pty.1 Introduction

int:pty:int:sec

The lambda calculus is a simple model of computation. It operates on terms built from variables. If N and M are terms, then (NM) is also a term ("N applied to M"). If N is a term, then $\lambda x.\, N$ is a term. If N contains the variable x, then the corresponding occurrences of x in $\lambda x.\, N$ are bound by the λx operator, and so are no longer free. A term of the form $(\lambda x.\, NM)$ is said to β-contract to $N[M/x]$ (the result of replacing every free occurrence of x in N by M). A term N β-converts to N' if N' results from N by β-contracting a subterm, and we write $N \to N'$. A computation in the λ-calculus is a sequence $N \to N_1 \to N_2 \to \ldots$. If this sequence ends in a term N_k where no subterm can be β-contracted, it is called *normal* or a normal form of N. Think of computation in the λ-calculus as such sequences. The computation halts if it results in a normal form, and the normal form is the result of the computation. This simple model of computations turns out to be Turing complete: every partial recursive function can be computated by a term in the λ-calculus.

Haskell Curry and William Howard independently discovered a close similarity between natural deduction and the λ-calculus. Intuitively, a term $\lambda x.\, N$ is a function: x is the argument; N tells us how to compute the value given x. A term $(\lambda x.\, NM)$ then is applying the function $\lambda x.\, N$ to an argument M, and β-contraction tells us how to evaluate it: replace x in N by M (and keep evaluating the resulting term). Now think of these as functions with a fixed domain and range. If the domain of $\lambda x.\, N$ is A and its range B, then we should think of the variable x as only taking arguments from A and N as producing values from B. Consequently, $(\lambda x.\, NM)$ only should make sense if $\lambda x.\, N$ is a function $A \to B$, and $M \in A$. If we want to add this information to the λ-calculus we obtain the *typed* λ-calculus. In the typed λ-calculus not every expression $(\lambda x.\, NM)$ is legal, but only those where the "types" of $\lambda x.\, N$ and M fit together, i.e., $\lambda x.\, N$ has type $A \to B$ and M has type A. The type of $\lambda x.\, N$ is $A \to B$ only if the type of x is A and the type of N is B. In general, not every expression $(M_1 M_2)$ is legal. Only those terms $(M_1 M_2)$ are legal where M_1 has type $A \to B$ and M_2 has type A; if so, then $(M_1 M_2)$ has type B.

These typing rules now clearly correspond to derivations in natural deduction, where the types are represented by formulas, and the derivations themselves to λ-terms. For instance, a derivation of $\varphi \to \psi$ corresponds to a term $\lambda x^{\varphi}.\, N$, which has the function type $\varphi \to \psi$. The inference rules of natural deduction correspond to typing rules in the typed lambda calculus, e.g.,

$$\dfrac{\begin{array}{c}[\varphi]^x \\ \vdots \\ \psi\end{array}}{\varphi \to \psi}\ {\scriptstyle x}\ {\to}\text{Intro} \qquad \text{corresponds to} \qquad \dfrac{x : \varphi \;\Rightarrow\; N : \psi}{\Rightarrow\; \lambda x^{\varphi}.\, N : \varphi \to \psi}$$

where the rule on the right means that if x is of type φ and N is of type ψ, then $\lambda x^{\varphi}.\, N$ is of type $\varphi \to \psi$.

The →Elim rule corresponds to the typing rule for composition terms, i.e.,

$$\frac{\varphi \to \psi \qquad \varphi}{\psi} \to\text{Elim} \quad \text{corresponds to} \quad \frac{\Rightarrow M_1 : \varphi \to \psi \qquad \Rightarrow M_2 : \varphi}{\Rightarrow (M_1 M_2) : \psi}$$

If a →Intro rule is followed immediately by a →Elim rule, the derivation can be simplified:

$$\begin{array}{c} [\varphi]^x \\ \vdots \\ \psi \\ \hline {\scriptstyle x}\ \varphi \to \psi \end{array} \to\text{Intro} \qquad \begin{array}{c} \vdots \\ \varphi \end{array} \qquad \to\text{Elim} \qquad \Rightarrow \qquad \begin{array}{c} \vdots \\ \varphi \\ \vdots \\ \psi \end{array}$$

which corresponds to β-contraction of lambda terms

$$(\lambda x^{\varphi}.\, NM) \quad \to \quad N[M/x].$$

Similar correspondences hold between the rules for $\wedge$ and "product" types, and between the rules for $\vee$ and "sum" types.

This correspondence between terms in the simply typed lambda calculus and natural deduction derivations is called the "Curry–Howard", "proofs-as-programs," or "propositions as types" correspondence. In addition to formulas (propositions) corresponding to types, and proofs to terms, we can summarize the correspondences as follows:

logic	program
proposition/formula	type
proof	term
assumption	variable
discharged assumption	bound variable
undischarged assumption	free variable
$\varphi \to \psi$	function type
$\varphi \wedge \psi$	product type
$\varphi \vee \psi$	sum type
$\bot$	empty type

The Curry–Howard correspondence is one of the cornerstones of automated proof assistants and type checkers for programs, since checking a proof witnessing a proposition (as we did above) amounts to checking if a program (term) has the declared type.

Photo Credits

Bibliography