Inducción y deducción.
Aplicaciones a la definición y verificación de programas

freire@lfcia.org

"La logique classique est la logique de la vérité absolue idéale. Elle manipule des valeurs de vérité Booléennes: {vrai,faux}. C'est une logique de la description du monde, une logique du discours sur les choses, de preuve par non-contradiction. En bref une logique passive.
La logique intuitionniste ou constructive est une logique de la connaissance. Elle manipule implicitement trois valeurs de vérité: {établi, réfuté, non décidé}. C'est une logique de création de connaissances sur le monde , une logique d'action sur les choses, de calcul de justifications positives. En bref une logique active."
Gerard Huet.

"If programming is understood not as the writings of instructions for this or that computing machine but as the design of methods of computation that it is the computer's duty to execute, then it no longer seems possible to distinguish the discipline of programming from constructive mathematics."
Martin–Löf.

Fundamentos

- Actividad matemática: producir demostraciones a partir de axiomas usando reglas correctas.
- Naturaleza de las pruebas: teoría de la demostración.
  - pruebas constructivas
  - teorema de completitud del cálculo de predicados (Gödel): mecanización la prueba de teoremas en esta lógica de primer orden. (J. A. Robinson).
  - correspondencia de Curry-Howard:
    W. Howard, *The formulae-as-types notion of construction*. En J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda-Calculus, and Formalism*, 479–490. Academic Press, NY, 1980.

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

Sistemas de deducción

- Algunos sistemas: (http://www-formal.stanford.edu/clt/ARS/systems.html)

  N. de Bruijn: proyecto **Automath**; **LCF** de R. Milner **ACL2**; **HOL**; **ALF**; **Izabelle**; Boyer y Moore **Theorem Prover** (R. S. Boyer, J. S. Moore. *A Computational Logic*. Academic Press. 1979; R. S: Boyer, J. S. Moore. *A Computational Logic Handbook*. Academic Press, 1988); **Coq**; **Theorema** de B. Buchberger; **Nuprl** de R. Constable; **PVS** (Prototype Verification System) de Sam Owre y otros (SRI Computer Science Lab.); etc.

Curry-Howard

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

Así pues, la regla *modus ponens* se corresponde con la instrucción *composición*

Nace de:

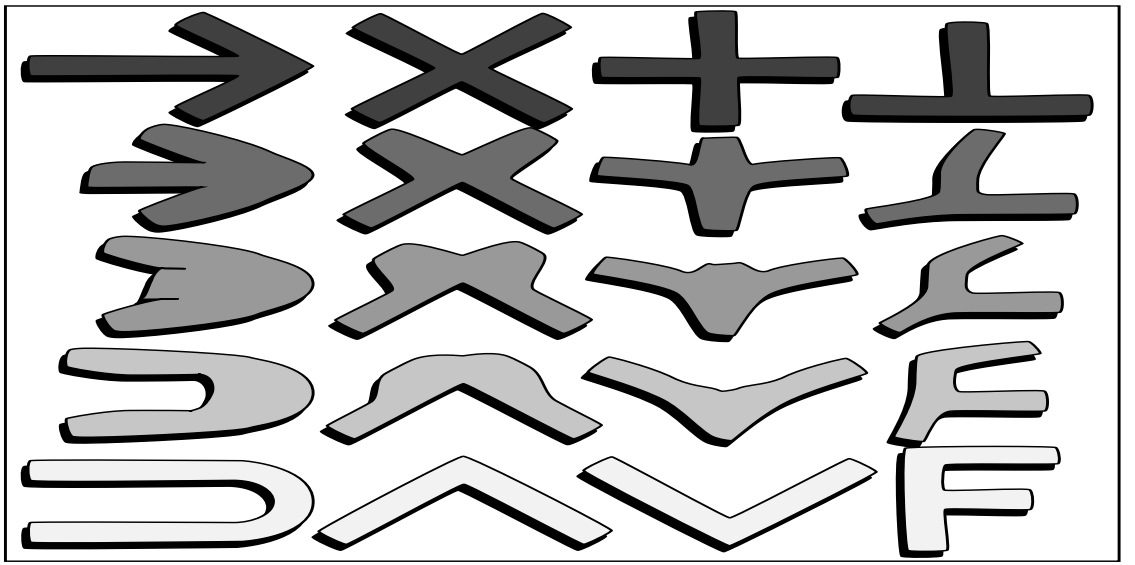
Semántica de Heyting y Kolmogorof:

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.		

Germen de:

- Teoría intuicionista de tipos (Martin-Löf)
- λ–cálculo polimórfico F_ω (Girard)

CH-homeomorfismo



LC'90 *The Curry-Howard homeomorphism*

CIC

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. Una expresión $a : A$ puede interpretarse, o bien como que a es una "prueba" de la proposición A o bien como que el elemento a pertenece a (tiene tipo) A.

  El lenguaje consta de:

  - constantes
  - clases (*sorts*): $S \equiv \{Prop, Set, Type(i) \,|\, i \in \mathbf{N}\}$. **Impredicativo**: $Prop$ y predicativos Set y **los universos**: $Type(i)$'s. Existe la posibilidad de considerar Set impredicativo aunque por defecto no lo sea.
  - variables
  - si x es una variable y T y U son términos, entonces

    $$forall\, x : T\ ,\ U$$

    es un término (de hecho es un tipo) llamado *producto dependiente* . En general x puede estar presente en U. Este tipo puede interpretarse como la prosición $\forall x$:

$T.U$ "para todo s en T se verifica U. o como el conjunto de los programas que toman argumentos en T y devuelven valores de tipos $U(x)$ y que no es sino el producto cartesiano

$$\prod_{x:T} U(x)$$

Si x no está presente en U, entonces se lee simplemente o como la proposición *"si T entonces U"* o como el tipo que representa el conjunto de todas las funciones de T a U, y se puede escribir

$$T \to U$$

(producto *no dependiente*).

- si x es una variable y T y t son términos, entonces

$$\mathit{fun}\,(x : T) \Rightarrow t$$

es un término. Esta notación que representa la λ–abstracción del λ–cálculo se interpreta como la función (o programa en general) que toma un argumento x en T y nos devuelve el término t.

- si t y u son términos entonces $(t\ u)$ es un término (aplicación).

Reglas de cálculo

- β–reducción:

$$((\mathit{fun}\,(x : T) \Rightarrow t)\ u) \quad \triangleright_\beta \quad t\{x/u\}$$

  Esta reducción es *confluente* y *finitaria*.

- Otras: δ–reducción, ι–reducción.

Tipado

CIC es un λ–cálculo tipado con un sistema de reglas de deducción natural para derivar expresiones de la forma:

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

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

- Entornos y contextos

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

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

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

- **impredicativo**: $Prop$
- **predicativo**: 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:

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

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

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

$$\frac{E[\Gamma] \vdash T : s \quad E[\Gamma :: (x : T)] \vdash U : Prop \quad s \in S}{E[\Gamma] \vdash \mathit{forall}\, x : T\,,\, U : Prop}\, Prod - 1$$

$$\frac{E[\Gamma] \vdash T : s \quad E[\Gamma :: (x : T)] \vdash U : Set \quad s \in \{Prop, Set\}}{E[\Gamma] \vdash \mathit{forall}\, x : T\,,\, U : Set}\, Prod - 2$$

$$\frac{E[\Gamma] \vdash T : Type(i) \quad i \leq k \quad E[\Gamma :: (x : T)] \vdash U : Type(j) \quad j \leq k}{E[\Gamma] \vdash \mathit{forall}\, x : T\,,\, U : Type(k)}\, Prod - 3$$

$$\frac{E[\Gamma] \vdash \mathit{forall}\, x : T\,,\, U : s \quad E[\Gamma :: (x : T)] \vdash t : U}{E[\Gamma] \vdash \mathit{fun}\,(x : T) \Rightarrow t : \mathit{forall}\, x : T\,,\, U}\, Lam$$

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

Dados tipos T y U, escribimos

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

para expresar la convertibilidad de tipos. Entonces:

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

Tácticas