Universdad Nacional Autónoma de México

Lenguajes de Programacion
Examen Parcial I

- Edgar Montiel Ledesma
  317317794
- Carlos Daniel Cortes Jimenez
  420004846

---

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

   a) Defina un juicio ozv tal que s ozv se cumpla si y solo si s es un nombre valido de variable en Zoo.
   ozv(s) $\Leftrightarrow$ (s comienza con V) $\wedge$ (s contiene al menos un o $\vee$ z despues de V)
   Donde:

   - s representa la cadena que se esta evaluando.
   - $\Leftrightarrow$ "si y solo si"
   - (s comienza con V) es verdadero si el primer carácter de la cadena s es V.
   - (s contiene al menos un o $\vee$ z despues de V) es verdadero si la cadena s contiene al menos un carácter o $\vee$ z después del primer caracter V.

   Teniendo la siginete gramatica

   $E ::= Vs$
   $s ::= os \mid zs \mid o \mid z$

   En forma norlmal de Chomsky tenemos

   S $\rightarrow$ S'x
   S' $\rightarrow$ V
   O $\rightarrow$ o
   Z $\rightarrow$ z
   x $\rightarrow Ox \mid Zx \mid o \mid z$

   Las reglas de inferencia serian

   $$\frac{}{z \text{ zoo}}\text{r1} \quad \frac{}{o \text{ zoo}}\text{r2} \quad \frac{o \text{ zoo} x \text{ zoo}}{Ox \text{ zoo}}\text{r3} \quad \frac{z \text{ zoo} x \text{ zoo}}{Zx \text{ zoo}}\text{r4} \quad \frac{S' \text{ zoo} x \text{ zoo}}{S'x \text{ zoo}}\text{r5}$$

   b) Derive V ozo ozv usando sus reglas.

   $$\frac{\frac{}{V \text{ zoo}} \quad \frac{\frac{}{o \text{ zoo}}\text{r2} \quad \frac{}{z \text{ zoo}}\text{r1}}{ozz\text{oo}}\text{r3}}{Voz \text{ zoo}}\text{r5}$$

c) Enuncie el principio de induccion estructural para el juicio ozv y utilicelo para demostrar que: si w ozv entonces $\exists u \in o, z + (w = Vu)$

Si w = "V", entonces ∃u ∈ o, z (w = V u).

Paso de la inducción: Supongamos que tenemos una cadena w que es un nombre válido de variable en el lenguaje Zoo, es decir, ozv(w) es verdadero. Queremos demostrar que para cualquier cadena más compleja wq́ue se pueda formar al agregar un carácter óó źál final de w, la propiedad sigue siendo verdadera. Entonces, consideremos dos casos:

Caso 1: Si agregamos óál final de w para formar w; entonces podemos decir que w≕ w ó; Dado que w ya es un nombre válido de variable, podemos tomar u igual a w y todavía se cumplirá la propiedad:

Si w≕ w ó; entonces ∃u ∈ o, z (w≕ V u).

Caso 2: Podemos decir que w' = w 'z' si agregamos "z.ªl final de w para formar w'. Dado que w ya es un nombre válido de variable, podemos tomar u igual a w y aún se cumplirá la propiedad:

Si w≕ w ź; entonces ∃u ∈ o, z (w≕ V u).

En ambos casos demostramos que si la propiedad es verdadera para w, también es verdadera para w', donde w' se construye agregando .º.º "z.ªl final de w.

Por lo tanto la propiedad es verdadera para todas las cadenas w, que son nombres de variable válidos en el lenguaje Zoo.

2. En muchos lenguajes de programacion las expresiones flotantes incluyen el uso de notacion cientıfica. Por ejemplo en Pascal la notacion cientıfica se expresa usando una letra e. Por ejemplo $+9.67e{-}15$ significa $9.67 \times 10 - 15$ y las expresiones flotantes son de alguna de las siguientes tres formas:

s.u s e r s.u 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 tambien definir juicios para enteros signados y no signados. Teniendo la siginete gramatica

$PF ::= SI'.'UI'e'SI$
$SI ::=' +'UInt|' -' UInt|UInt$
$UInt ::= Nu|NuUInt$
$Nu ::=' 0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'$

En forma norlmal de Chomsky

$PF ::=' +'UInt'.'UInt'e'-UInt\,|-UInt'.'UInt'e'-UInt\,|UInt'.'UInt'e'-UInt\,|'+'UInt'.'UInt'e'UInt\,|-UInt'.'UInt'e'UInt\,|UInt'.'UInt'e'UInt$

$SI ::= '+'UInt \,|\, '-'UInt \,|\, UInt$
$UInt ::= Nu \,|\, NuUInt$
$Nu ::= '0' \,|\, '1' \,|\, '2' \,|\, '3' \,|\, '4' \,|\, '5' \,|\, '6' \,|\, '7' \,|\, '8' \,|\, '9'$

b) Utilice su definicion para dar una derivacion de +9.67e−15.

PF ::= '+' UInt '.' UInt 'e' '-' UInt
PF ::= '+' UInt '.' UInt 'e' '-' UInt (Utilizamos la producción PascalFloat)
'+' UInt '.' UInt 'e' '-' UInt (Elegimos la rama de la producción PascalFloat)
'+' '9' '.' UInt 'e' '-' UInt (Elegimos UnsignedInt '9')
'+' '9' '.' '67' 'e' '-' UInt (Elegimos UnsignedInt '67')
'+' '9' '.' '67' 'e' '-' '15' (Elegimos UnsignedInt '15')

La derivación muestra cómo la expresión '+9.67e-15' se deriva utilizando la gramática en forma normal de Chomsky.

Sugerencia: siempre es util definir primero una gramatica libre de contexto adecuada y transformarla en el juicio deseado.

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

$$e ::= x|n|e+e|-e|e*e|(e)$$

a) Defina una funcion $cfold :: EAs \to EAs$ que realize el proceso de plegado de constantes. Puede utilizar pseudocodigo de Haskell.

```
cfold :: EAs -> EAs
cfold (x) = x
cfold (n) = n
cfold (e1 + e2) | isNumber e1 && isNumber e2 = e1 + e2
                | otherwise = cfold e1 + cfold e2
cfold (-e) | isNumber e = -e
           | otherwise = -cfold e
cfold (e1 * e2) | isNumber e1 && isNumber e2 = e1 * e2
                | otherwise = cfold e1 * cfold e2
cfold (e) = e

isNumber :: EAs -> Bool
isNumber (n) = True
isNumber (_) = False
```

b) Verifique que su definicion es correcta mediante el computo de $cfold((5+2)+x*1)$
Verificamos que la definicion de $cfold$ es correcta:
Evaluamos primero $cfold((5+2) \rightarrow cfold(7)$ dado que 7 es una constante se simplifica a 7.

Ahora evealuamos $cfold(x*1) \rightarrow cfold(x)$ ya que 1 es una constante, y al multiplicar por 1 obtendremos el mismo valor por lo que la expresión se simplifica a solo x.

Por lo que el resultado de aplicar $cfold$ a $(5+2)+x*1$ es $7+x$.

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

$$evalse = evals(cfolde)$$

```
eval :: EVar -> EAs -> Integer
eval s (x) = s x
eval s (n) = n
eval s (e1 + e2) = eval s e1 + eval s e2
eval s (-e) = -eval s e
eval s (e1 * e2) = eval s e1 * eval s e2
```

Ahora demostramos la corrección del intérprete con respecto al plegado de constantes.

Demostración: Induccion estructural sobre la expresión e.

Casos Base:

- Si e es una variable x, en ambos lados de la expresión son iguales, en $eval\ s\ e$ devuelve el valor esperado x, así como en $eval\ s\ (cfold\ e)$ tambien devuelve x, ya que $cfold$ no hace ninguna simplificación. Por lo tanto $eval\ s\ e = eval\ s\ (cfold\ e)$.

- Si e es una constante n, asi como en el caso anterior ambas partes serán iguales, debido a que $cfold$ no altera constantes, por lo que en ambos lados devuelve n, entonces $eval\ s\ e = eval\ s\ (cfold\ e)$.

Hipótesis de Inducción(H.I.): Para una expresión e arbitraria y estado s, se cumple que $eval\ s\ e = eval\ s\ (cfold\ e)$.

Paso Inductivo:

- Suma(+): Supongamos que se cumple la H.I. para $e1$ y $e2$, esto es $eval\ s\ e1 = eval\ s\ (cfold\ e1)$ y $eval\ s\ e2 = eval\ s\ (cfold\ e2)$.Consideremos $e = e1 + e2$, para este caso $eval\ s\ (e1 + e2)$ realiza la suma de los valores $e1$ y $e2$ que seria $eval\ s\ e1 + eval\ s\ e2$.Por lo que se tiene $eval\ s\ (e1 + e2) = eval\ s\ e1 + eval\ s\ e2$.Hacemos los mismo para $eval\ s\ (cfold\ e1 + cfold\ e2)$, e cual realiza la suma de los valores $cfold\ e1$ y $cfold\ e2$ que sería $eval\ s\ (cfold\ e1) + eval\ s\ (cfold\ e2)$. Por H.I. tenemos que $eval\ s\ (cfold\ e1) = eval\ s\ e1$, a su vez tambien se tiene que $eval\ s\ (cfold\ e2) = eval\ s\ e2$. Por lo tanto $eval\ s\ (cfold\ e1 + cfold\ e2) = eval\ s\ e1 + eval\ s\ e2$.Se demuestra que $eval\ s\ (e1 + e2) = eval\ s\ (cfold\ e1 + cfold\ e2)$.

- Multiplicación($*$): Analogamente al caso de la suma, supongamos que se cumple la H.I. para $e1$ y $e2$, esto es $eval\ s\ e1 = eval\ s\ (cfold\ e1)$ y $eval\ s\ e2 = eval\ s\ (cfold\ e2)$.Consideremos $e = e1 * e2$, para este caso $eval\ s\ (e1 * e2)$ realiza el producto de los valores $e1$ y $e2$ que seria $eval\ s\ e1 * eval\ s\ e2$.Por lo que se tiene $eval\ s\ (e1 * e2) = eval\ s\ e1 + eval\ s\ e2$.Hacemos los mismo para $eval\ s\ (cfold\ e1 + cfold\ e2)$, el cual realiza el producto de los valores $cfold\ e1$ y $cfold\ e2$ que sería $eval\ s\ (cfold\ e1) * eval\ s\ (cfold\ e2)$. Por H.I. tenemos que $eval\ s\ (cfold\ e1) = eval\ s\ e1$, a su vez tambien se tiene que $eval\ s\ (cfold\ e2) = eval\ s\ e2$. Por lo tanto $eval\ s\ (cfold\ e1 * cfold\ e2) = eval\ s\ e1 + eval\ s\ e2$.Se demuestra que $eval\ s\ (e1 + e2) = eval\ s\ (cfold\ e1 + cfold\ e2)$y entonces $eval\ s\ e = eval\ s\ (cfold\ e)$.

- Negación($-$): Supongamos que se cumple la H.I. para e, entonces se tiene que eval s e = eval s (cfold e).Consideremos $e = -e$, para este caso $eval\ s\ (-e)$ realiza la negación del valor e que seria $-eval\ s\ e$.Por lo que se tiene $eval\ s\ (-e) = -eval\ s\ e$. Hacemos lo mismo para $eval\ s\ (-cfold\ e)$ el cual realiza la negación del valor $cfold\ e$ que sería $-eval\ s\ (cfold\ e)$. Por H.I. tenemos que $eval\ s\ (cfold\ e) = eval\ s\ e$.Por lo tanto $eval\ s\ (-cfold\ e) = -eval\ s\ e$. Se demuestra que $eval\ s\ (-e) = eval\ s\ (-cfold\ e)$.

Por lo tanto, para cualquier expresion e se cumple que $eval\ s\ e = eval\ s\ (cfold\ e)$ en el lenguaje EAs.

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

$$e ::= x|n|e1e2|fun(x) \to e$$

En donde el primer constructor representa las variables del lenguaje, el segundo numeros naturales, el tercero aplicacion de funcion y el ultimo la definicion de funciones.

a) Traduce la gramatica anterior a una definicion inductiva con reglas de inferencia.

$$\frac{}{x\ Var} \qquad \frac{n \in Nat}{n\ Exp} \qquad \frac{e_1\ Exp \qquad e_2\ Exp}{(e_1\ e_2)\ Exp} \qquad \frac{x\ Var \qquad e\ Exp}{(fun(x) \to e)\ Exp}$$

b) Disena una sintaxis abstracta apropiada para este lenguaje. Hint: Primero observa si es necesario alguna especie de ligado como el que define el operador let visto en clase.

$$\frac{}{var[x]} \qquad \frac{n \in Nat}{num[n]\ asa} \qquad \frac{e_1\ asa \quad e_2\ asa}{app(e_1, e_2)\ asa} \qquad \frac{x\ Var \quad e\ asa}{fun(x, x.e)\ asa}$$

c) Escribe las reglas para la relacion de analisis sintactico ($\leftarrow\!\!\rightarrow$) del lenguaje.

$$\frac{}{x\ Var\ E \leftrightarrow\ var[x]\ asa} \qquad \frac{n \in Nat}{n\ E \leftrightarrow\ num[n]\ asa} \qquad \frac{e_1\ E \leftrightarrow\ a_1\ asa \quad e_2\ E \leftrightarrow\ a_2\ asa}{e_1\ e_2\ E \leftrightarrow\ app(e_1, e_2)\ asa}$$

$$\frac{x\ Var\ E \leftrightarrow\ var[x] \qquad e\ E \leftrightarrow\ a\ asa}{fun(x) \to e \leftrightarrow fun(x, x.e)\ asa}$$

d) Disena un algoritmo de sustitucion para este lenguaje.

$x[x := e] = e$
$z[x := e] = z$
$num[n][x := e] = n$
$(e_1\ e_2)[x := e] = (e_1[x := e])(e_2[x := e])$
$fun(x, x.t)[x := e] = fun(x[x := e], x.e[x := e])$
$fun(x, a)[x := e] = fun(x[x := e], a[x := e])$
$(z.a)[x := e] = z.(a[x := e])$, Si $x \neq z$ y $z \notin FV(e)$
$(z.a)[x := e] = indefinido$, Si $z \in FV(e)$

e) La relacion de α-equivalencia en este lenguaje se da respecto al operador de definicion de funcion fun, se dice que dos expresiones son α-equivalentes si solo difieren en el nombre de la variable del parametro de la funcion. Por ejemplo:

$$fun(x) \to x \equiv \alpha fun(y) \to y$$

Demuestra que $\equiv \alpha$ es una relacion de equivalencia.

Para poder ver que $\equiv \alpha$ una relacion de equivalencia, debemos demostrar que es:

Reflexiva:
Notemos que para cualquier expresión e, $e \equiv \alpha\ e$, es decir que una expresión es α-equivalente a sí misma.

Simetría:
Si una expresión $e_1 \equiv \alpha\ e_2$, entonces $e_2 \equiv \alpha\ e_1$, es decir que si dos expresiones son α-equivalentes en un sentido, también lo son en el otro.

Transitiva:
Si una expresión $e_1 \equiv \alpha\ e_2$ y tenemos otra expresión $e_2 \equiv \alpha\ e_3$ siendo e_3 una expresión, entonces tenemos que $e_1 \equiv \alpha\ e_3$. Esto significa que si dos expresiones son α-equivalentes, y una de ellas es α-equivalente a una tercera expresión, entonces las dos primeras también son α-equivalentes.

Por lo tanto $\equiv \alpha$ es una relación de equivalencia.

5. Juanito Alimana quiere definir un lenguaje que le permita controlar un robot con movimientos y funcionalidades muy simples. El robot se mueve sobre una cuadricula 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 vacıa de los comandos move y turn separados por punto y coma, cada comando tiene el siguiente funcionamiento:

   - turn hace que el robot de un giro de 90 grados en el sentido de las manecillas del reloj.
   - move provoca que el robot avance una casilla en la direccion hacıa la que esta viendo.

   Un ejemplo de un programa valido 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 posicion vertical mientras que la segunda es la posicion horizontal.
Juanito nos pidio definir la semantica operacional de paso pequeno para el lenguaje que controla al robot. Para esto responde las siguientes preguntas formalmente, para disenar un sistema de transicion:

a) Determina el conjunto de estados.
   Nuestro conjunto de estados, estara compueto, por la posición del robot en el que se encuentre en la cuadricula, es decir nuestras coordenadas (x,y) y la direccion del robot, las cuales son: Norte, Sur, Este, y Oeste.

   Entonces nuestros estados son:

   coordenadas = (x,y) $\in Z$.
   dirección = Norte, Sur, Este, Oeste.

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

   Estado incial: El estado incial nos lo dan en la descripción de problema, el cual es (0,0,Norte)

   Estado final:Al seguir la secuencia de moviviemtos y giros que realiza el robot move;turn;move;turn;turn;turn;move, llegamos a que nos encontramos en la posición (2,1,Norte)

c) Define la funcion de transicion $\rightarrow$R que indique como se debe transitar entre los estados del sistema. De tal forma que defina una semantica operacional de paso pequeno.

   Las transiciones que hara el robot, para que pueda realizar la acciones de "turnz "move"son las siguientes:

   Cuando tenemos el comando "move"hacemos lo siguiente :

   Si estamos en el Norte, se hace:
   (x, y, Norte) $\rightarrow$R(x, y + 1, Norte)
   Si estamos en el Sur, se hace:
   (x, y, Sur) $\rightarrow$R(x, y 1, Sur)
   Si estamos en el Este, se hace:
   (x, y, Este) $\rightarrow$R(x + 1, y, Este)
   Si estamos en el Oeste, se hace:
   (x, y, Oeste) $\rightarrow$R(x 1, y, Oeste)

   Cuando se lee el comando "turn"hacemos lo siguiente:

   Si estamos en el Norte gira de 90 grados en el sentido de las manecillas del reloj:
   (norte) $\rightarrow$ R(este)
   Si estamos en el Este gira de 90 grados en el sentido de las manecillas del reloj:

(este) → R(sur)
Si estamos en el Sur gira de 90 grados en el sentido de las manecillas del reloj:
(sur) → R(oeste)
Si estamos en el Oeste gira de 90 grados en el sentido de las manecillas del reloj:
(oeste) →R(norte)

d) Muestra paso a paso la ejecucion del programa

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

usando la relacion →R y partiendo del estado inicial correspondiente.

Dadas las transiciones dle inciso c) tenemos lo siguiente:

1.-Nos encontramos en l posición (0, 0, Norte) al ejecutar "move"tenemos:
(0, 0, Norte)→R (0,1, Norte).

2.-Nos encontramos en l posición (0, 1, Norte) al ejecutar "turn"tenemos:
(0, 1, Norte)→R (0,1, Este).

3.-Nos encontramos en l posición (0, 1, Este) al ejecutar "move"tenemos:
(0, 1, Este)→R (1,1, Este).

4.-Nos encontramos en l posición (1,1, Este) al ejecutar "turn"tenemos:
(1,1, Este)→R (1,1, Sur).

5.-Nos encontramos en l posición (1, 1, Sur) al ejecutar "turn"tenemos:
(1, 1, Sur)→R (1,1, Oeste).

6.-Nos encontramos en l posición (1, 1, Oeste) al ejecutar "turn"tenemos:
(1, 1, Oeste)→R (1,1, Norte).

7.-Por último nos encontramos en la posición (1,1, Norte) ejecutamos "movez tenemos:
(1,1,Norte)→R (2,1, Norte).

Por lo tanto el estado final del robot es (2,1, Norte).