pty.1 Types

int:pty:typ: sec

A proof term such as (MN) by itself does not uniquely determine the proof it is a proof term of. That's because the free variables, i.e., the proof terms of axioms, do not specify the formulas in the corresponding axioms. However, if we specify these in a context Γ, the proof *is* uniquely determined.

Definition pty.1. A *context* for a proof term N is a set Γ such that for every free variable of N, Γ contains exactly one pair $x : \varphi$.

Note that the definition does not require that Γ contains $x : \varphi$ *only* for free variables of N. This, e.g., $\Gamma = \{x : \varphi, y : \psi, z : \chi\}$ is a context for $\langle x, y\rangle$.

int:pty:typ: defn:type

Definition pty.2. Suppose Γ is a context for N. The *type* of N (relative to Γ) is inductively defined as follows:

1. The type of x is φ iff $x : \varphi \in \Gamma$.
2. If N has type ψ relative to $x : \varphi, \Gamma$, then the type of $\lambda x^{\varphi}. N$ is $\varphi \to \psi$.
3. If N has type $\varphi \to \psi$ and M has type φ then (NM) has type ψ
4. If N has type φ and M has type ψ, then $\langle N, M\rangle$ has type $\varphi \wedge \psi$.
5. If N has type $\varphi_1 \wedge \varphi_2$, then $\pi_i(N)$ has type φ_i.
6. If N has type φ, then $\iota_i^{\psi}(N)$ has type $\varphi \vee \psi$ if $i = 1$ and type $\psi \vee \varphi$ if $i = 2$.
7. If M has type $\varphi \vee \psi$, N_1 has type χ relative to $x_1 : \varphi, \Gamma$, and N_2 has type χ relative to $x_2 : \psi, \Gamma$, then $\delta\, M\, x_1.N_1\, x_2.N_2$ has type χ.
8. If N has type $\bot$, then $\varepsilon^{\varphi}(N)$ has type φ.

Proposition pty.3. *Every proof term N has exactly one type relative to a context Γ for N, if it has a type at all (i.e., is a correct proof term).*

Proof. By induction on N. □

We write $\Gamma \Rightarrow N : \varphi$ if the type of N relative to Γ is φ. The clauses in Definition pty.2 should seem familiar: they are precisely the rules of **tN2i** with one slight difference: the context Γ is the same throughout. We can codify the definition as a calculus **tN3i** with the rules given in Table 1.

Proof terms are terms in a version of the λ-calculus, the *typed λ-calculus with products, sums, and empty type*. The classical λ-calculus only has term built from variables, λ-*abstracts* $\lambda x. N$ and application (MN), and it lacks the annotation by φ of the typed λ-abstract $\lambda x^{\varphi}. N$. The λ-calculus is a model of computation which is Turing complete (i.e., every partial computable function can be represented in it by a term). The version of the λ-calculus given by proof terms and typing rules is a restricted model of computation, but the basic ideas

Axioms:

$$\Gamma \Rightarrow x : \varphi \text{ if } x : \varphi \in \Gamma$$

Inference Rules:

$$\frac{x : \varphi, \Gamma \Rightarrow N : \psi}{\Gamma \Rightarrow \lambda x^{\varphi}.\, N : \varphi \to \psi} \to\text{Intro}$$

$$\frac{\Gamma \Rightarrow N : \varphi \to \psi \qquad \Gamma \Rightarrow M : \varphi}{\Gamma \Rightarrow (NM) : \psi} \to\text{Elim}$$

$$\frac{\Gamma \Rightarrow N_1 : \varphi \qquad \Gamma \Rightarrow N_2 : \psi}{\Gamma \Rightarrow \langle N_1, N_2 \rangle : \varphi \wedge \psi} \wedge\text{Intro}$$

$$\frac{\Gamma \Rightarrow N : \varphi \wedge \psi}{\Gamma \Rightarrow \pi_1(N) : \varphi} \wedge\text{Elim}_1 \qquad \frac{\Gamma \Rightarrow N : \varphi \wedge \psi}{\Gamma \Rightarrow \pi_2(N) : \psi} \wedge\text{Elim}_2$$

$$\frac{\Gamma \Rightarrow N : \varphi}{\Gamma \Rightarrow \iota_1^{\psi}(N) : \varphi \vee \psi} \vee\text{Intro}_1 \qquad \frac{\Gamma \Rightarrow N : \varphi}{\Gamma \Rightarrow \iota_2^{\psi}(N) : \psi \vee \varphi} \vee\text{Intro}_2$$

$$\frac{\Gamma \Rightarrow M : \varphi \vee \psi \qquad x : \varphi, \Gamma \Rightarrow N_1 : \chi \qquad y : \psi, \Gamma \Rightarrow N_2 : \chi}{\Gamma \Rightarrow \delta\, M\, x.N_1\, y.N_2 : \chi} \vee\text{Elim}$$

$$\frac{\Gamma \Rightarrow N : \bot}{\Gamma \Rightarrow \varepsilon^{\varphi}(N) : \varphi} \bot_I$$

Table 1: Rules of the propositional term-labelled intuitionistic natural deduction system **tN3ip** serving as a type assignment system.

int:pty:typ:
tab:tN3ip

are the same. It is restricted by the type assignment: only *well-typed* terms, i.e., proof terms with types (relative to a context Γ) are legal terms. For instance, $\lambda x.\,(xx)$ is a legal term in the classical untyped λ-calculus, but $\lambda x^{\varphi}.\,(xx)$ is not well-typed for any φ. That's because the typing rules require that in order for (xx) to have a type, the first x would have to have type $\varphi \to \psi$ and the second x type φ, and that's impossible since φ is a proper sub-formula of $\varphi \to \psi$.

We can think of types intuitively as the kind of thing named by a term of that type. So a term of type $\varphi \wedge \psi$ picks out a pair with first component of type φ and second component of type ψ, i.e., an element of the product $\varphi \times \psi$. A term of type $\varphi \to \psi$ is a function from things of type φ to things of type ψ. A term of type $\varphi \vee \psi$ is either something of type φ or something of type ψ, plus the information which one it is. This is like the disjoint union of the two sets φ and ψ defined as $\{\langle i, x\rangle : i = 1 \text{ or } 2, x \in \varphi \text{ if } i = 1, x \in \psi \text{ if } i = 2\}$. The type $\bot$ is the empty set, i.e., if a term has that type it can't name anything. Since natural deduction (including **tN3i**) is consistent, there is no derivation

of $\bot$ unless there are open assumptions, and so there is no closed proof term (term without free variables) that has type $\bot$.

The operators of the typed λ-calculus produce terms that intuitively name the right sort of thing, provided the terms it is applied to name certain things. E.g., "if $N_1 : \varphi$ and $N_2 : \psi$, then $\langle N_1, N_2 \rangle : \varphi \wedge \psi$" says that if N_1 names somethign of type φ and N_2 names something of type ψ, then $\langle N_1, N_2 \rangle$ names something of type $\varphi \wedge \psi$.

Photo Credits

Bibliography