjunto de variables inseguras en cada punto del siguiente programa:

```
1  x := read()   4  label_A:     7  z := read()     10  z := x * x
2  y := 2        5  y := y + z   8  jump label_A    11  jump label_B
3  z := y        6  label_B:     9  x := 3
```

Ejercicio 4. Considerar el cálculo-λ extendido con un tipo $[A]$ que representa el tipo de las listas de elementos de tipo A, y con los siguientes programas:

$$t, s, \ldots ::= \ldots \mid [\,] \mid (t :: s) \mid \lambda\{[\,] \mapsto t \parallel (x :: y) \mapsto s\}$$

donde $[\,]$ denota la lista vacía; $(t :: s)$ denota una lista cuya cabeza es t y cuya cola es s; y $\lambda\{[\,] \mapsto t \parallel (x :: y) \mapsto s\}$ denota la función que recibe una lista y hace *pattern matching* de tal forma que, en caso de ser vacía devuelve t, y en caso de ser no vacía devuelve s, ligando x a la cabeza de la lista y y a la cola. Las reglas de tipado se extienden del siguiente modo:

$$\frac{}{\Gamma \vdash [\,] : [A]}\ \text{T-NIL} \qquad \frac{\Gamma \vdash t : A \quad \Gamma \vdash s : [A]}{\Gamma \vdash t :: s : [A]}\ \text{T-CONS} \qquad \frac{\Gamma \vdash t : B \quad \Gamma, x : A, y : [A] \vdash s : B}{\Gamma \vdash \lambda\{[\,] \mapsto t \parallel (x :: y) \mapsto s\} : [A] \to B}\ \text{T-LLAM}$$

a. Proponer un tipo para completar el agujero **?1** y dar una derivación para el juicio:

$$f : (A \to B) \to [A] \to [B] \vdash \lambda g.\lambda\{[\,] \mapsto [\,] \parallel (x :: \ell) \mapsto g\, x :: f\, g\, \ell\} : \boxed{\textbf{?1}}$$

b. Proponer un tipo para completar el agujero **?2** y dar una derivación para el juicio:

$$f : \boxed{\textbf{?2}} \vdash \lambda\{[\,] \mapsto \lambda y.y \parallel (x :: \ell) \mapsto \lambda y.(x :: (f\, \ell\, y))\} : \boxed{\textbf{?2}}$$

Nota: las dos apariciones del agujero **?2** se deben completar con el mismo tipo.

Justificar todas las respuestas.