Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Escuela de Ciencias Exactas y Naturales
Departamento de Ciencia de la Computación
Análisis de Lenguajes de Programación

Nombre y Apellido: Legajo:

Examen Parcial 2

1. Dada la representación de listas en Sistema F:

$$\texttt{List}\ X = \forall\, R.\, (X \to R \to R) \to R \to R$$

Definir las siguientes funciones en Sistema F, suponiendo que se tienen definidas las funciones `nil`, `cons`, `map` y append.

a) $\texttt{singleton} : \forall\, X.\, X \to \texttt{List}\ X$, que construye una lista unitaria a partir de un elemento.

b) $\texttt{inits} : \forall\, X.\, \texttt{List}\ X \to \texttt{List}\ (\texttt{List}\ X)$, que dada una lista construye una lista con los comienzos de ésta. (En Haskell, por ejemplo $\texttt{inits}\ [1,2,3] = [[1],[1,2],[1,2,3]]$)

2. A continuación se presenta la sintaxis de un lenguaje de expresiones booleanas y números una semántica de paso chico.

Sintaxis:

$$\begin{array}{l} e ::= \texttt{T} \mid \texttt{F} \mid n \mid x \mid \texttt{and}\ e\ e \mid \texttt{not}\ e \mid \texttt{let}\ x = e\ \texttt{in}\ e \\ v ::= \texttt{T} \mid \texttt{F} \mid n \\ \texttt{Type} ::= \texttt{Bool} \mid \texttt{Nat} \end{array}$$

Semántica:

$$\frac{e \to e_1}{\texttt{not}\ e \to \texttt{not}\ e_1}\ \text{(E-Not1)} \qquad \texttt{not T} \to \texttt{F}\ \text{(E-Not2)} \qquad \texttt{not F} \to \texttt{T}\ \text{(E-Not3)}$$

$$\texttt{let}\ x = v\ \texttt{in}\ e \to e\,[v\,/\,x]\ \text{(E-Let1)} \qquad \frac{e \to e_2}{\texttt{let}\ x = e\ \texttt{in}\ e_1 \to \texttt{let}\ x = e_2\ \texttt{in}\ e_1}\ \text{(E-Let2)}$$

$$\texttt{and T}\ e \to e\ \text{(E-And)} \qquad \texttt{and F}\ e \to \texttt{F}\ \text{(E-And1)} \qquad \frac{e_1 \to e_2}{\texttt{and}\ e\ e_1 \to \texttt{and}\ e_2\ e}\ \text{(E-And2)}$$

a) Enunciar las propiedades de progreso y preservación.

b) Suponiendo que el lenguaje dado tiene un sistema de tipos usual, el lenguaje no cumple con la propiedad de progreso. Demostrar con un contraejemplo por qué no se cumple la propiedad.

CamScanner

3. El siguiente tipo es utilizado para definir cómputos monádicos que leen información de un entorno:

```
type Env = [(String, Bool)]
newtype Reader a = R {runR :: Env → a}
```

a) Dar la instancia de Monad para Reader.

b) Definir las operaciones de ésta mónada:

   i. ask :: Reader Env, obtiene el entorno.

   ii. local :: (Env → Env) → Reader a → Reader a, permite modificar el contenido del entorno.

c) El siguiente tipo de datos permite combinar los efectos de Reader con los efectos de la mónada Write. La cual utilizaremos para definir un evaluador de expresiones booleanas que lleva una traza de evaluación junto con la cantidad de pasos que fueron necesarios para evaluar la expresión.

```
type Traza = String
newtype ReaderWriter a = RW {runRW :: Env → (a, Traza, Int)}
```

Dar la instancia de Monad para ReaderWriter.

Para ésta mónada se definieron las siguientes operaciones:

```
addTrace :: Traza → ReaderWriter ()
addTrace s = RW (\_ → ((), s, 0))
addStep :: ReaderWriter ()
addStep = RW (\_ → ((), "", 1))
```

y se modificaron las operaciones ask y local, que ahora tienen éste tipo:

- local :: (Env → Env) → ReaderWriter a → ReaderWriter a
- ask :: ReaderWriter Env

d) El siguiente tipo de datos representa expresiones booleanas:

```
data BoolExp = T | F                          -- constantes
             | Var String                     -- variable boolena
             | Let String BoolExp BoolExp     -- define una variable local nueva
             | And BoolExp BoolExp
             | Not BoolExp
  deriving (Eq, Show)
```

Definir una función eval :: BoolExp → ReaderWriter Bool que evalúe una expresión generando una traza de evaluación para las operaciones: let , not y and y cuente la cantidad de pasos de evaluación.

e) Definir una función mostrar :: BoolExp → IO (), que evalúe una expresión en el entorno vacío y muestre en pantalla el resultado, la traza y el número de operaciones.

Por ejemplo mostrar (Let "x" T (And (Var "x") (And (Not (Var "x")) F))) debe motrar:

```
El resultado de la evaluación es False

Pasos de evaluación:
Not (Var "x") --> False
And (Not (Var "x")) F --> False
And (Var "x") (And (Not (Var "x")) F) --> False
Let "x" T (And (Var "x") (And (Not (Var "x")) F)) --> False

La expresión se evaluó en 4 pasos.
```


CamScanner