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: FERNANDO FIORI Legajo: F-3239/5

Examen Parcial 1

1. Un lenguaje, según lo visto en la materia, puede ser definido por su sintáxis, su semántica y un sistema de tipos. A continuación se presenta un lenguaje, definiendo estos componentes.

Sintaxis:

$$
\begin{array}{rl}
t & ::= \texttt{true} \mid \texttt{false} \mid \texttt{not}\ t \mid 0 \mid \texttt{suc}\ t \mid \texttt{error} \mid \texttt{catch}\ t\ t \\
v & ::= \texttt{true} \mid \texttt{false} \mid \texttt{error} \mid nv \\
nv & ::= 0 \mid \texttt{suc}\ nv \\
T & ::= \mathbb{N} \mid \mathbb{B}
\end{array}
$$

Sistema de tipos:

$$\texttt{true} : \mathbb{B} \quad \text{(T-True)}$$

$$\texttt{false} : \mathbb{B} \quad \text{(T-False)}$$

$$\texttt{error} : T \quad \text{(T-Error)}$$

$$0 : \mathbb{N} \quad \text{(T-Zero)}$$

$$\frac{t : \mathbb{N}}{\texttt{suc}\ t : \mathbb{N}} \quad \text{(T-Suc)}$$

$$\frac{t : \mathbb{B}}{\texttt{not}\ t : \mathbb{B}} \quad \text{(T-Not)}$$

$$\frac{t_1 : T \quad t_2 : T}{\texttt{catch}\ t_1\ t_2 : T} \quad \text{(T-Catch)}$$

Semántica:

$$\frac{t_1 \to t_1'}{\texttt{catch}\ t_1\ t_2 \to \texttt{catch}\ t_1'\ t_2} \quad \text{(E-Catch0)}$$

$$\frac{v \neq \texttt{error}}{\texttt{catch}\ v\ t_2 \to v} \quad \text{(E-Catch1)}$$

$$\texttt{catch error}\ t_2 \to t_2 \quad \text{(E-Catch2)}$$

$$\frac{t \to t'}{\texttt{suc}\ t \to \texttt{suc}\ t'} \quad \text{(E-Suc)}$$

$$\frac{t \to t'}{\texttt{not}\ t \to \texttt{not}\ t'} \quad \text{(E-Not0)}$$

$$\texttt{not true} \to \texttt{false} \quad \text{(E-Not1)}$$

$$\texttt{not false} \to \texttt{true} \quad \text{(E-Not2)}$$

a) Calcular, según la semántica operacional, a qué valor evalúan los términos:

i. `catch true (not true)`

ii. `suc (catch error (suc 0))`

b) Determinar si los siguientes términos están bien tipados. Si lo están, dar el árbol de derivación correspondiente.

i. `catch (catch (suc 0) 0) (suc (suc 0))`

ii. `not (catch error true)`

c) Explique brevemente cuáles son los teoremas principales (y clásicos) para demostrar la seguridad de un lenguaje según lo visto en clase.

d) Analice si los teoremas valen para el lenguaje presentado: ejemplos de ésto. Para cada teorema justifique informalmente su validez o dé un contrajemplo y proponga modificaciones para solucionar el problema.

2. La representación de los números naturales en el cálculo lambda no tipado mediante la codificación de Church tiene el problema que algunas operaciones (como el predecesor) pueden ser engorrosas. En el Sistema T de Gödel no existe este problema porque el recursor R permite el acceso al predecesor directamente.

$$
\begin{array}{l}
R\,0 \qquad f\ z = z \\
R\,(n+1)\,f\ z = f\ (R\ n\ f\ z)\ n
\end{array}
$$

a) Obtener una nueva codificación de los naturales que esté basada en R en lugar de *foldn*. Es decir, encontrar λ-expresiones para cero, sucesor y R, tal que cumplan con la especificación de más arriba.

b) Definir la función *pred* para esta codificación.

c) Dadas las siguientes expresiones, reducirlas a su forma β-normal de ser posible, o probar que no poseen forma β-normal.

   i. *pred* (*suc* ω), donde $\omega = (\lambda x.\, x\ x)\ (\lambda x.\, x\ x)$.

   ii. *suc* (*pred cero*).

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. Se representarán los tipos de un lenguaje con el siguiente tipo de datos:

```
data Type v = TVar v                    -- Variable de tipo
            | TInt                      -- Tipo Integer
            | Fun (Type v) (Type v)     -- Tipo de función
```

a) Dar una instancia de *Functor* para *Type*.

b) Para modelar la sustitución de variables se definió la siguiente instancia:

```
instance Monad Type where
  return v       = TVar v
  TVar v >>= f   = f v
  TInt   >>= f   = TInt
  Fun d r >>= f  = Fun (d >>= f) (r >>= f)
```

Es decir, que *result* corresponde a la sustitución nula (no hay sustitución de variables), y $t \gg\!= s$ corresponde al resultado de aplicar la sustitución s al término t, reemplazando cada ocurrencia de una variable v en t con el término $s\ v$.

Probar que *Type* es una mónada.

c) Definir las siguientes funciones:

i. $apply :: (a \to Type\ b) \to Type\ a \to Type\ b$, que aplique una sustitución a un tipo.

ii. $comp :: (a \to Type\ b) \to (c \to Type\ a) \to (c \to Type\ b)$, que permita componer sustituciones.

iii. $(\ggg) :: Eq\ v \Rightarrow v \to Type\ v \to (v \to Type\ v)$, tal que $v \ggg t$ sea una función que mapee la variable v con el término t y deje las demás variables sin modificar.

iv. $varBind :: Eq\ v \Rightarrow v \to Type\ v \to Maybe\ (v \to Type\ v)$, que dada una variable v y un término t, falle si v es una variable de t, o devuelva la sustitución $v \ggg t$ si v no está en t.

d) La mónada sustitución puede utilizarse para implementar de manera concisa un *algoritmo de unificación* de tipos para valores de tipo *Type* v.

Dados dos tipos $t_1, t_2 :: Type\ v$, la *unificación* de t_1 con t_2 es una sustitución de variables $\sigma :: v \to Type\ v$ que al aplicarse sobre t_1 y t_2 los hace iguales. Es decir, la unificación de t_1 y t_2 es un σ tal que

$$apply\ \sigma\ t_1 = apply\ \sigma\ t_2$$

Notar que algunos términos pueden unificarse, pero otros no. Por ejemplo, considere la siguiente tabla con los resultados de una implementación de unificación sobre ciertos tipos:

$unify\ x$	y	$= [x \mapsto y]$
$unify\ (Int \to Int)$	$(x \to Int)$	$= [x \mapsto Int]$
$unify\ (x \to y)$	$(a \to b)$	$= [x \mapsto a, y \mapsto b]$
$unify\ (x \to (y \to z))$	$((a \to b) \to c)$	$= [x \mapsto (a \to b), c \mapsto (y \mapsto z)]$
$unify\ (x \to y)$	Int	= No existe substitución que unifique
$unify\ (x \to x)$	y	$= [y \mapsto (x \to x)]$
$unify\ x$	$(x \to y)$	= No existe substitución que unifique

Notar que en el último ejemplo la unificación falla ya que, si se unificase una variable con un término que contiene dicha variable, se obtendría un tipo infinito.

Definir una función $unify :: Type\ v \to Type\ v \to Maybe\ (v \to Type\ v)$, que implemente la unificación de tipos. Utilizar la función *varBind* para el caso de unificación de una variable con un término.

Ayuda: Las funciones definidas en el apartado anterior pueden ser de utilidad. Si así lo desea, puede usar la instancia de *Monad* para *Maybe* para manejar errores:

```
instance Monad Maybe where
  return a        = Just a
  Nothing >>= f   = Nothing
  Just x  >>= f   = f x
```

2. Sean D y E dominios, $f : D \to E$ y $g : E \to D$ funciones continuas, f estricta. Probar que:

$$\bigsqcup_{i=0}^{\infty} (g \cdot f)^n \perp = fix\,(g \cdot f) = g\,(fix\,(f \cdot g)) = g \cdot \bigsqcup_{i=0}^{\infty} (f \cdot g)^n \perp$$

(1)

(1) $\bigsqcup_{i=0}^{\infty} (g \cdot f)\perp = \perp \sqsubseteq (g \cdot f)\perp \sqsubseteq (g \cdot f)((g \cdot f)\perp) \sqsubseteq \ldots$

$= \perp \sqsubseteq g(f\perp) \sqsubseteq (g \cdot f)(g(f\perp)) \sqsubseteq \ldots$

(f estricta) $= \perp \sqsubseteq g\perp \sqsubseteq (g \cdot f)(g\perp) \sqsubseteq \ldots$ ($(g \cdot f)(g\perp) = g((f \cdot g)\perp)$)

(2) $g\left(\bigsqcup_{i=0}^{\infty} (f \cdot g)\perp\right) = g \cdot (\perp \sqsubseteq f g \perp \sqsubseteq \ldots)$

(g continua) $= g\perp \sqsubseteq g((f g)\perp) \sqsubseteq \ldots$

($\perp$ es el mínimo) $= \perp \sqsubseteq g\perp \sqsubseteq g((f g)\perp) \sqsubseteq \ldots$

$g \bigsqcup_n = \bigsqcup_n g (f g)^n \perp = \bigsqcup_n g (f \cdot g)^n (f\perp)$

$= \bigsqcup_n (g f)^{n+1} \perp$