Universidad Nacional Autónoma de México

Facultad de Ciencias

Tarea Examen I

Lenguajes de Programación

Dafne Bonilla Reyes
José Camilo García Ponce
Rodrigo Aldair Ortega Venegas

Profesor: Javier Enríquez Mendoza

Ayudante: Kevin Axel Prestegui Ramos
Ayudante: Karla Denia Salas Jiménez
Ayudante Laboratorio: Ramón Arenas Ayala
Ayudante Laboratorio: Oscar Fernando Millán Pimentel

Mayo 2023

Lenguajes de Programación

Tarea Examen 1

1. En el lenguaje de programación Zoo, los nombres para variables deben empezar con el caracter 'V' seguido por una cadena cualquiera no vacía de caracteres 'o' o 'z'.

   a) Defina un juicio `ozv` tal que s `ozv` se cumpla si y solo si s es un nombre válido de variable en Zoo.

   A continuación, se definen las siguientes reglas de inferencia, cuyos esquemas serán los siguientes:

   $$\frac{}{\text{Vo ozv}}\ \text{VO} \qquad \frac{}{\text{Vz ozv}}\ \text{VZ}$$

   $$\frac{\text{Vw ozv}}{\text{Vwo ozv}}\ \text{VwO} \qquad \frac{\text{Vw ozv}}{\text{Vwz ozv}}\ \text{VwZ}$$

   b) Derive $Vozo$ `ozv` usando sus reglas.

   Usando las reglas construidas en el inciso anterior, definamos la derivación de $Vozo$ `ozv`. Esto es:

   $$\dfrac{\dfrac{\dfrac{}{\text{Vo ovz}}\ \text{VO}}{\text{Voz ovz}}\ \text{VwZ}}{\text{Vozo ozv}}\ \text{VwO}$$

   c) Enuncie el principio de inducción estructural para el juicio `ozv` y utilícelo para demostrar que: si w `ozv` entonces $\exists u \in \{o, z\}^+\ (w = Vu)$

   Por el principio de inducción estructural que enuncia que:

   Sea A una propiedad o relación definida inductivamente por un conjunto de reglas de inferencia Δ, para probar que una propiedad P es válida para todos los elementos de A basta probar que para cada regla $R \in \Delta$ de la forma

   $$\frac{x_1 A \quad \ldots \quad x_n A}{xA}$$

   se cumple que:

   Si P es válida para $x_1, \ldots, x_n$, entonces P es válida para x.

   Tomando esto en cuenta, probaremos que cada regla en el conjunto de reglas de inferencia definido anteriormente es válida. Esto es:

   Demostración. Inducción estructural sobre el lenguaje de programación Zoo.

   Si queremos probar que una propiedad P se cumple para todas las cadenas en Zoo mediante inducción estructural, basta probar que s Zoo implica s P.

Entonces, solo probaremos la validez del siguiente conjunto de reglas:

$$\frac{}{\text{Vo } P} \qquad \frac{}{\text{Vz } P}$$

$$\frac{\text{Vw } P}{\text{Vwo } P} \qquad \frac{\text{Vw } P}{\text{Vwz } P}$$

1) **Caso Base:**
   - Para VO:

     - Sea $w = Vo$.

     Notemos que existe una $u = o \in \{o, z\}^+$, por lo que podemos afirmar que $w = Vu$.
     $\therefore$ Se cumple.
   - Para VZ:

     - Sea $w = Vz$.

     Notemos que existe una $u = z \in \{o, z\}^+$, por lo que podemos afirmar que $w = Vu$.
     $\therefore$ Se cumple.

2) **Paso Inductivo:**

   - Por demostrar: Vwo y Vwz cumplen con la propiedad P.
   - Hipótesis de inducción: Vw cumple con la propiedad P.

   - Para Vwo :

     Sea $\sigma = Vwo$. Notemos que similar al caso base, el último carácter de σ, o, pertenece a $\{o, z\}^+$, por lo que solo resta probar que Vw cumple con la propiedad P, sin embargo, veamos que por hipótesis de inducción esto último se cumple, esto es, Vw cumple con la propiedad P.

     $\therefore$ Como $\sigma = Vwo$ $\mathsf{ozv} \Rightarrow \exists\, uo \in \{o, z\}^+$ tal que $\sigma = Vuo$.

   - Para Vwz :

     Sea $\tau = Vwz$. Notemos que similar al caso base, el último carácter de τ, z, pertenece a $\{o, z\}^+$, por lo que solo resta probar que Vw cumple con la propiedad P, sin embargo, veamos que por hipótesis de inducción esto último se cumple, esto es, Vw cumple con la propiedad P.

     $\therefore$ Como $\tau = Vwz$ $\mathsf{ozv} \Rightarrow \exists\, uz \in \{o, z\}^+$ tal que $\sigma = Vuz$.

$\therefore$ Si w $\mathsf{ozv} \Rightarrow \exists\, u \in \{o, z\}^+$ tal que $w = Vu$.

□

2. En muchos lenguajes de programación, las expresiones flotantes incluyen el uso de notación científica. Por ejemplo, en PASCAL la notación científica se expresa usando una letra **e**. Por ejemplo, $+9.67\mathbf{e}{-}15$ significa 9.67×10^{-15} y las expresiones flotantes son de alguna de las siguientes tres formas:

$$s.u\ s\mathbf{e}r\ s.u\mathbf{e}r$$

donde s, r son enteros signados y u es un entero no signado.

   a) Defina un juicio pfloat que genere a las expresiones flotantes de PASCAL. Observe que debe también definir juicios para enteros signados y no signados.

$$\frac{}{0\ \text{Digito}} \quad \frac{}{1\ \text{Digito}} \quad \frac{}{2\ \text{Digito}} \quad \frac{}{3\ \text{Digito}} \quad \frac{}{4\ \text{Digito}} \quad \frac{}{5\ \text{Digito}} \quad \frac{}{6\ \text{Digito}} \quad \frac{}{7\ \text{Digito}} \quad \frac{}{8\ \text{Digito}}$$

$$\frac{}{9\ \text{Digito}} \quad \frac{\text{d Digito}}{\text{d Entero}} \quad \frac{\text{n Entero} \quad \text{d Digito}}{\text{nd Entero}} \quad \frac{\text{n Entero}}{+\text{n EntSig}} \quad \frac{\text{n Entero}}{-\text{n EntSig}} \quad \frac{\text{s EntSig} \quad \text{u Entero}}{s.u\ \text{pfloat}} \quad \frac{\text{s EntSig} \quad \text{r EntSig}}{s\mathbf{e}r\ \text{pfloat}}$$

$$\frac{\text{s EntSig} \quad \text{u Entero} \quad \text{r EntSig}}{s.u\mathbf{e}r\ \text{pfloat}}$$

b) Utilice su definición para dar una derivación de $+9.67\mathbf{e}{-15}$.

$$\frac{\dfrac{\dfrac{\dfrac{\checkmark}{9\ \text{Digito}}}{9\ \text{Entero}}}{+9\ \text{EntSig}} \quad \dfrac{\dfrac{\dfrac{\checkmark}{6\ \text{Digito}}}{6\ \text{Entero}} \quad \dfrac{\checkmark}{7\ \text{Digito}}}{67\ \text{Entero}} \quad \dfrac{\dfrac{\dfrac{\dfrac{\checkmark}{1\ \text{Digito}}}{1\ \text{Entero}} \quad \dfrac{\checkmark}{5\ \text{Digito}}}{15\ \text{Entero}}}{-15\ \text{EntSig}}}{+9.67\mathbf{e} - 15\ \text{pfloat}}$$

3. Un proceso estándar en la implementación de compiladores es el llamado plegado de constantes (*constant folding*) que consiste en reconocer y evaluar expresiones constantes en tiempo de compilación en vez de computar los resultados en tiempo de ejecución, esto incluye simplificaciones usando propiedades aritméticas. Por ejemplo, la expresión $(2+3+y)*(x+4*5)$ se simplifica a $(5+y)*(x+20)$, la expresión $2*x+0$ se simplifica en $2*x$ y la expresión $1*(x+2*z)$ se simplifica en $x+2*z$. Considere el siguiente lenguaje EAs de expresiones aritméticas simples.

$$e ::= x \mid n \mid e+e \mid -e \mid e*e \mid (e)$$

a) Defina una función `cfold :: EAs -> EAs` que realice el proceso de plegado de constantes. Puede utilizar pseudocódigo de Haskell.

A continuación, se define la función `cfold`:

```haskell
instance Show EAs where
show (Var x) = x
show (Const n) = show n
show (Add e1 e2) = "(" ++ show e1 ++ " + " ++ show e2 ++ ")"
show (Sub e) = "-(" ++ show e ++ ")"
show (Mult e1 e2) = "(" ++ show e1 ++ " * " ++ show e2 ++ ")"
show (Brack e) = "(" ++ show e ++ ")"

-- Definition of the data type for simple arithmetic expressions
data EAs =  Var String | Const Int | Add EAs EAs | Sub EAs
            | Mult EAs EAs | Brack EAs

-- Function that fold constants
cfold :: EAs -> EAs
cfold (Var x) = Var x
cfold (Const n) = Const n
cfold (Add e1 e2) = case (cfold e1, cfold e2) of
    (Const 0, e2') -> e2'
    (e1', Const 0) -> e1'
    (Const n1, Const n2) -> Const (n1 + n2)
    (e1', e2') -> Add e1' e2'
cfold (Sub e) = case cfold e of
    Const n -> Const (-n)
    e' -> Sub e'
cfold (Mult e1 e2) = case (cfold e1, cfold e2) of
    (Const n1, Const n2) -> Const (n1 * n2)
```

```
    (Const 0, _) -> Const 0
    (_, Const 0) -> Const 0
    (Const 1, e) -> e
    (e, Const 1) -> e
    (e1', e2') -> Mult e1' e2'
cfold (Brack e) = Brack (cfold e)
```

b) Verifique que su definición es correcta mediante el cómputo de `cfold ((5 + 2) + x * 1)`.

Para verificar que la definición de `cfold` es correcta usando este ejemplo, ejecutaremos lo siguiente:

```
ghci> cfold ((Const 5 `Add` Const 2) `Add` Var "x" `Mult` Const 1)
```

Lo cual nos devuelve como resultado: `(7 + x)`.

c) Defina el intérprete denotativo para EAs y demuestre su corrección con respecto al plegado de constantes, es decir, demuestre que para cualquier expresión e se cumple que

$$eval\ s\ e = eval\ s\ (cfold\ e)$$

A continuación, se define el intérprete denotativo para EAs:

```haskell
-- Definition of the variable environment type
type EnVar = [(String, Int)]

-- Denotative interpretation function
eval :: EnVar -> EAs -> Int
eval s (Var x) = case lookup x s of
  Just n -> n
  Nothing -> error ("Variable inválida: " ++ x)
eval _ (Const n) = n
eval s (Add e1 e2) = eval s e1 + eval s e2
eval s (Sub e) = -eval s e
eval s (Mult e1 e2) = eval s e1 * eval s e2
eval s (Brack e) = eval s e
```

Por último, haremos formalmente la demostración por inducción estructural sobre las expresiones del lenguaje EAs, probando que para cualquier expresión e, $eval\ s\ e$ es igual a $eval\ s\ (cfold\ e)$ para un entorno de variables s dado.

Demostración. Inducción estructural sobre las expresiones del lenguaje EAs

1) **Caso Base:**

Veamos que para variables y constantes tenemos que `eval s (Var x) = lookup x s` y que `eval s (Const n) = n`. Además, también recordemos que la función `cfold` no altera estas expresiones.
∴ `eval s (Var x) = eval s (cfold (Var x))` y `eval s (Const n) = eval s (cfold (Const n))`

2) **Paso Inductivo:**

- Por demostrar: La condición se cumple para todas las operaciones.
- Hipótesis de inducción: Supongamos que para una expresión `e` arbitraria, se cumple que `eval s e = eval s (cfold e)`.

Hagamos esto por casos:

- Para `Add`:

  Tenemos que, por definición, `eval s (Add e1 e2) = eval s e1 + eval s e2` y que `eval s (cfold (Add e1 e2)) = eval s (cfold e1 + cfold e2)`.

Ahora bien, por H.I tenemos que `eval s (cfold e1) = eval s e1`, y lo mismo para `e2`. Entonces, `eval s (cfold e1 + cfold e2) = eval s (e1 + e2)`.

$\therefore$ `eval s (Add e1 e2) = eval s (cfold (Add e1 e2))`.

- Para `Sub`:

  Tenemos que, por definición, `eval s (Sub e) = -eval s e` y que `eval s (cfold (Sub e)) = -eval s (cfold e)`.

  Ahora bien, por H.I tenemos que `eval s (cfold e) = eval s e`. Entonces, `eval s (Sub e) = eval s (cfold (Sub e))`.

  $\therefore$ `eval s (Sub e) = eval s (cfold (Sub e))`.

- Para `Mult`:

  Tenemos que, por definición, `eval s (Mult e1 e2) = eval s e1 * eval s e2` y que `eval s (cfold (Mult e1 e2)) = eval s (cfold e1 * cfold e2)`.

  Ahora bien, por H.I tenemos que `eval s (cfold e1) = eval s e1`, y lo mismo para `e2`. Entonces, `eval s (cfold e1 * cfold e2) = eval s (e1 * e2)`.

  $\therefore$ `eval s (Mult e1 e2) = eval s (cfold (Mult e1 e2))`.

- Para `Brack`:

  Tenemos que, por definición, `eval s (Brack e) = eval s e` y que `eval s (cfold (Brack e)) = eval s (cfold e)`, debido a que los paréntesis no alteran el valor de la expresión dentro de ellos.

  Ahora bien, por H.I tenemos que `eval s (cfold e) = eval s e`. Entonces, `eval s (Brack e) = eval s (cfold (Brack e))`.

  $\therefore$ `eval s (Brack e) = eval s (cfold (Brack e))`.

□

4. A continuación se define la sintaxis concreta de un lenguaje funcional muy simple.

$$e ::= x \mid n \mid e_1 e_2 \mid \mathbf{fun}(x) \Rightarrow e$$

En donde el primer constructor representa las variables del lenguaje, el segundo números naturales, el tercero aplicación de función y el último la definición de funciones.

a) Traduce la gramática anterior a una definición inductiva con reglas de inferencia.

$$\frac{x \text{ Identificador}}{x \text{ Variable}} \quad \frac{x \text{ Variable}}{x \text{ E}} \quad \frac{n \text{ Natural}}{n \text{ E}} \quad \frac{e_1 \text{ E } e_2 \text{ E}}{e_1 e_2 \text{ E}} \quad \frac{e \text{ E}}{\mathbf{fun}(x) \Rightarrow e \text{ E}}$$

b) Diseña una sintaxis abstracta apropiada para este lenguaje.

$$\frac{x \text{ Identificador}}{x \text{ asa}} \quad \frac{\text{n}\in\text{Naturales}}{\text{num[n] asa}} \quad \frac{\text{t}_1 \text{ asa } \text{t}_2 \text{ asa}}{\text{apli}(\text{t}_1, \text{t}_2) \text{ asa}} \quad \frac{\text{t asa}}{\text{fun}(x.\text{t}) \text{ asa}}$$

c) Escribe las reglas para la relación de análisis sintáctico ($\longleftrightarrow$) del lenguaje.

$$\frac{x \text{ Variable } x \text{ Indentificador}}{x \text{ E} \longleftrightarrow x \text{ asa}} \quad \frac{n \text{ Natural n}\in\text{Naturales}}{n \text{ E} \longleftrightarrow \text{num[n] asa}} \quad \frac{e_1 \text{ E} \longleftrightarrow \text{t}_1 \text{ asa } e_2 \text{ E} \longleftrightarrow \text{t}_2 \text{ asa}}{e_1 e_2 \text{ E} \longleftrightarrow \text{apli}(\text{t}_1, \text{t}_2) \text{ asa}}$$

$$\frac{e \text{ E} \longleftrightarrow \text{t asa}}{\mathbf{fun}(x) \Rightarrow e \text{ E} \longleftrightarrow \text{fun}(x.\text{t}) \text{ asa}}$$

d) Diseña un algoritmo de sustitución para este lenguaje.

Primero veamos como obtendremos las variables libres de una expresión. $L(\text{t})$ serán las variables libres de t. Entonces:

- $L(x) = \{x\}$

- $L(n) = \{\}$
- $L(\text{apli}(\text{t}_1, \text{t}_2)) = L(\text{t}_1) \cup L(\text{t}_2)$
- $L(\text{fun}(\text{t})) = L(\text{t})$
- $L(x.\text{t}) = L(\text{t}) \setminus \{x\}$

Ahora bien, el algoritmo de sustitución será el siguiente:
$x[x := e] = e$, $z[x := e] = z$
$num[n][x := e] = n$
$apli(\text{t}_1,\text{t}_2)[x := e] = apli(\text{t}_1[x := e]$
$\text{t}_2[x := e])$
$fun(\text{t})[x := e] = fun(\text{t}[x := e])$
$(z.\text{t})[x := e] = z.(\text{t}[x := e])$ si $x \neq z$ y $z \notin L(e)$
$(z.\text{t})[x := e]$ =indefinido si $z \in L(e)$
$(z.\text{t})[x := e] = z.\text{t}$ si $z = x$ y $z \notin L(e)$

e) La relación de α-equivalencia en este lenguaje se da respecto al operador de definición de función **fun**, se dice que dos expresiones son α-equivalentes si solo difieren en el nombre de la variable del parámetro de la función. Por ejemplo:

$$\textbf{fun}(x) \Rightarrow x \equiv_\alpha \textbf{fun}(y) \Rightarrow y$$

Demuestra que $\equiv_\alpha$ es una relación de equivalencia.

Para esto demostraremos 3 cosas
Primero la reflexividad, sea $\textbf{fun}(x) \Rightarrow x$ una expresión cualquiera, notemos que $\textbf{fun}(x) \Rightarrow x$ y $\textbf{fun}(x) \Rightarrow x$ no difieren en el nombre de la variable del parámetro (son la misma expresión) entonces por definición de $\equiv_\alpha$ tenemos que $\textbf{fun}(x) \Rightarrow x \equiv_\alpha \textbf{fun}(x) \Rightarrow x$
Ahora la simetría, sean $\textbf{fun}(x) \Rightarrow x$ y $\textbf{fun}(y) \Rightarrow y$ dos expresiones tales que $\textbf{fun}(x) \Rightarrow x \equiv_\alpha \textbf{fun}(y) \Rightarrow y$, con esto sabemos que solo difieren en el nombre de la variable del parámetro, por lo tanto usamos la definición de $\equiv_\alpha$ y podemos concluir que $\textbf{fun}(y) \Rightarrow y \equiv_\alpha \textbf{fun}(x) \Rightarrow x$
Por ultimo la transitividad, sean $\textbf{fun}(x) \Rightarrow x$, $\textbf{fun}(y) \Rightarrow y$ y $\textbf{fun}(z) \Rightarrow z$ tres expresiones tales que $\textbf{fun}(x) \Rightarrow x \equiv_\alpha \textbf{fun}(y) \Rightarrow y$ y $\textbf{fun}(y) \Rightarrow y \equiv_\alpha \textbf{fun}(z) \Rightarrow z$, entonces por definición de $\equiv_\alpha$ sabemos que $\textbf{fun}(x) \Rightarrow x$ y $\textbf{fun}(y) \Rightarrow y$ solo difieren en el nombre de la variable del parámetro, lo mismo con $\textbf{fun}(y) \Rightarrow y$ y $\textbf{fun}(z) \Rightarrow z$, por lo tanto $\textbf{fun}(x) \Rightarrow x$ y $\textbf{fun}(z) \Rightarrow z$ solo difieren en el nombre de la variable del parámetro y concluimos que $\textbf{fun}(x) \Rightarrow x \equiv_\alpha \textbf{fun}(z) \Rightarrow z$
Con lo que vimos arriba podemos concluir que $\equiv_\alpha$ es de equivalencia $\square$

5. Juanito Alimaña quiere definir un lenguaje que le permita controlar un robot con movimientos y funcionalidades muy simples. El robot se mueve sobre una cuadrícula siguiendo las instrucciones especificadas por el programa. Al inicio el robot se encuentra en a coordenada $(0, 0)$ y viendo hacia el norte. El programa consiste en una secuencia posiblemente vacía de los comandos **move** y **turn** separados por punto y coma, cada comando tiene el siguiente funcionamiento:

   - **turn** hace que el robot dé un giro de 90 grados en el sentido de las manecillas del reloj.
   - **move** provoca que el robot avance una casilla en la dirección hacía la que está viendo.

   Un ejemplo de un programa válido es:

   ```
   move;turn;move;turn;turn;turn;move
   ```

   Al final del programa el robot termina en la casilla $(2, 1)$. La primera entrada de la coordenada indica la posición vertical, mientras que la segunda es la posición horizontal. Juanito nos pidió definir la semántica operacional de paso pequeño para el lenguaje que controla al robot. Para esto responde las siguientes preguntas formalmente, para diseñar un sistema de transición:

a) Determina el conjunto de estados.

$$\frac{}{\text{(0,0,norte,w) estado}} \quad \frac{}{\text{(x,y,z,w) estado}}$$

b) Identifica los estados iniciales y finales del sistema de transición.

$$\frac{\text{(0,0,norte,w) estado}}{\text{(0,0,norte,w) inicial}} \quad \frac{\text{(x,y,z,0) estado}}{\text{(x,y,z,0) final}} \quad \frac{\text{(0,0,norte,0) estado}}{\text{(0,0,norte,0) final}}$$

c) Define la función de transición $\rightarrow_R$ que indique como se debe transitar entre los estados del sistema. De tal forma que defina una semántica operacional de paso pequeño.

$$\frac{\text{(0,0,norte,w)} \rightarrow \text{(1,0,norte,w-1)}}{\text{move} \rightarrow \text{(1,0,norte,w-1)}} \quad \frac{\text{(0,0,norte,w)} \rightarrow \text{(0,0,este,w-1)}}{\text{turn} \rightarrow \text{(0,0,este,w-1)}}$$

$$\frac{\text{(x,y,norte,w)} \rightarrow \text{(x,y,este,w-1)}}{\text{(x,y,norte,w);turn} \rightarrow \text{(x,y,este,w-1)}} \quad \frac{\text{(x,y,este,w)} \rightarrow \text{(x,y,sur,w-1)}}{\text{(x,y,este,w);turn} \rightarrow \text{(x,y,sur,w-1)}}$$

$$\frac{\text{(x,y,sur,w)} \rightarrow \text{(x,y,oeste,w-1)}}{\text{(x,y,sur,w);turn} \rightarrow \text{(x,y,oeste,w-1)}} \quad \frac{\text{(x,y,oeste,w)} \rightarrow \text{(x,y,norte,w-1)}}{\text{(x,y,oeste,w);turn} \rightarrow \text{(x,y,norte,w-1)}}$$

$$\frac{\text{(x,y,norte,w)} \rightarrow \text{(x+1,y,norte,w-1)}}{\text{(x,y,norte,w);move} \rightarrow \text{(x+1,y,norte,w-1)}} \quad \frac{\text{(x,y,sur,w)} \rightarrow \text{(x-1,y,sur,w-1)}}{\text{(x,y,sur,w);move} \rightarrow \text{(x-1,y,sur,w-1)}}$$

$$\frac{\text{(x,y,este,w)} \rightarrow \text{(x,y+1,este,w-1)}}{\text{(x,y,este,w);move} \rightarrow \text{(x,y+1,este,w-1)}} \quad \frac{\text{(x,y,oeste,w)} \rightarrow \text{(x,y-1,oeste,w-1)}}{\text{(x,y,oeste,w);move} \rightarrow \text{(x,y-1,oeste,w-1)}}$$

$$\frac{\text{(x,y,z,0)} \nrightarrow}{\text{(x,y,z,0)} \rightarrow \text{(x,y,z,0)}}$$

d) Muestra paso a paso la ejecución del programa

> **move;turn;move;turn;turn;turn;move**

usando la relación $\rightarrow_R$ y partiendo del estado inicial correspondiente.

move;turn;move;turn;turn;turn;move

$\rightarrow (1, 0, norte, 6); turn; move; turn; turn; turn; move$

$\rightarrow (1, 0, este, 5); move; turn; turn; turn; move$

$\rightarrow (1, 1, este, 4); turn; turn; turn; move$

$\rightarrow (1, 1, sur, 3); turn; turn; move$

$\rightarrow (1, 1, oeste, 2); turn; move$

$\rightarrow (1, 1, norte, 1); move$

$\rightarrow (2, 1, norte, 0)$