El Isomorfismo de Curry-Howard

Mario Román

mromang08+github@gmail.com

Universidad de Granada

6 de noviembre de 2014

Overview

La lógica intuicionista

La lógica intuicionista se diferencia de la lógica clásica en que usa una **noción constructivista de verdad**. En particular, el enunciado $A \vee B$ sólo puede ser demostrado si se construye alguna prueba de A o de B. En consecuencia, la ley del tercio excluso $A \vee \neg A$, no puede demostrarse.

Aun así, puede introducirse como axioma posteriormente:

Axioma del tercio excluso

Puede hallarse siempre evidencia de A o de $\neg A$. En particular, para cualquier proposición A puede hallarse evidencia de:

$$A \vee \neg A$$

La deducción natural

En 1935, Gerhard Genzen publicó dos nuevas formulaciones de la lógica intuicionista, siendo una de ellas la **deducción natural**. Junto a ellas estableció un método de simplificación de demostraciones, que servía para asegurarse que la demostración no daba vueltas innecesarias.

Para asegurarse de esto, estableció que en la demostración normalizada de una fórmula sólo podían aparecer sus subfórmulas. Por ejemplo, para demostrar $A \wedge B$, sólo podían usarse A, B y sus subexpresiones, nunca algo como $A \vee B$. Este método asegura además la consistencia. No existe forma de demostrar $\bot$, la proposición contradicción (False).

Reglas de la deducción natural

La deducción natural posee las siguientes reglas:

$$\frac{\mathtt{A} \qquad \mathtt{B}}{\mathtt{A}\ \&\ \mathtt{B}}\ (\&-I)$$

$$\frac{\mathtt{A}\ \&\ \mathtt{B}}{\mathtt{A}}\ (\&\text{-}E_1) \qquad \frac{\mathtt{A}\ \&\ \mathtt{B}}{\mathtt{B}}\ (\&\text{-}E_2)$$

$$\frac{\mathtt{A} \to \mathtt{B} \qquad \mathtt{A}}{\mathtt{B}}\ (\to\text{-}E) \qquad \frac{\begin{array}{c}[\mathtt{A}]^x \\ \vdots \\ \mathtt{B}\end{array}}{\mathtt{A}\to\mathtt{B}}\ (\to\text{-}I^x)$$

El cálculo lambda tipado

En 1940, Alonzo Church publicó una nueva formulación del cálculo lambda, siendo esta el cálculo lambda tipado. Junto a ella podía usarse la simplificación de programas con estrategias de reducción.

Para asegurarse de que el cálculo lambda no hacía aparecer ciertas paradojas, introdujo los tipos y limitó los constructores de tipos a uno solo: el operador $\rightarrow$, que construye el tipo función entre dos tipos. Por esto, puede denotarse por cálculo $\lambda^{\rightarrow}$.

Reglas del cálculo lambda tipado

El cálculo lambda tipado posee las siguientes reglas:

$$\frac{\mathtt{A::M} \qquad \mathtt{B::N}}{\mathtt{A\times B::(M,N)}}\ (\times - I)$$

$$\frac{\mathtt{A\times B::(M,N)}}{\mathtt{A::M}}\ (\pi_1)$$

$$\frac{\mathtt{A\times B::(M,N)}}{\mathtt{B::N}}\ (\pi_2)$$

$$\frac{\lambda \mathtt{A.B::M\to N} \qquad \mathtt{A::M}}{\mathtt{B::N}}\ (\lambda\text{-}E)$$

$$\frac{\begin{array}{c}[\mathtt{A::M}]^x \\ \vdots \\ \mathtt{B::N}\end{array}}{\lambda A.B\ ::\ M \to N}\ (\lambda\text{-}I^x)$$

Las proposiciones son tipos

De todo esto sacamos una conclusión: **las proposiciones lógicas son tipos**. A cada proposición lógica le corresponde un tipo, unívocamente. Y una proposición será verdadera si y sólo si su tipo correspondiente está habitado. Es decir, existe alguna instancia de ese tipo.

- Los tipos conocidos y habitados son los **axiomas** (`int`, `char`, `string`).
- Una función de tipos hace que uno implique otro (`f :: a -> b`, si `a` está poblado por `x`, `f(x)` puebla `b`). Es la **implicación** lógica.
- El producto está poblado ssi ambos lo están (hay un objeto `(a,b)` por cada pareja `a` y `b`). Es la **conjunción** lógica.

El isomorfismo en Haskell

El sistema de tipos de Haskell permite denotar por a un tipo arbitrario. Denota con -> el tipo de las funciones entre dos elementos. Sólo algunos tipos están habitados.

Tipos permitidos

```
a -> a                    -- identity x = x
a -> (b -> a)             -- const a = (\b->a)
(a,b) -> a                -- proy (a,b) = a
(a, (a -> b)) -> b        -- apply f x = f(x)
```

Tipos no permitidos

```
a -> (a -> b)
a -> b
b -> (b,c)
```

El isomorfismo en Coq

El asistente de demostraciones Coq consiste en un lenguaje de programación y un intérprete del lenguaje destinados a demostrar proposiciones lógicas. Para esto, usa el isomorfismo en sentido inverso, un objeto de un tipo determinado es una demostración de una proposición determinada.

Demostración en Coq

```
(* Theorem / Type declaration*)
Theorem eight_is_beautiful : beautiful 8.
(* Proof / Construction *)
Proof.
  apply b_sum with (n:=3) (m:=5).
  apply b_3.
  apply b_5.
Qed.
```

References

Benjamin Pierce et al. (2014)

Software Foundations.

University of Pennsylvania

Philip Walder

Propositions as Types

University of Edimburg

Morten Heine B. Sørensen, Paweł Urzyczyn,

Lectures on the Curry-Howard isomorphism

Universities of Warsaw and Copenhagen