

La noción de principalidad en Especialización de Tipos

Pablo E. “Fidel” Martínez López

Tesis de Doctorado
en Ciencias de la Computación

Dpto de Computación, Facultad de Cs Exactas y Naturales

Universidad Nacional de Buenos Aires

3ras Jornadas de Ciencias de La Computación

Universidad Nacional de Rosario

El comienzo...

Bilbo tomó un copón de doble asa, de los más pesados que podía cargar, y echó una temerosa mirada hacia arriba. (...)

Entonces escapó corriendo. (...) Su principal pensamiento era “¡Lo hice! (...) ¡Ahora soy realmente un saqueador!”

El Hobbit

John R. R. Tolkien

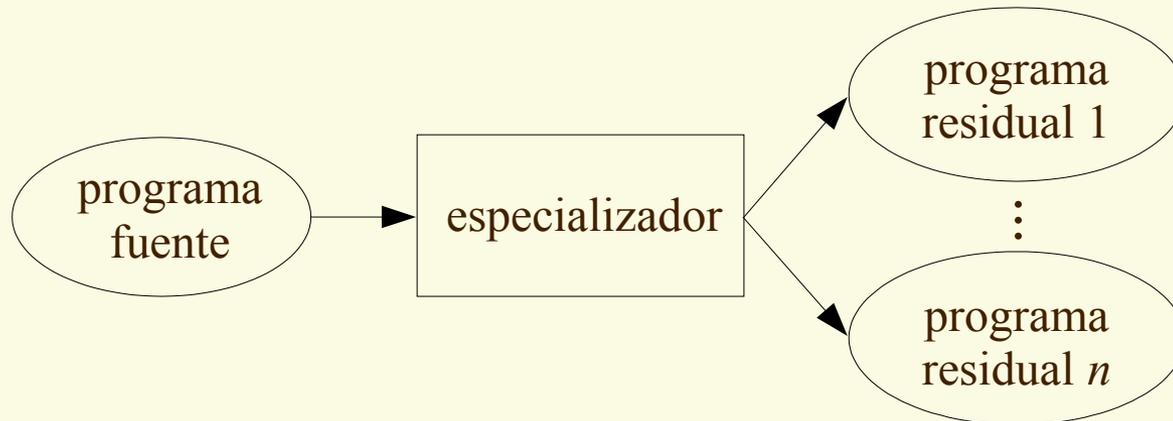
Especialización de Programas

✓ Motivación:

- Generación de programas por medios automáticos

✓ Método:

- Especialización de programas
- Enfoque tradicional: evaluación parcial



Especialización de Programas

power (n,x) =

if n==1

then x

else x * power (n-1,x)

n=2

power₂ x = x * x

n=3

power₃ x = x * (x * x)

n=27

power₂₇ x = x * (x * ... * (x * x)...)

27 veces

mix

Programa
fuente

Especializador

Programas
residuales

Evaluación Parcial (PE)

- ✓ Especialización por EVALUACIÓN (reducción)
- ✓ Programa fuente: varios argumentos
 - Argumentos conocidos: *estáticos* (S)
 - Los restantes: *dinámicos* (D)
- ✓ Operaciones anotadas con S ó D
 - Análisis de binding times (BTA)
- ✓ Técnica:
 - Reducir todas las operaciones estáticas

Evaluación Parcial

✓ Compilando por especialización de intérpretes

– **Ecuación Mix**

$\text{prog_res} = \text{mix prog_fuente datos}^S$

t.q. $\text{prog_res datos}^D = \text{prog_fuente datos}^S \text{ datos}^D$

– **Primera Proyección de Futamura**

$\text{prog}' = \text{mix interp prog}$

t.q. $\text{prog}' \text{ data} = \text{interp prog data}$

– **Especialización Óptima**

$\text{mix auto_interp prog} = \text{prog}' = \text{prog}$

Evaluación Parcial

✓ Compilando por especialización de intérpretes **con tipos**

– **Ecuación Mix**

$\text{prog_res} = \text{mix prog_fuente datos}^S$

t.q. $\text{prog_res datos}^D = \text{prog_fuente datos}^S \text{ datos}^D$

– **Primera Proyección de Futamura**

$\text{prog}' = \text{mix interp prog}$

t.q. $\text{prog}' \text{ data} = \text{interp prog data}$

$\text{interp} :: \text{Prog} \rightarrow \text{Val} \rightarrow \text{Val}$

– **Especialización Óptima, perdida**

$\text{mix auto_interp prog} = \text{prog}' \neq \text{prog}$

$\text{prog}' :: \text{Val} \rightarrow \text{Val}$

El problema de los *tags*

data Val = N^D Int^D | F^D (Val \rightarrow^D Val)

eval (Const n) env = N^D n

eval (Var x) env = env x

eval (Lam x e) env =

F^D ($\lambda^D v$. eval e (bind x v env))

eval (App e e') env =

unF^D (eval e env) @^D (eval e' env)

`` $\lambda f. f(f 0)$ ``
(Lam 'f (App (Var 'f)
(App (Var 'f) (Const 0))))

mix

F (λv . unF v
(unF v (N 0)))
:: Val

El problema de los *tags*

data Val = N^D Int^D | F^D (Val \rightarrow^D Val)

eval (Const n) env = N^D n

eval (Var x) env = env x

eval (Lam x e) env =

F^D ($\lambda^D v$. eval e (bind x v env))

eval (App e e') env =

unF^D (eval e env) @^D (eval e' env)

``0 2``
(App (Const 0) (Const 2))

mix

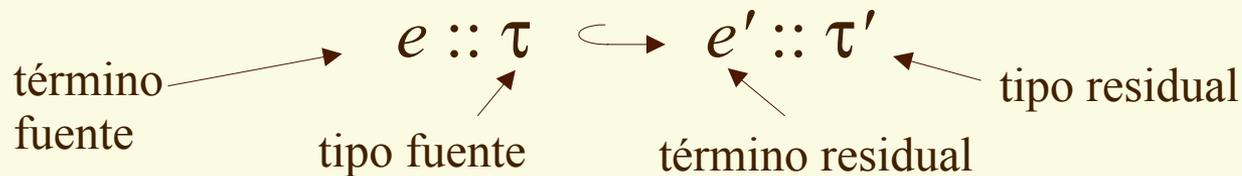
unF (N 0) (N 2)
:: Val

Especialización de Tipos

- ✓ Los tipos aproximan información estática
 - e.g. $e :: \text{Int}$ significa que $[e]$ es un entero
(si su computación termina)
- ✓ ¿Podemos tener una aproximación mejor?
¡Sí!
 - e.g. podemos tener un *tipo* que exprese que e es 42
(llamémoslo $\hat{42}$)
 - La expresión para calcular e no se necesita más
 - Por ello, podemos especializar $e :: \text{Int} \hookrightarrow \bullet :: \hat{42}$

Especialización de Tipos

- ✓ ¡Los tipos se pueden especializar!
 - e.g. Int especializa a $\widehat{42}$ en el ejemplo previo
- ✓ Nueva forma de juicio



- ✓ Más posibilidades:

$$\underbrace{(\lambda^D x. x +^S 1^S) 41^S}_{:: \text{Int}^S \rightarrow \text{Int}^S} :: \text{Int}^S \longleftrightarrow \underbrace{(\lambda x. \bullet)}_{:: \widehat{41} \rightarrow \widehat{42}} \bullet :: \widehat{42}$$

Especialización de Tipos

- ✓ Especialización por INFERENCIA DE TIPOS
 - Mejor flujo de información
- ✓ Programa fuente anotado
 - ¡Las anotaciones son más flexibles que en PE!
 - BTA completamente automático no es posible
 - Las anotaciones son *parte del input*
- ✓ Técnica:
 - Mover la información estática al tipo
 - ¡Precisa un sistema de tipos residual más rico!

Especialización de Tipos

- ✓ Los tipos suma se especializan a una rama

- e.g. $\text{InL}^S 41^D :: \text{Int}^{D+S} \text{Bool}^S \hookrightarrow 41 :: \text{InL Int}$

- ✓ La información de tipos extra se utiliza en el case para elegir la alternativa

$$\begin{aligned} & (\lambda^D x. \text{case}^S x \text{ of } \text{InL } a \rightarrow a +^D 1^D; \\ & \quad \text{InR } b \rightarrow \text{if}^S b \text{ then } 1^D \text{ else } 0^D \\ &) (\text{InL}^S 41^D) :: \text{Int}^D \hookrightarrow \underbrace{(\lambda x. x + 1)}_{\text{InL Int} \rightarrow \text{Int}} 41 :: \text{Int} \end{aligned}$$

¡El problema de los *tags*, resuelto!

data Val = N^S Int^D | F^S (Val \rightarrow^D Val)

eval (Const n) env = N^S n

eval (Var x) env = env x

eval (Lam x e) env =

F^S ($\lambda^D v$. eval e (bind x v env))

eval (App e e') env =

unF^S (eval e env) @^D (eval e' env)

`` $\lambda f. f (f 0)$ ``
(Lam 'f (App (Var 'f)
(App (Var 'f) (Const 0))))

tspec

($\lambda v. v (v 0)$)
:: F (F (N Int \rightarrow N Int)
 \rightarrow N Int))

¡El problema de los *tags*, resuelto!

data Val = $N^S \text{Int}^D \mid F^S (\text{Val} \rightarrow^D \text{Val})$

eval (Const n) env = $N^S n$

eval (Var x) env = env x

eval (Lam x e) env =

$F^S (\lambda^D v. \text{eval } e \text{ (bind } x \ v \ \text{env)})$

eval (App e e') env =

$\text{unF}^S (\text{eval } e \ \text{env}) @^D (\text{eval } e' \ \text{env})$

``0 2``
(App (Const 0) (Const 2))

tspec

Error: (N Int) is different
than (F (N Int \rightarrow N Int))

Especialización de Tipos

✓ Anotación adicional: lift

- **lift** transforma un Int estático a uno dinámico
- comparar

$$(\lambda^D x. x +^S 1^S) 41^S :: \text{Int}^S \hookrightarrow (\lambda x. \bullet) \bullet :: \widehat{42}$$

$$(\lambda^D x. \text{lift } (x +^S 1^S)) 41^S :: \text{Int}^D \hookrightarrow (\lambda x. 42) \bullet :: \text{Int}$$

- entonces

$$\text{let}^D f = (\lambda^D x. \text{lift } (x +^S 1^S)) \text{ in } f 41^S :: \text{Int}^D \hookrightarrow \text{let } f = (\lambda x. 42) \text{ in } f \bullet :: \text{Int}$$

Especialización de Tipos

- ✓ No hay polivariarización por defecto

```
letD f = (λDx. lift (x +S1S))  
in (f 41S, f 17S) :: (IntD, IntD)
```

↪  *error* ($\hat{41}$ no es igual a $\hat{17}$)

- ¡Se precisan tanto $f :: \hat{41} \rightarrow \text{Int}$ como $f :: \hat{17} \rightarrow \text{Int}$!
 - ¡Este tipo de errores es una **característica**!
- ✓ Polivariarización
 - Habilidad de especializar a más de un residuo
 - Guiado por anotaciones
 - Característica de primera clase

Especialización de Tipos

- ✓ Anotaciones adicionales: poly y spec

$$\text{let}^{\text{D}} f = \text{poly } (\lambda^{\text{D}}x. \text{lift } (x +^{\text{S}} 1^{\text{S}})) \\ \text{in } (\text{spec } f \ 41^{\text{S}}, \text{spec } f \ 17^{\text{S}}) :: (\text{Int}^{\text{D}}, \text{Int}^{\text{D}})$$

$$\text{let } f = (\lambda x. 42, \lambda x. 18)^{\text{S}} \\ \text{in } (\pi_1^{\text{S}} f \bullet, \pi_2^{\text{S}} f \bullet) :: (\text{Int}, \text{Int})$$

- Observar como poly se transforma en una tupla, y cada spec en la proyección correspondiente

Monomorfización

data Val = N^S Int^D | F^S (Val \rightarrow^D Val)

data MPVal = M^S Val | P^S (poly Val)

eval (Const n) env = N^S (lift n)

eval (Var x) env =

case^S env x of

M^S v \rightarrow v; P^S v \rightarrow spec v

:

eval (Let x e e') env =

let^D v = P^S (poly (eval e env))

in eval e' (bind x v env)

``let i = $\lambda x.x$ in i i 0''
(Let 'i' (Lam 'x' (Var 'x'))
(App (App (Var 'i') (Var 'i'))
(Const 0)))

tspec

let i = ($\lambda x.x, \lambda x.x$)^S
in (π_1^S i) (π_2^S i) 0
:: N Int

Especialización de Tipos

- ✓ e.g. expresiones lift

$$(\lambda^D x. \text{lift } (x^{+S} 1^S)) :: \text{Int}^S \rightarrow^D \text{Int}^D$$



$$\lambda x. m :: \hat{n} \rightarrow \text{Int}, \text{ para todos } n, m \text{ tal que } m = n+1$$

- ✓ Sólo puede decidirse usando el contexto,

e.g.

$$(\lambda^D x. \text{lift } (x^{+S} 1^S)) 41^S :: \text{Int}^D$$



$$(\lambda x. 42) \bullet :: \hat{41} \rightarrow \text{Int}$$

Especialización de Tipos

- ✓ Es peor para expresiones polivariantes

$$\text{let}^D f = \text{poly } (\lambda^D x. \text{lift } (x +^S 1^S)) \\ \text{in } (\text{spec } f \ 41^S, \text{spec } f) :: (\text{Int}^D, \text{Int}^S \rightarrow^D \text{Int}^D)$$

$$\text{let } f = (\lambda x. 42)^S \\ \text{in } (\pi_1^S f \bullet, \pi_1^S f) :: (\text{Int}, \widehat{42} \rightarrow \text{Int})$$

pero también


$$\text{let } f = (\lambda x. 42, \lambda x. m)^S \\ \text{in } (\pi_1^S f \bullet, \pi_2^S f) :: (\text{Int}, \widehat{n} \rightarrow \text{Int}), \\ \text{para todos } n, m \text{ tal que } m = n+1$$

Especialización Principal de Tipos

✓ Primera idea: usar variables de tipos

- e.g.

$$(\lambda^D x. x +^S 1^S) :: \text{Int}^S \rightarrow^D \text{Int}^S \hookrightarrow \lambda x. \bullet :: \forall \alpha \beta. \alpha \rightarrow \beta$$

- ¡No captura la relación entre α y β !

✓ Mi propuesta:

- Usar un sistema residual con tipos calificados

- Los predicados expresan dependencias sobre información estática desconocida

(e.g. $_ := _ + _$ es un predicado que relaciona 3 tipos).

- entonces $\lambda x. \bullet :: \forall \alpha \beta. \beta := \alpha + \hat{1} \Rightarrow \alpha \rightarrow \beta$

Especialización Principal de Tipos

- ✓ Los predicados pueden verse como restricciones entre tipos
- ✓ Habrá una clase de predicados para cada clase de construcción estática, e.g.
 - $\text{IsInt } \alpha$ expresa que α debe ser un tipo \hat{n}
 - $\alpha := \alpha_1 + \alpha_2$ expresa que α debe ser un tipo \hat{n} , α_1 un tipo \hat{m}_1 , y α_2 un tipo \hat{m}_2 , tal que $n = m_1 + m_2$

Especialización Principal de Tipos

✓ Yo distingo entre

- tipos (τ) : $\text{Int} \rightarrow \text{Int}$
- tipos calificados (ρ) : $\beta := \alpha + \hat{1} \Rightarrow \alpha \rightarrow \beta$
- esquemas (σ) : $\forall \alpha \beta. \beta := \alpha + \hat{1} \Rightarrow \alpha \rightarrow \beta$

✓ La polivarianza se expresa diferente.

- (poly σ) es un tipo que representa al residuo de expresiones polivariantes

✓ Se define una noción de “más general que” entre esquemas de tipos.

Especialización Principal de Tipos

✓ ¿Qué sucede en el lenguaje de términos?

- Vimos que

$$(\lambda^D x. x +^S 1^S) :: \text{Int}^S \rightarrow^D \text{Int}^S$$



$$\lambda x. \bullet :: \forall \alpha \beta. \beta := \alpha + \hat{1} \Rightarrow \alpha \rightarrow \beta$$

- pero, ¿cuál debe ser el resultado con lift?

$$(\lambda^D x. \text{lift}(x +^S 1^S)) :: \text{Int}^S \rightarrow^D \text{Int}^D$$



$$(\lambda x. \text{??}) \bullet :: \forall \alpha \beta. \beta := \alpha + \hat{1} \Rightarrow \alpha \rightarrow \text{Int}$$

- Algo que se corresponda con los predicados...

Especialización Principal de Tipos

✓ *Evidencia*: la contraparte de los predicados

– La evidencia se puede abstraer

$$(\lambda^D x. \text{lift}(x +^S 1^S)) :: \text{Int}^S \rightarrow^D \text{Int}^D$$



$$\Lambda h. \lambda x. h :: \forall \alpha \beta. \beta := \alpha + \hat{1} \Rightarrow \alpha \rightarrow \text{Int}$$

– La evidencia se puede instanciar

$$(\lambda^D x. \text{lift}(x +^S 1^S)) 41^S :: \text{Int}^D$$



$$(\Lambda h. \lambda x. h) \{\{42\}\} \bullet :: \text{Int}$$

– Cada predicado tiene su propia forma de evidencia.

Especialización Principal de Tipos

✓ La polivarianza usa esquemas
(en lugar de tuplas)

– Las expresiones polivariantes abstraen evidencia

$$\text{poly } (\lambda^D x. \text{lift}(x +^S 1^S)) :: \text{poly } (\text{Int}^S \rightarrow^D \text{Int}^D)$$

↪

$$\Lambda h. \lambda x. h \quad :: \text{poly } (\forall \alpha \beta. \beta := \alpha + \hat{1} \Rightarrow \alpha \rightarrow \text{Int})$$

– Los specs proveen la instancia correcta

$$\text{let}^D f = \text{poly } (\lambda^D x. \text{lift}(x +^S 1^S)) \text{ in spec } f \ 41^S \quad :: \text{Int}^D$$

↪

$$\text{let } f = \Lambda h. \lambda x. h \text{ in } f \ \{\{42\}\} \bullet \quad :: \text{Int}$$

Especialización Principal de Tipos

- ✓ Se divide la especialización en fases:
 - 1) Generación de restricciones (especialización propia)
 - 2) Resolución de restricciones
- ✓ Generación de restricciones
 \cong especificación del problema
- ✓ Resolución de restricciones
 \cong implementación de la especificación

Especialización Principal de Tipos

✓ Generación de restricciones

- Dirigida por sintaxis
- Los términos tienen especialización principal
 - Para cada $e : \tau$, especializable existe $e'_p : \sigma'_p$ tal que

$$e :: \tau \quad \hookrightarrow \quad e'_p :: \tau'_p$$

y, para cualquier especialización $e' : \sigma'$ de e , $e'_p : \sigma'_p$ es más general que $e' : \sigma'$

Especialización Principal de Tipos

✓ Resolución de restricciones

- Permite diferentes heurísticas y diferentes lenguajes objetivo
 - Resolución de restricciones regular: abstracción y aplicación de evidencia
 - Eliminación de evidencia: tuplas
 - Otras heurísticas permiten:
 - Generación de polimorfismo
 - Especialización de estructuras lazy

Especialización Principal de Tipos

✓ *Eliminación de evidencia*

- Mueve la evidencia hacia las abstracciones
- ¡Reintroduce las tuplas!

$$\text{let}^D f = \text{poly } (\lambda^D x. \text{lift } (x +^S 1^S)) \\ \text{in } (\text{spec } f \ 41^S, \text{spec } f \ 17^S) :: (\text{Int}^D, \text{Int}^D)$$

↪

$$\text{let } f = \Lambda h. \lambda x. h \text{ in } (f \ \{\{42\}\} \bullet, f \ \{\{18\}\} \bullet) :: (\text{Int}, \text{Int})$$

•→

$$\text{let } f = ((\Lambda h. \lambda x. h) \ \{\{42\}\}, (\Lambda h. \lambda x. h) \ \{\{18\}\})^S \\ \text{in } (\pi_1^S f \bullet, \pi_2^S f \bullet) :: (\text{Int}, \text{Int})$$

Generando polimorfismo

```
``let i = λx.x in i i 0``  
(Let 'i' (Lam 'x' (Var 'x'))  
  (App (App (Var 'i') (Var 'i'))  
        (Const 0)))
```

tspec

$\text{poly}(\forall \alpha. \text{IsResOf } \alpha \text{ Val} \Rightarrow F (\alpha \rightarrow \alpha))$

let i = $\Lambda h. \lambda x. x$
in i {...} (i {...}) 0
:: N Int

Generando polimorfismo

```
data Val = NS IntD | FS (Val →D Val)
data MPVal = MS Val | PS (polym Val)
:
eval (Var x) env =
  caseS env x of
    MS v → v; PS v → inst v
:
eval (Let x e e') env =
  letD v = PS (polym (eval e env))
  in eval e' (bind x v env)
```

```
``let i = λx.x in i i 0``
(Let 'i' (Lam 'x' (Var 'x'))
 (App (App (Var 'i') (Var 'i'))
 (Const 0)))
```

tspec

$\forall \alpha. F (\alpha \rightarrow \alpha)$

```
let i = λx.x
in i i 0
:: N Int
```

Estructuras *Lazy*

```
data ListS t = NilS | ConsS t (ListS t);  
let recS f x = ConsS 1D (f x)  
in caseS f ()S of ConsS x xs → x :: IntD
```



```
 $\Lambda$  hU, hL, hy. firstS (hL @v • @v ()S)  
::  $\forall$   $\alpha$ ,  $\alpha_{ys}$ ,  $\alpha_e$ . IsFix clxs  $\alpha$ ,  
                          IsFun  $\alpha$  clos( $\alpha_e \rightarrow$  Cons Int  $\alpha_{ys}$ ),  
                          IsConstrOf (ListS IntD)  $\alpha_{ys}$   
⇒ Int
```

Recursión Dinámica

letrec^D poly f x ys = if^D lift x ==^D 0^D then x +^S 1^S
else spec f @ x @ snd^D ys
in $\lambda^D zs \rightarrow \text{spec f @ 3}^S @ zs$
 $:: \alpha \rightarrow^D \text{Int}^S$ where $\alpha = (\text{Int}^S, \alpha)$



letrec f x ys = if 3 == 0 then •
else f @ x @ snd ys
in $\lambda zs \rightarrow f @ \bullet @ zs$
 $:: \alpha \rightarrow \hat{4}$ where $\alpha = (\hat{n}, \alpha)$

Recursión Dinámica

Pero también...

```
letrecD poly f x ys = ifD lift x ==D 0D then x +S 1S
                        else spec f @ x @ sndD ys
in λD zs → spec f @ 3S @ zs
:: α →D IntS where α = (IntS, α)
```



```
letrec f x ys = if 3 == 0 then •
                else f' @ x @ snd ys
      f' x ys = if 3 == 0 then •
                else f @ x @ snd ys
in λ zs → f @ • @ zs
:: α →  $\hat{4}$  where α = ( $\hat{n}$ , α)
                β = ( $\hat{m}$ , β)
```

Mis Contribuciones

✓ Dos fases:

- *Cálculo de restricciones* \Rightarrow dirigido por sintaxis
- *Resolución* \Rightarrow permite diferentes heurísticas

✓ Nuevo tratamiento de polivarianza.

✓ Existencia de especializaciones principales.

✓ Posibilidades de mi formulación:

- Generación de programas residuales polimórficos
- Especialización de lenguajes con evaluación *lazy*
- Mejor formulación de los problemas que trae la interacción entre polivarianza y recursión dinámica

Trabajos Futuros (1)

- ✓ Considerar construcciones fuente adicionales
 - Tipos suma dinámicos
 - Especialización de constructores
 - ¡¡Recursión dinámica!!
- ✓ Implementación
 - Implementación más eficiente
 - Especificación y testeo usando QuickCheck
 - Asistente de cálculo de *binding times*

Trabajos Futuros (2)

- ✓ Arity raising
 - Programas polimórficos
 - Términos con predicados
- ✓ ¿Especialización de Tipos para Haskell?
 - Polimorfismo en programas fuente
 - Especialización de evaluación *lazy*
(interacción entre resolución de restricciones y arity raising)
 - Especialización de clases de tipos

Concluyendo...

*El Portero sonrió y dijo su nombre; y Ged,
repiteéndolo, entró por última vez en aquella Casa.*

Un Mago de Terramar

Úrsula K. Le Guin