Tipos y Categorias: algunas conexiones

José Luis Freire Nistal
Departamento de Computación
Facultad de Informática
Universidad de A Coruña.
freire@udc.es

SECA III
Santiago de Compostela, Setiembre 2005

Algunas referencias

- CCC y λ–cálculo. J. Lambek & P.J. Scott: *Higher Order Categorical Logic*. Cambridge Studies in Advanced Mathematics. Vol **7**. 1986
- Monadas (Triples) y semántica de procesos: compilación con continuaciones. Eugenio Moggi. 1989
- Semántica denotacional: tipos de datos y programas como objetos y morfismos de categorías. A. Asperti & G. Longo. *Categories, Types and Structures*. MIT Press, Foundations of Computing Series. 1991

..One of the most powerfull ideas in logic and informatics in recent years has been the analogy between

conjuction $(\wedge, \top)$	and	product types $(\times, \mathbf{1})$
implication $(\Rightarrow)$	and	function–types $(\rightarrow)$
disjunction $(\vee, \bot)$	and	sum types $(+, \emptyset)$

which puts propositions and types on a par.

Formulas correspond to types and their deductions to terms. Crudely, a type gives rise to the proposition that the type has an element, and a proposition to the type whose elements are its proofs.

Paul Taylor. *Practical Foundations of Mathematics*, Cambridge studies in advanced mathematics.

Marco

Correspondencia de Curry–Howard:

William Howard, *The formulae–as–types notion of construction*. En J. P. Seldin and J. R. Hindley, editors, lo identificó en el trabajo de *To Haskell. B. Curry: Essays on Combinatory Logic, Lambda-Calculus, and Formalism*, 479–490. Academic Press, NY, 1980.

Nikolas Bruijn, Joachim Lambek, Hans Läuchli and Bill Lawvere (60's). Desarrollada por Dana Scott para formalizar las intuiciones de Brouwer y de un modo más preciso por Martin–Löf.

$$\frac{\text{demostración}}{\text{fórmula}} \equiv \frac{\lambda\text{–término}}{\text{tipo}} \equiv \frac{\text{programa}}{\text{especificación}}$$

$$\begin{aligned} D_A &\equiv P_A \\ D_{A\to B} &\equiv P_{A\to B} \\ D_B &\equiv P_{A\to B}(P_A) \end{aligned}$$

Semántica de Heyting y Kolmogorof:

A	A	$a : A$
$A \supset B$	$A \to B$	$f : A \to B$
$\exists x \in X.P(x)$	$\sum_{x\in X} P(x)$	$(x, p),\ x \in X,\ p : (P(x))$
$\forall x \in X.P(x)$	$\prod_{x\in X} P(x)$	$\lambda t.p,\ p : (P(t))$
$A \wedge B$	$A \times B$	$(a, b),\ a : A,\ b : B$
etc.		

Coq básico

Coq: sistema interactivo de desarrollo de pruebas que implementa el cálculo de construcciones (CC) (T. Coquand, G. Huet, *The calculus of constructions*, Information and Computation 1988).

- sistema de tipos más fuerte (Barendregt). Implementa tipos polimórficos, de orden superior y dependientes.
- tipos inductivos: permite definir tipos de datos, especificaciones y predicados y pruebas por inducción estructural.
- CIC es un λ–cálculo tipado con un sistema de reglas de deducción natural para derivar expresiones de la forma:

$$E[\Gamma] \vdash t : T$$

- Entornos y contextos

$$\mathcal{WF}(E)[\Gamma]$$

$$\frac{}{\mathcal{WF}([])[[]]}\, W-E$$

$$\frac{E[\Gamma] \vdash T : s \quad s \in S \quad x \notin \Gamma \cup E}{\mathcal{WF}(E)[\Gamma :: (x : T)]}\, W-s$$

Tipado

$S \equiv \{Prop, Set, Type(i) \mid i \in \mathbf{N}\}$ Clases (*sorts*) .

- **básicos**: $Prop$ (impredicativo) y Set
- **universos**: $Type(i)$'s

t está bien tipado en un entorno E si existe un contexto Γ y un término T tal que la expresión $E[\Gamma] \vdash t : T$ puede derivarse usando las reglas siguientes:

$$\frac{\mathcal{WF}(E)[\Gamma]}{E[\Gamma] \vdash Prop : Type(0)} \qquad \frac{\mathcal{WF}(E)[\Gamma]}{E[\Gamma] \vdash Set : Type(0)} \qquad \frac{\mathcal{WF}(E)[\Gamma] \quad i < j}{E[\Gamma] \vdash Type(i) : Type(j)}\, Ax$$

$$\frac{\mathcal{WF}(E)[\Gamma] \quad (x : T) \in \Gamma}{E[\Gamma] \vdash x : T}\, Var$$

$$\frac{\mathcal{WF}(E)[\Gamma] \quad (c : T) \in \Gamma}{E[\Gamma] \vdash c : T}\, Const$$

$$\frac{E[\Gamma] \vdash T : s_1 \quad E[\Gamma :: (x : T)] \vdash U : s_2 \quad s_1 \in \{Prop,\, Set\} \text{ o bien } s_2 \in \{Prop,\, Set\}}{E[\Gamma] \vdash \forall (x : T).U : s_2}\, Prod$$

$$\frac{E[\Gamma] \vdash T : Type(i) \quad E[\Gamma :: (x : T)] \vdash U : Type(j) \quad i \leq k \quad j \leq k}{E[\Gamma] \vdash \forall (x : T).U : Type(k)}\, Prod$$

$$\frac{E[\Gamma] \vdash \forall(x:T).U : s \quad E[\Gamma :: (x:T)] \vdash f : U}{E[\Gamma] \vdash fun(x:T) => f : \forall(x:T).U}\, Lam$$

$$\frac{E[\Gamma] \vdash f : \forall(x:T).U \quad E[\Gamma] \vdash t : T}{E[\Gamma] \vdash (f\ t) : U\{x/t\}}\, App$$

Dados tipos T y U, escribimos

$$T \leq_{\beta\delta\iota} U$$

para expresar la convertibilidad de tipos. Entonces:

$$\frac{E[\Gamma] \vdash U : s \quad E[\Gamma] \vdash t : T \quad T \leq_{\beta\delta\iota} U}{E[\Gamma] \vdash t : U}\, Conv$$

Tipos Inductivos

Reglas de introducción

```
Coq < Inductive nat : Set :=  O : nat | S : nat->nat
Coq < Check nat.
nat: Set
Coq < Check O.
O: nat
Coq < Check S.
S: nat->nat
```

Reglas de eliminación

```
nat_ind: forall(P:(nat->Prop)),(P O)->(forall(n:nat),(P n)->(P (S n)))->forall(n:nat),(P n)
nat_rec: forall(P:(nat->Set)),(P O)->(forall(n:nat),(P n)->(P (S n)))->forall(n:nat),(P n)
nat_rect: ,(P:(nat->Type)),(P O)->(forall(n:nat),(P n)->(P (S n)))->forall(n:nat),(P n)
```

Reglas de cómputo

```
Lemma nat_ind_1:forall(P:(nat->Prop)),
                (init: (P O))
                (paso:forall(n:nat),(P n)->(P (S n)))
                (nat_ind P init paso O)==init.
Auto.
Save.
Lemma nat_ind_2:forall(P:(nat->Prop)),
                (init: (P O))
                (paso:forall(n:nat),(P n)->(P (S n)))
                forall(n:nat),
                   (nat_ind P init paso (S n))
                       ==(paso n (nat_ind P init paso n)).

Auto.
Save.
```

Expresividad

```
Variable U : Type.

Definition Relation := U -> U -> Prop.

Variable R : Relation.

Definition Contains (R R' : Relation) := forall x y : U, R' x y -> R x y.

.................

Definition Reflexive := forall x : U, R x x.

Definition Transitive := forall x y z : U, R x y -> R y z -> R x z.

.................

  Inductive noetherian : U -> Prop :=
       Build_noetherian :
         forall x : U, (forall y : U, R x y -> noetherian y) -> noetherian x.

   Definition Noetherian : Prop := forall x : U, noetherian x.

Theorem Noetherian_contains_Noetherian :
  forall (U : Type) (R R' : Relation U),
  Noetherian R -> Contains R R' -> Noetherian R'.
Proof.
unfold Noetherian at 2 in |- *.
intros U R R' H' H'0 x.
elim H' with x; auto.
Qed.
```

Un ejemplo

Especificación de la función de Ackermann

```
Inductive Ack  : nat->nat->nat->Prop :=
      AckO : forall n:nat ,(Ack O n (S n))
    | AcknO : forall n m:nat ,(Ack n (S O) m)->(Ack (S n) O m)
    | AckSS : forall n m p q:nat ,
              (Ack (S n) m q)->(Ack n q p)->(Ack (S n) (S m) p).
```

teorema a demostrar

```
Theorem ack: forall n m:nat,{p:nat | (Ack n m p)}.
```

demostración

```
Proof.
induction n; induction m.
 split with 1; constructor 1.
 split with (S (S m)).
   apply (AckO (S m)).
 case (IHn 1); intros.
   split with x; constructor 2.
   auto.
 case IHm; intros.
   case (IHn x); intros.
   split with x0.
   apply (AckSS n m x0 x); auto.
Defined.
```

Si definimos:

```
Definition proj :=
fun (A : Set) (P : A -> Prop) (H : x : A | P x) => let (x, _) := H in x.
```

podremos calcular:

```
Eval compute in (proj nat (fun p : nat => Ack 1 2 p) (ack 1 2)).
     = 4
     : nat
Definition ackerman:=fun n m:nat => (proj nat (fun p:nat => Ack n m p) (ack n m)).
Eval compute in ackerman 3 4.
     = 125
     : nat
```








Categorias en teoría de tipos

```
Variables (Ob : Type) (Hom : Ob -> Ob -> Setoid).
Infix "-->" := Hom (at level 95, right associativity).
Variable Op_comp : forall a b c : Ob, Map2 (a --> b) (b --> c) (a --> c).
Definition Cat_comp (a b c : Ob) (f : a --> b) (g : b --> c) :=
  Op_comp a b c f g.
Infix "o" := Cat_comp (at level 20, right associativity).
Definition Assoc_law :=
  forall (a b c d : Ob) (f : a --> b) (g : b --> c) (h : c --> d),
  f o g o h =_S (f o g) o h.
Variable Id : forall a : Ob, a --> a.
Definition Idl_law := forall (a b : Ob) (f : a --> b), Id _ o f =_S f.
Definition Idr_law := forall (a b : Ob) (f : b --> a), f =_S f o Id _.
End cat.

Structure Category : Type :=
  {Ob :> Type;
   Hom : Ob -> Ob -> Setoid;
   Op_comp : forall a b c : Ob, Map2 (Hom a b) (Hom b c) (Hom a c);
   Id : forall a : Ob, Hom a a;
   Prf_ass : Assoc_law Op_comp;
   Prf_idl : Idl_law Op_comp Id;
   Prf_idr : Idr_law Op_comp Id}.
```

Théorie Constructive des Catégories. *Amokrane Saïbi*. INRIA Roquencourt.

1. Setoides
2. Categories
3. Construcciones categóricas: epis, monos,....
4. Funtores
5. Transformaciones naturales (ley de intercambio)
6. Construcciones universales y límites (construcción de límites, condiciones de completitud...)
7. Adjunciones (categorías cartesianas cerradas) Teorema de Freyd.

Una paradoja en la teoría de tipos.

- tipos inductivos $\rightarrow$ entornos generales
- problema: el entorno

$$[\,(b : Prop);\ (r : b \rightarrow Prop);\ (s : Prop \rightarrow b);\ (\phi : \forall X : Prop \cdot r(s(X)) \leftrightarrow X)\,]$$

  es inconsistente.

- debemos aprender más sobre la consistencia de entornos: la teoría de categorías es un buen marco para formalizar este problema. (Fibred Category Theory)

B. Jacobs. *Categorical Logic and Type Theory*. Studies in Logic and the Foundations of Mathematics. Vol **141** Elsevier (2001)

- Implementación.

```
Variable b : Prop.
Variable r : b -> Prop.
Variable s : Prop -> b.
Variable phi : forall A : Prop, r (s A) <-> A.
```

- Prueba.

```
Definition rs_1 : forall A : Prop, r (s A) -> A.

Definition rs_2 : forall A : Prop, A -> r (s A).

Variable B : Prop.
Definition V := forall P : Prop, ((P -> b) -> P -> b) -> P -> b.
Definition U := V -> b.
Definition sb (z : V) (P : Prop) (f : (P -> b) -> P -> b)
  (p : P) := f (z P f) p.
Definition Le (i : U -> b) (u : U) :=
  u
    (fun (P : Prop) (f : (P -> b) -> P -> b) (p : P) =>
     i (fun v : V => sb v P f p)).
Definition induct (i : U -> b) := forall u : U, r (Le i u) -> r (i u).
Definition WF (z : V) := s (induct (z U Le)).
Definition Y (u : U) :=
  (forall i : U -> b, r (Le i u) -> r (i (fun v : V => sb v U Le u))) -> B.
Lemma Om : forall i : U -> b, induct i -> r (i WF).

Lemma lemma1 : induct (fun u : U => s (Y u)).

Lemma lemma2 : (forall i : U -> b, induct i -> r (i WF)) -> B.

Theorem paradox : B.
```

Interpretación I

- $$\frac{\epsilon : (P \to b) \to P \to b}{\dfrac{\Sigma : P \times (b^P) \to b}{\sigma : P \to b^{b^P}}\ \Sigma(p,f) = ((\sigma(p))\, f)}\ ((\epsilon\, f)\, p) = \Sigma(p,f)$$

- $\epsilon : (P \to b) \to P \to b$ es *adecuado* si existe $t : ((P \to b) \to b) \to P$ tal que para todo $C : (P \to b) \to b$ y para todo $f : P \to b$

$$((\epsilon\, f)\, (t\, C)) \;=\; (C\, (\lambda y : P \cdot (f\, (t\, \overbrace{(\lambda g : P \to b \cdot ((\epsilon\, g)\, y))}))) \qquad (1)$$

Interpretación II

- $\mathcal{K}$ categoría con productos, L objeto de $\mathcal{K}$, un conjunto X. Para cada $x \in X$ sea $\pi_x : L^X \to L$ la correspondiente proyección de L^X a L.

La biyección natural:

$$\left(Hom_{\mathcal{K}^{op}}(L^X, K) = Hom_{\mathcal{K}}(K, L^X)\right) \xrightarrow{\Phi} Hom_{\mathsf{Set}}(X, Hom_{\mathcal{K}}(K, L))$$

$\Phi(f_2 : K \to L^X) = f_1 : X \longrightarrow Hom_{\mathcal{K}}(K, L)$, with K un objeto de $\mathcal{K}$ y $f_1(x) = \pi_x \cdot f_2$ para todo $x \in X$,
establece que el funtor

$$U = Hom_{\mathcal{K}}(-, L) : \mathcal{K}^{op} \longrightarrow \mathsf{Set} \ \ (\text{denotado como } (-, L)),$$

tiene como adjunto a la izquierda:

$$F = L^{(-)} : \mathsf{Set} \longrightarrow \mathcal{K}^{op},$$

definido por $(\pi_x : L^X \to L) \cdot (L^g : L^Y \to L^X) = (\pi_{g(x)} : L^Y \to L), \ \ \forall g : X \longrightarrow Y.$
Su unidad $(\eta : id_{\mathsf{Set}} \Rightarrow U \cdot F)$ y counidad $(\epsilon : F \cdot U \Rightarrow id_{\mathcal{K}^{op}})$ se define por:

$$\eta_X = (id_{L^X})_1 \ \text{ and } \ \epsilon_K = (id_{(K,L)})_2$$

o sea:

$$\eta_X(x) = \pi_x \; \forall x \in X \;\text{ and }\; \psi = \pi_\psi \cdot \epsilon_K \;\text{ for all }\; \psi : K \to L.$$

- Tenemos una mónada $\mathbb{T} = (T = U \cdot F : \text{Set} \to \text{Set},\ \eta,\ \mu : T^2 \Rightarrow T)$ dada por

$$\begin{array}{rcl} T(X) & = & (L^X, L), \\ T(f : X \to Y) & : & (L^X, L) \to (L^Y, L) \text{ es } T(f)(\mathcal{A} : L^X \to L) = \mathcal{A} \cdot L^f, \\ \mu_X(\phi) & = & \phi \cdot \epsilon_{(L^X)} \text{ para todo } \phi \in T^2(X). \end{array}$$

- $$\frac{\beta : L^X \to L^X}{\sigma : X \to L^{L^X} = T(X)}\ \beta(f)(x) = \sigma(x)(f)$$

- $\beta : L^X \to L^X$ es *adecuada* si existe $\tau : T(X) \to X$ tal que para toda $\mathcal{A} : T(X)$ y $f : X \to L$

$$((\beta\, f)\, (\tau\, \mathcal{A})) \;=\; (\mathcal{A}\, (\lambda x : X \bullet (f\, (\tau\, \overbrace{(\lambda g : X \to L \bullet ((\beta\, g)\, x))}))) \qquad (2)$$

Un par $(X, \tau : T(X) \to X)$ se llama *poderoso* (Girard) si existe $\sigma : X \to T(X)$ tal que el siguiente diagrama es conmutativo:

$$\begin{array}{ccc} T^2(X) & \xrightarrow{\;T(\tau)\;} & T(X) \\ {\scriptstyle T(\sigma)}\uparrow & & \uparrow{\scriptstyle \sigma} \\ T(X) & \xrightarrow{\;\tau\;} & X \end{array}$$