Algoritmos y Estructuras de Datos I

Primer cuatrimestre de 2013

Departamento de Computación - FCEyN - UBA

Programación funcional - clase 4

Correctitud de programas funcionales

Correctitud de un programa funcional

- Una especificación de un problema en lenguaje de especificación define las propiedades que debe tener una función (contrato).
- Un programa en lenguaje de programación funcional es una definición recursiva de una función.
- Dada una especificación y un programa que responde a esta especificación, queremos **demostrar** que el programa satisface la especificación.

Correctitud

Un programa cumple con una especificación cuando para valores de entrada que satisfacen la precondición:

1. El programa devuelve un valor definido (terminación)
2. El valor devuelto satisface la poscondición (correctitud)

Dada una función `f` y una precondición P_f, diremos que `f` termina si para cada valor de entrada $\overline{\mathtt{x}}$ definido que haga verdadero a P_f, la evaluación de `f` $\overline{\mathtt{x}}$ termina.

Causas de no terminación: indefinición

La evaluación de una función puede no terminar (bien) porque ...

1. ... se genera una cadena infinita de reducciones:

   ```
   func1 :: Int -> Int -> Int
   func1 0 acum = acum
   func1 i acum = func1 (i-1) (2*acum)
   ```

   Si evaluamos `func1 6 5` nos devuelve 320, pero la evaluación de `func1 (-6) 5` no termina.
2. ... ninguna línea de la definición de la función indica qué hacer:

   ```
   func2 :: [a] -> (a,a)
   func2 [x1,x2] = (x1,x2)
   func2 (x:xs) = (x, snd (func2 xs))
   ```

   Por ejemplo, `func2 ''algo1'' = ('a','1')`, pero la evaluación de `func2 ''a''` no termina.
3. ... invocamos a una función que no termina (bien)

Ejemplo

- Especificación:

  problema $fact(n : \text{Int}) = result : \text{Int}\{$
  requiere $n \geq 0$;
  asegura $result == \prod[1..n]$;
  $\}$

- Programa:

```
fact:: Int -> Int
fact n | n==0 = 1
fact n | n > 0 = n * fact (n-1)
```


Demostración de correctitud `fact n` $== \prod[1..n]$

Si la definición es recursiva, lo más natural es demostrar por inducción

Caso base ($n == 0$)

- Debemos ver que `fact 0` $== \prod[1..0]$
- Por la primera ecuación, `fact n | n==0 = 1` concluimos `fact 0` $== \prod[1..0]$

Hipótesis inductiva (HI): `fact n` $== \prod[1..n]$

- Queremos ver que `fact (n+1)` $== \prod[1..n+1]$
- Reducimos la expresión `fact (n + 1)`
- Dado que $n+1 > 0$, usamos la segunda ecuación instanciada en $n+1$ `fact (n + 1) = (n + 1) * fact (n)`
- Usando HI, `fact (n + 1)` $== (n+1) * \prod[1..n]$
- ... y, por propiedad de la productoria, `fact n+1` $== \prod[1..n+1]$

Principio de inducción para $\mathbb{N}$

Sea $P(x)$ una propiedad de $x \in \mathbb{N}$. Queremos probarla para todo $x \in \mathbb{N}$.

En el ejemplo anterior de `fact x`, $P(x)$ dice

"la función `fact` con entrada x da $\prod[1..x]$"

Axioma de inducción: Si

- Vale $P(0)$ y
- para todo n, vale que $P(n)$ implica $P(n+1)$

entonces $P(x)$ es verdadero para todo $x \in \mathbb{N}$

Axioma de inducción completa: Si

- Vale $P(0)$ y
- para todo n, vale que $(\forall m \leq n\ P(m))$ implica $P(n+1)$

entonces $P(x)$ es verdadero para todo $x \in \mathbb{N}$

Órdenes bien fundados

- Podemos aplicar el principio de inducción sobre cualquier tipo de datos con un orden bien fundado.
- Lo aplicaremos sobre tipos algebraicos recursivos, por ejemplo las listas.
- Sea A un conjunto y $\prec$ un orden parcial sobre los elementos de A.
  - antisimétrico (si $a \prec b$ y $b \prec a$ entonces $a = b$ para todo $a, b \in A$)
  - reflexivo ($a \prec a$ para todo $a \in A$)
  - transitivo (si $a \prec b$ y $b \prec c$ entonces $a \prec c$, para todo $a, b, c \in A$)
- $\prec$ es un orden total sobre A si además todos los elementos son comparables, es decir, para todo $a, b \in A$, $a \prec b$ o $b \prec a$

Órdenes bien fundados

- $\prec$ es un orden bien fundado sobre A si $\prec$ es un orden total y cada subconjunto X no vacío tiene un mínimo según $\prec$.
- Ejemplos:
  - $(\leq, \mathbb{N})$, $(\leq, \mathbb{Z})$ y $(\leq, \mathbb{R})$ son órdenes totales
  - $\leq$ sobre $\mathbb{N}$ es un orden bien fundado
  - $\leq$ sobre $\mathbb{Z}$ no es un orden bien fundado porque, por ejemplo, el conjunto de los números negativos no tiene un mínimo
  - $\leq$ sobre $\mathbb{R}$ no es un orden bien fundado, porque por ejemplo el conjunto $\{2^{-n} : n \in \mathbb{N}\}$ no tiene un mínimo.

Inducción en longitud de una lista

- problema $suma(\ell : [\mathsf{Int}]) = result : \mathsf{Int}\{$
  asegura $result == \sum \ell$;
  $\}$
- ```
  suma [] = 0
  suma (x:xs) = x + suma xs
  ```
- Queremos probar $P(\ell)$: " `suma` $\ell == \sum \ell$"
- Hacemos inducción en $|\ell|$ (que toma valores en $\mathbb{N}$)
- Caso base:
  - Tenemos que probar $P(\ell)$ para ℓ tal que $|\ell| == 0$.
  - Es decir, tenemos que probar $P(\ell)$ para $\ell == [\,]$.
  - Por la primera ecuación, `suma` $[\,] = 0$.
  - Como $0 == \sum[]$, concluimos `suma []` $== \sum[]$.

Inducción en longitud de una lista

- Hipótesis inductiva:
  - Asumimos que vale $P(\ell)$: `suma` $\ell == \sum \ell$para toda ℓ tal que $|\ell| \leq n$, $n \in \mathbb{N}$
  - Queremos ver que vale $P(\ell)$ para toda ℓ tal que $|\ell| == n + 1$.
  - Sea ℓ tal que $|\ell| == n + 1$.
  - Luego, $\ell == (x : xs)$ para algun entero x y lista xs.
  - Por la segunda ecuación, `suma (x:xs) = x + suma xs`
  - Dado que $|xs| \leq n$, aplicamos la HI: `suma xs` $= \sum xs$
  - Por propiedad de $\sum$, $x + \sum xs == \sum(x : xs)$
  - Concluimos que `suma (x:xs)` $== \sum(x : xs)$, por lo que vale $P(\ell)$.

Principio de inducción para tipos algebraicos

Inducción estructural: Sea $P(x)$ una propiedad de un elemento x de un tipo algebraico T. Queremos probar que vale $P(x)$ para todo x de tipo T. Sea $c : T \to \mathbb{N}$.

Axioma de inducción: Si

- $P(y)$ es verdadero para toda y tal que $c(y) = 0$
- para todo y, si $\forall y\ (c(y) = n \to P(y))$ entonces $\forall y\ (c(y) = n + 1 \to P(y))$

entonces $P(y)$ es verdadero para todo y de tipo T.

Axioma de inducción completa: Si

- $P(y)$ es verdadero para toda y tal que $c(y) = 0$
- para todo y, si $\forall y\ (c(y) \leq n \to P(y))$ entonces $\forall y\ (c(y) = n + 1 \to P(y))$

entonces $P(y)$ es verdadero para todo y de tipo T

En el ejemplo anterior T es el tipo [Int]

- $P(y)$ es " `suma` $y == \sum y$"
- $c(y) = |y|$.
  ```

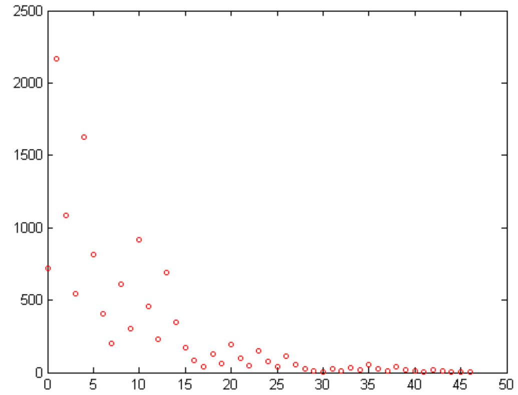
Demostración de terminación

- ```
  func1 :: Int -> Int -> Int
  func1 0 acum = acum
  func1 i acum = func1 (i-1) (2*acum)
  ```
  Es evidente que `func1` termina para $i \geq 0$.

- ```
  func3 :: Int -> Bool
  func3 n | n == 1 = True
          | n ‘mod‘ 2 == 0 = func3 (n ’div’ 2)
          | otherwise = func3 (3*n + 1)
  ```
  ¿Es evidente que `func3` termina para $n > 0$?

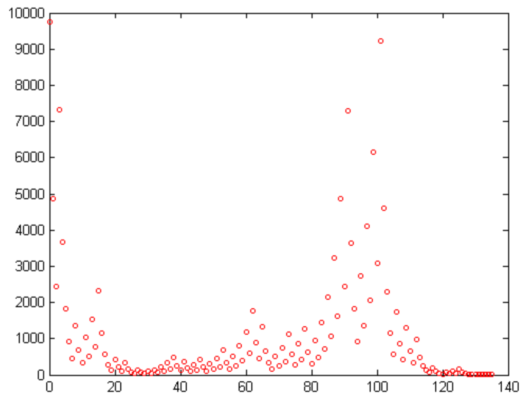
Sucesión de Collatz

- Para $n = 7$, genera la sucesión 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1 ... y entonces `func3 7 = True`
- Para $n = 721$, genera la siguiente sucesión:



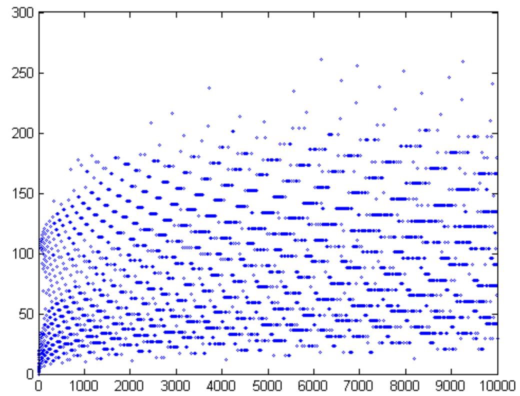
Sucesión de Collatz

- Para $n = 9748$, genera la siguiente sucesión:



Sucesión de Collatz

- Para $n = 1, \ldots, 10^5$ la sucesión converge finalmente a 1, generando la siguiente cantidad de llamadas recursivas:



Función variante

Definición. Dada `f:: T1->... -> Tn`, llamamos función variante de `f` a una función $\mathsf{Fv}_f : T1 \times ... \times T_{n-1} \rightarrow \mathsf{Int}$ estrictamente decreciente en los argumentos de las invocaciones recursivas de `f`.

Es decir, para cada ecuación `f` $\bar{x}$ `= ... f` $\bar{y_1}$ `... f` $\bar{y_m}$ se cumple que

$$Fv_f(\bar{x}) \; > \; Fv_f(\bar{y_i}), \text{ para cada } i : 1..m.$$

Función variante

Teorema (Condición suficiente de terminación).
Sea `f:: T1->... -> Tn`, y sea $\mathsf{Fv}_f : T1 \times ... \times T_{n-1} \rightarrow \mathsf{Int}$ una función variante de `f`, que además cumple con la siguiente condición:

- si $Fv_f(\bar{x}) \leq 0$, entonces `f` $\bar{x}$ se define mediante una ecuación de caso base.

Entonces `f` termina, para cada $\bar{x}$ en $T1 \times ... \times T_{n-1}$.

Corolario. Para cada $\bar{x}$ en $T1 \times ... \times T_{n-1}$, `f` $\bar{x}$ termina en a lo sumo $\mathsf{Fv}_f(\bar{x})$ llamadas recursivas.

Ejemplos

```
factorial :: Int -> Int
factorial n | n == 0 = 1
            | otherwise = n * factorial (n-1)
```

$Fv_{factorial}(n) = n$

```
suma :: [Int] -> Int
suma [] = 0
suma (x:xs) = x + suma xs
```

$Fv_{suma}(\ell) = |\ell|$

```
ultimo :: [a] -> a
ultimo [x] = x
ultimo (x:xs) = ultimo xs
```

$Fv_{suma}(\ell) = |\ell| - 1$

```
mezOrd :: [Int] -> [Int] -> [Int]
mezOrd [] l2 = l2
mezOrd l1 [] = l1
mezOrd (x:xs) (y:ys) | x<=y = x : (mezOrd xs (y:ys))
mezOrd (x:xs) (y:ys) | otherwise = y : (mezOrd (x:xs) ys)
```

$Fv_{mezOrd}(\ell 1, \ell 2) == |\ell 1| + |\ell 2|$