I. Paradigmas de Lenguajes de Programación - 1C2024

Todo el contenido en este archivo es una recopilación de las teóricas/prácticas del campus de la materia. Puede contener errores.

Resumen para el primer parcial

Haskell

Tipos de datos algebraicos

En general, los tipos de datos algebraicos tienen la siguiente estructura:

Los constructores base NO reciben parámetros de tipo T.

Los constructores recursivos reciben al menos UN parámetro de tipo T.

Definición inductiva de T: Los valores de tipo T solamente se pueden construir aplicando una cantidad finita de constructores base y/o recursivos del mismo tipo T.

```
    data T =  casoBase1 <parametros>
                   | casoBase2 <parametros>
                       ...
                   | casoBaseN <parametros>

                   | casoRecursivo1 <parametros>
                   | casoRecursivo2 <parametros>
                       ...
                   | casoRecursivoN <parametros>
```

Recursión estructural

Dada una función ***g : T → Y*** definida por ecuaciones:

```
g(Cbase1 <parametros>)       = <caso base1>
g(Cbase2 <parametros>)       = <caso base2>
...
g(CbaseN <parametros>)       = <caso baseN>

g(Crecursivo1 <parametros>)  = <caso recursivo1>
```

```
g(Crecursivo2 <parametros>)  = <caso recursivo2>
...
g(CrecursivoN <parametros>)  = <caso recursivoN>
```

g está dada por recursión estructural si cumple:

1. Cada caso base devuelve un valor fijo
2. Cada caso recursivo se escribe combinando:
    a. Los parámetros del constructor que ***NO*** son de tipo T (sin usar los parámetros del constructor que son del tipo T).
    b. El llamado recursivo sobre ***cada*** parámetro de tipo T (sin hacer otros llamados recursivos).

Ejemplo de recursión estructural:

```
data AB a = Nil | Bin (AB a) a (AB a)

-- Funcion que abstrae el esquema de recursion estructural en arboles binario
foldAB :: b -> (b -> a -> b -> b) -> (AB a) -> b
foldAB cNil cBin Nil = cNil
foldAB cNil cBin (Bin izq r der) =
    cBin (foldAB cNil cBin izq) r (foldAB cNil cBin der)

-- Ejemplo de uso: sumar todos los elementos del arbol
sumAB :: (Num a) => AB a -> a
sumAB = foldAB 0 (\recI r recD -> recI + r + recD)

mapAB :: (a->b) -> AB a -> AB b
mapAB fMap = foldAB Nil (\recI r recD -> Bin recI (fMap r) recD)
```

Recursión primitiva

Dada una función ***g : T → Y***

g está dada por recursión primitiva si cumple:

1. Cada caso base devuelve un valor fijo
2. Cada caso recursivo se escribe combinando:
    a. **Todos** los parámetros del constructor del tipo T (incluyendo a los que son de tipo T)
    b. El llamado recursivo sobre todos los constructores recursivos (sin hacer otros llamados recursivos)

Recursión iterativa

— TO-DO (en el parcial entra recursión estructural sobre tipos algebraicos)

Inducción estructural

Queremos **demostrar que ciertas expresiones son equivalentes**

Para eso debemos asumir lo siguiente:

1. Trabajamos con estructuras de datos finitas**:** **tipos de datos inductivos**
2. Trabajamos con **funciones totales**:
    a. Las ecuaciones deben cubrir **todos** los casos.
    b. La recursión **siempre** debe terminar.
3. El programa **no** depende del orden de las ecuaciones.

Principio de reemplazo

Sea ***e1 = e2*** una ecuación dentro del programa. Las siguientes operaciones preservan la **igualdad de expresiones:**

1. Reemplazar cualquier instancia de e1 por e2
2. Reemplazar cualquier instancia de e2 por e1

Si una igualdad se puede demostrar solo por principio de reemplazo, decimos que la igualdad **vale por definición.**

A veces conviene dar nombre a todas las ecuaciones del programa para saber cuál ecuación se utilizó para reemplazar una instancia por otra.

Ejemplo:

```
sucesor :: Int -> Int
-- tenemos una igualdad de la forma e1 = e2
sucesor n = n+1      -- {SUC}

-- sucesor (factorial 10) + 1
-- ((factorial 10) + 1) + 1        (por SUC. reemplazo e1 por e2)
-- sucesor (factorial 10 + 1)      (por SUC. reemplazo e2 por e1)
```

Inducción sobre booleanos

Si $P(True)$ y $P(False)$ entonces: $\forall x :: Bool.\ \ P(x)$

Ejemplo:

$$\begin{gathered}\text{Para probar } \forall x :: Bool.\, not(not\ x) = x \\ \text{Basta con probar:} \\ \text{1. P(True): not(not True)} = \text{True} \\ \text{2.P(False): not(not False)} = \text{False}\end{gathered}$$

Ahora podemos usar principio de reemplazo usando las siguientes ecuaciones para probar 1 y 2:

```
not True  = False   -- {NT}
not False = True    -- {NF}
```

Inducción sobre pares

Si $\forall x :: a.\ \forall y :: b.\ P((x, y))$ entonces $\forall p :: (a, b).\ P(p)$

Ejemplo:

$$\text{Para probar } \forall p :: (a,b).\ \text{fst p} = \text{snd (swap p)}$$
$$\text{Basta con probar:}$$
$$\forall x :: a. \forall y :: b.\ \text{fst (x,y)} = \text{snd (swap (x,y))}$$

Y para probar esto podemos usar ppo. de reemplazo usando, por ejemplo, las siguientes ecuaciones dadas:

```
fst (x,_)   =  x      -- {FST}
snd (_,y)   =  y      -- {SND}
swap (x, y) = (y, x)  -- {SWAP}
```

Inducción sobre naturales

```
-- Usamos la siguiente representacion de numeros naturales
data Nat = Zero | Suc Nat
```

Si $P(Zero)$ y $\forall n :: \text{Nat}.\ (P(n) \implies P(\text{Suc}\ n))$ entonces: $\forall n :: \text{Nat}.\ P(n)$

Inducción estructural

Para probar una propiedad P sobre **todas** las instancias de tipo T, basta probar P para cada uno de los constructores (asumiendo como H.I que se cumple para los constructores recursivos)

Tenemos un tipo de datos inductivo de la forma:

```
data T =  casoBase1 <parametros>
                    | casoBase2 <parametros>
                       ...
                    | casoBaseN <parametros>

                    | casoRecursivo1 <parametros>
                    | casoRecursivo2 <parametros>
                       ...
                    | casoRecursivoM <parametros>
```

Sea P una propiedad acerca de las expresiones de tipo T tal que:
-
P vale sobre todos los constructores base de T.
-
P vale sobre todos los constructores recursivos de T, asumiendo como H.I. que vale para los parámetros de tipo T.
Entonces:

$\forall x :: T.\ P(x)$

Ejemplo: Inducción sobre árboles binarios

```
data AB a = Nil | Bin (AB a) a (AB a)
```

$$\begin{gathered}\text{Sea } P \text{ una propiedad sobre expresiones de tipo AB tal que:}\\ -\ P(\text{Nil})\\ -\ \forall i :: (AB\ \text{a}).\ \forall r :: \text{a}.\ \forall d :: (AB\ \text{a}).\\ (P(i) \land P(d)) \implies (P(\text{Bin } i\ r\ d))\\ \\ \text{Entonces } \forall x :: AB\ a.\ P(x)\end{gathered}$$

Ejemplo: Inducción sobre listas

```
data [a] = [] | a : [a]
```

$$\begin{gathered}\text{Sea } P \text{ una propiedad sobre expresiones de tipo [a] tal que:}\\ -\ P([\,])\\ -\ \forall x :: \text{a}.\ \forall xs :: [\text{a}].\ (P(xs) \implies P(x : xs))\\ \\ \text{Entonces } \forall xs : [\text{a}].\ P(xs)\end{gathered}$$

Extensionalidad

Extensionalidad para pares

(se prueba mediante ppo. de inducción estructural)

Si $\text{p} :: (\text{a,b})$, entonces $\exists \text{x} :: \text{a}.\ \exists \text{y} :: \text{b}.\ \text{p} = (\text{x, y})$

Extensionalidad para sumas

```
data Either a b = Left a | Right b
```

Si $\mathrm{e} :: \mathrm{Either\ a\ b}$, entonces:
- O bien
$\exists \mathrm{x} :: \mathrm{a}.\ \ \mathrm{e} = \mathrm{Left\ x}$
- O bien
$\exists \mathrm{y} :: \mathrm{b}.\ \ \mathrm{e} = \mathrm{Right\ y}$

Principio de extensionalidad funcional

Sea $\mathrm{f}, \mathrm{g} ::\ \ \mathrm{a} \to \mathrm{b}$

Si $\left(\forall \mathrm{x} :: \mathrm{a}.\ \ \mathrm{f\,x} = \mathrm{g\,x}\ \right) \implies \ \mathrm{f} = \mathrm{g}$

Para probar una igualdad de tipo $\mathrm{f} = \mathrm{g}$ basta con probar que vale $\mathrm{f\ x} = \mathrm{g\ x}$ para todo x del dominio de las funciones f y g. (Son iguales punto a punto)

Calculo lambda

Calculo lambda puro (sin tipos) extendido con booleanos

```
-- Sintaxis (términos)
M := x                     -- Variables
        | λx.M                 -- Abstraccion
        | M M                  -- Aplicacion

        | true
        | false
        | if M then M else M

-- Semantica (reglas de reduccion)
Si M -> M' entonces:
    M N  ->  M' N
    N M  ->  N M'
    λx.M ->  λx.M'

    if M then N else O -> if M' then N else O
```

Calculo lambda tipado. λ^{BN}

Se agregan los siguientes tipos:

- Funciones
- Booleanos
- Naturales

Es determinista: estrategia **Call-by-value (e**valuar primero los argumentos antes de pasarlos a la función).

$\lambda(x.M)\,V$ reduce solo cuando V está en **forma normal** (no existe V′ tal que V → V′)

Sintaxis del cálculo lambda tipado λ^{BN}

```
-- Términos
M ::=  x
        | λx:σ. M
        | M M
        | true
        | false
        | if M then M else M
        | zero
        | suc(M)
        | pred(M)
        | isZero(M)

-- Tipos
σ ::= Bool      -- Booleanos
        | Nat       -- Naturales
        | σ -> σ    -- Funciones
```

Reglas de tipado de λ^{BN}

Cada término tiene su propia regla de tipado.

$$\frac{}{\Gamma, x:\sigma \vdash x:\sigma}\,ax_v$$

$$\frac{\Gamma, x:\sigma \vdash M:\tau}{\Gamma \vdash \lambda x:\sigma.M\ :\sigma \to \tau}\to_i \qquad \frac{\Gamma \vdash M:\sigma\to\tau \quad \Gamma\vdash N:\sigma}{\Gamma\vdash MN:\tau}\to_e$$

$$\frac{}{\Gamma \vdash \text{true}: Bool}\,ax_{true} \qquad \frac{}{\Gamma \vdash \text{false}: Bool}\,ax_{false}$$

$$\frac{\Gamma \vdash M: Bool \quad \Gamma \vdash P:\sigma \quad \Gamma\vdash Q:\sigma}{\Gamma \vdash \text{if M then P else Q}:\sigma}\,\text{if}$$

$$\frac{}{\Gamma \vdash \text{Zero}:\text{Nat}}\,\text{zero} \qquad \frac{\Gamma \vdash M:\text{Nat}}{\Gamma \vdash \text{isZero(M)}: Bool}\,\text{isZero}$$

$$\frac{\Gamma \vdash \text{M}:\text{Nat}}{\Gamma \vdash \text{pred(M)}:\text{Nat}}\,\text{pred} \qquad \frac{\Gamma \vdash \text{M}:\text{Nat}}{\Gamma \vdash \text{succ(M)}:\text{Nat}}\,\text{succ}$$

Semántica del cálculo lambda tipado λ^{BN}

Valores: Son los términos M los cuales están en **forma normal** y además **FV(M) = ø**

Semántica operacional (small-step):

> Las reglas de cómputo nos dan el significado de los términos al definir como operan.

Las reglas de congruencia permiten reducir subtérminos. (Términos que están dentro de otros términos)

```
-- Valores
V  ::=  true
            | false
            | λx:σ.M
            | zero
            | succ(V)

-- Semantica operacional:
    -- Reglas de cómputo
{β}       - (λx:σ.M) V              -> M{x := V}
{if_t}    - if true then M else N  -> M
{if_f}    - if false then M else N -> N
{pred_}   - pred(succ(V))          -> V
{isZero0} - isZero(zero)           -> true
{isZeroN} - isZero(succ(V))        -> false

    -- Reglas de congruencia
    -- Si M -> M' entonces:
{u}       - M N                 -> M' N
{v}       - V M                 -> V M'
{ifC}     - if M then N else O  -> if M' then N else O
{succC}   - succ(M)             -> succ(M')
{predC}   - pred(M)             -> pred(M')
{isZero}  - isZero(M)           -> isZero(M')
```

Captura de variables (sustituciones)

$M\{x := N\}$ significa **sustitución sin captura de variables de las ocurrencias libres de x en M por N.**

Se define por inducción en M:

```
x{x := N}        = N
y{x := N}        = y
(λx:σ.M){x := N} = M{x := N}               -- No puedo sustituir x porque no
(λy:σ.M){x := N} = λy:σ.(M{x := N})         -- Si y ∉ FV(N)
(λy:σ.M){x := N} = λz:σ.(M{y := z}{x := N})  -- Si y ∈ FV(N), z ∉ FV(N)
(M O){x := N}    = (M{x := N})(O{x := N})

true{x := N}                  = true
false{x := N}                 = false
zero{x := N}                  = zero
(if M then O else P){x := N}  = (if M{x := N} then O{x := N} else P{x := N})
succ(M){x := N}               = succ(M{x := N})
```

```
pred(M){x := N}               = pred(M{x := N})
isZero(M){x := N}             = isZero(M{x := N})
```

Inferencia de tipos

Un término U sin anotaciones de tipo es **Tipable** si y solo si existen:

- Un contexto de tipado Γ
- Un término con anotaciones de tipos M
- Un tipo τ

Tales que **erase(M) = U** **y** $\Gamma \vdash M : \tau$

Donde erase(M) es el término sin anotaciones de tipos que resulta de borrar las anotaciones de tipos de M.

Problema de inferencia de tipos

- Dado un término U, determinar si este es tipable.
- En caso de que U sea tipable:
  - hallar un contexto Γ, un término M y un tipo τ tales que **erase(M) = U** **y** $\Gamma \vdash M : \tau$

Se resuelve con **algoritmo** W de inferencia de tipos.

Algoritmo de unificación de Martelli-Montanari

Unificación: Problema de resolver sistemas de ecuaciones entre tipos con incógnitas.

Definición:

Dado un problema de **unificación** E**:**

- Mientras $E \neq \varnothing$, se aplica sucesivamente alguna de las seis reglas de unificación
- La regla puede resultar en una **falla**
- De lo contrario, la regla es de la forma $E \rightarrow_S E'$
  La resolución del problema E se reduce a resolver otro problema
  E' aplicando la sustitución S.

Si $E = E_0 \rightarrow_{S_1} E_1 \rightarrow_{S_2} E_2 \rightarrow ... \rightarrow_{Sn} E_n \rightarrow_{Sn+1} \text{falla}$
En tal caso, el problema de unificación
E **no tiene solución**.

Si $E = E_0 \rightarrow_{S_1} E_1 \rightarrow_{S_2} E_2 \rightarrow ... \rightarrow_{Sn} E_n \rightarrow_{Sn+1} E_n = \varnothing$
En tal caso, el problema de unificación
E **tiene solución**.

Reglas:

1. **Delete**

$$\{x \stackrel{?}{=} x\} \cup E \qquad \rightarrow \qquad E$$

2. **Decompose**

$$\{C(\tau_1, ..., \tau_n) \stackrel{?}{=} C(\sigma_1, ..., \sigma_n)\} \ \cup \ E \quad \rightarrow \quad \{\tau_1 \stackrel{?}{=} \sigma_1, ..., \tau_n \stackrel{?}{=} \sigma_n\} \ \cup \ E$$

3. **Swap**

$$\{\tau \stackrel{?}{=} ?n\} \ \cup E \qquad \rightarrow \quad \{?n \stackrel{?}{=} \tau\} \cup E$$

(Si τ no es una incógnita)

4. **Elim**

$$\{?n \stackrel{?}{=} \tau\} \ \cup E \qquad \rightarrow_{\{?n \,:=\, \tau\}} \quad E' = \{?n := \tau\}(E)$$

(Si ?n no ocurre en τ)

5. **Clash**

$$\{C(\tau_1, ..., \tau_n) \stackrel{?}{=} C(\sigma_1, ..., \sigma_n)\} \ \cup \ E \quad \rightarrow \quad \text{falla} \quad (\text{Si } C \neq C')$$

6. **Occurs-Check**

$$\{?n \stackrel{?}{=} \tau\} \ \cup E \qquad \rightarrow \quad \text{falla} \quad (\text{Si } ?n \neq \tau \text{ y } ?n \text{ ocurre en } \tau)$$

Corrección del algoritmo de Martelli-Montanari

1. El algoritmo termina para cualquier E
2. Si E no tiene solución, el algoritmo falla
3. Si E tiene solución, el algoritmo llega a:

$$\boxed{E = E_0 \rightarrow_{S_1} E_1 \rightarrow_{S_2} E_2 \rightarrow ... \rightarrow_{Sn} E_n \rightarrow_{Sn+1} E_n = \varnothing}$$

Y además:

$$\boxed{S = S_n \ \circ \ ... \ \circ S_2 \ \circ S_1}$$

Es el unificador más general posible (
mgu)

Algoritmo W de inferencia de tipos

Recibe un término U sin anotaciones de tipos.
Procede recursivamente sobre la estructura de
U:

- Si falla, entonces Uno es tipable.
- Si tiene éxito:
  Devuelve una tripla
  (Γ, M, τ) tal que:
  erase(
  M) = U y $\Gamma \vdash M : \tau$ es válido.

Escribimos
$W(U) \rightsquigarrow \Gamma \vdash M : \tau$ para indicar que el algoritmo de inferencia tiene éxito cuando se le pasa U como entrada y devuelve una tripla (Γ, M, τ).

Reglas:

$$\frac{}{W(\text{True}) \rightsquigarrow \varnothing \vdash \text{True} : \text{Bool}}$$

$$\frac{}{W(\text{False}) \rightsquigarrow \varnothing \vdash \text{False} : \text{Bool}}$$

$$\frac{?k \text{ es una incógnita fresca}}{W(x) \rightsquigarrow x : ?k \vdash x : ?k}$$

$$\frac{\begin{array}{l} W(U_1) \rightsquigarrow \Gamma_1 \vdash M_1 : \tau_1 \\ W(U_2) \rightsquigarrow \Gamma_2 \vdash M_2 : \tau_2 \\ W(U_3) \rightsquigarrow \Gamma_3 \vdash M_3 : \tau_3 \end{array} \quad S = \text{mgu}\left(\begin{array}{c} \{\tau_1 \stackrel{?}{=} \text{Bool}, \tau_2 \stackrel{?}{=} \tau_3\} \\ \{\Gamma_{\text{i}}(x) \stackrel{?}{=} \Gamma_{\text{j}}(x) \mid i,j \in \{1,2,3\}, x \in \Gamma_{\text{i}} \cap \Gamma_{\text{j}}\} \end{array}\right)}{\begin{array}{c} W(\text{if } U_1 \text{ then } U_2 \text{ else } U_3) \rightsquigarrow S(\Gamma_1) \cup S(\Gamma_2) \cup S(\Gamma_3) \vdash \\ S(\text{if } M_1 \text{ then } M_2 \text{ else } M_3) : S(\tau_2) \end{array}}$$

$$\frac{\begin{array}{c} W(U) \rightsquigarrow \Gamma_1 \vdash M : \tau \\ W(V) \rightsquigarrow \Gamma_2 \vdash N : \sigma \\ ?k \text{ es una incógnita fresca} \\ S = \text{mgu}\{\tau \stackrel{?}{=} \sigma \to ?k\} \cup \{\Gamma_1(x) \stackrel{?}{=} \Gamma_2(x) : x \in \Gamma_1 \cap \Gamma_2\} \end{array}}{W(UV) \rightsquigarrow S(\Gamma_1) \ \cup \ S(\Gamma_2) \ \vdash \ S(MN) \ : \ S(?k)}$$

$$\frac{W(U) \rightsquigarrow \Gamma \ \vdash \ M : \tau \quad \sigma = \begin{cases} \Gamma(x) & \text{si x} \in \Gamma \\ \text{incógnita fresca } ?k & \text{si no} \end{cases}}{W(\lambda x. U) \ \rightsquigarrow \ \Gamma / \{x\} \ \vdash \ (\lambda x : \sigma. M) : \sigma \to \tau}$$

Algoritmo W extendido para λ^{BN}

$$\frac{}{W(\text{zero}) \rightsquigarrow \varnothing \vdash \text{zero} : \text{Nat}}$$

$$\frac{W(U) = \Gamma \vdash M : \sigma \qquad S = \text{mgu}(\sigma \stackrel{?}{=} \text{Nat})}{W(\text{succ}(U)) \rightsquigarrow S(\Gamma) \vdash S(\text{succ}(M)) : \text{Nat}}$$

$$\frac{W(U) = \Gamma \vdash M : \sigma \qquad S = \text{mgu}(\sigma \stackrel{?}{=} \text{Nat})}{W(\text{pred}(U)) \rightsquigarrow S(\Gamma) \vdash S(\text{pred}(M)) : \text{Nat}}$$

$$\frac{W(U) = \Gamma \vdash M : \sigma \qquad S = \text{mgu}(\sigma \stackrel{?}{=} \text{Nat})}{W(\text{isZero}(U)) \rightsquigarrow S(\Gamma) \vdash S(\text{isZero}(M)) : \text{Bool}}$$