A TYPE-THEORETICAL CURRY PARADOX AND ITS SOLUTION

ANSTEN KLEV

Abstract. The Curry–Howard correspondence, according to which propositions are types, suggests that every paradox formulable in natural deduction has a type-theoretical counterpart. I will give a purely type-theoretical formulation of Curry's paradox. On the basis of the definition of a type $\Gamma(A)$, Curry's reasoning can be adapted to show the existence of an object of the arbitrary type A. This is paradoxical for several reasons, among others that A might be an empty type. The solution to the paradox consists in seeing that $\Gamma(A)$ is not a well-defined type.

Keywords—Curry's paradox; type theory; inductive definitions; functions.

1. Introduction

Haskell Curry formulated the paradox that now bears his name in terms of his system of combinatory logic. Combinatory logic takes for granted an untyped universe of formal objects, or "obs" in Curry's terminology. Curry, in his book on combinatory logic (Curry and Feys 1958), presents his paradox as motivation for the introduction of types. I will show that, even in a typed universe, a Curry paradox can be produced.

The paradox will be formulated within a theory of inductively defined types. These are types whose objects may be thought of as having been built up in a stepwise manner. Examples include the type of natural numbers, the type of booleans, and the product of two inductively defined types. Starting from what might look like an inductively defined type, $\Gamma(A)$, Curry's reasoning shows—constructively—that there is an object of type A. This is paradoxical, since A is arbitrary and no properties of A are appealed to in the reasoning. In particular, A might be a type that by definition is empty.

Under the Curry–Howard correspondence, an object of type A is a proof of the proposition A, and A is true just when it is inhabited as a type (Howard 1980; Martin-Löf 1984). Curry's reasoning shows, without appeal to any property of A other than its being a proposition, that A is true. This is the more familiar logical Curry paradox.

Both the Russell paradox and the Burali-Forti paradox can be formulated in a type theory of the kind considered below (Coquand 1986; 1992; Martin-Löf 1998). These paradoxes, however, rely on an axiom to the effect that there is a type of all types. The Curry paradox is derived without appeal to this dubious axiom, whence its diagnosis must point to some other flaw. I propose to solve the paradox by denying that $\Gamma(A)$ is a well-defined type. Circularity is a feature of many inductive

definitions. The putative definition of $\Gamma(A)$, however, is viciously circular. Its circularity is as damning as the circularity in, say, this definition:

$$g(n) := 1 + g(n)$$

Both definitions fail to bring about understanding of what they are intended to define.

Essentially the same solution has been proposed by Dyckhoff (2016) and Pezlar (2021) to proof-theoretical formulations of the liar paradox. Readers familiar with the theory of inductive definitions might note that $\Gamma(A)$ has a so-called negative occurrence in its purported definition and that it therefore breaks a well-known syntactic criterion on the form of such definitions (Aczel 1977; Feferman and Sieg 1981; Dybjer 1994; 2000). To note this is, however, not yet to solve the type-theoretical Curry paradox, since the breach of a purely syntactic criterion does not explain why an inconsistency arises.

The type-theoretical Curry paradox shows that forms of constructive concept formation that on first sight might look sound can lead to inconsistency. The purported definition of $\Gamma(A)$ below is quite similar in form to the definition of the type of hereditarily countable sets, called $\mathbb{HC}$ below, but whereas the latter is a good definition, the former is not. In Section 7 below, I offer an explanation of why the one definition succeeds and the other fails based on an analysis of how an inductive definition is meant to work, how it is meant to bring about understanding of its definiendum. This analysis is, in turn, based on Martin-Löf's meaning explanations for his type theory (Martin-Löf 1982; 1984; 1993).

It is common in recent literature to solve the logical Curry paradox by a revision either of the laws governing implication or of the laws governing the consequence relation (Shapiro and Beall 2021). Curry himself contrasted such solutions with the solution he advocated, namely to seek a deeper analysis of the notion of proposition. Whereas for Curry, this remained only a programmatic remark, I offer a precise conception of propositions through the Curry–Howard isomorphism and solve the paradox on this basis. There is no need then to revise basic logical laws.

I have sought to avoid technicalities and jargon as much as possible. In particular, I do not employ the terminology of introduction and elimination rules. (The final paragraph of Section 4 explains how my presentation of type theory may be understood in terms of those rules.) Types are defined directly through the clauses making up an inductive definition. Such a clause says that if the object a is of type α, then there is an object $c(a)$ of type β, the type to be defined. Allowing for more premisses, such a clause looks in formal notation as follows:

$$\frac{a_1 : \alpha_1 \quad \dots \quad a_n : \alpha_n}{c(a_1, \dots, a_n) : \beta}$$

With zero premisses, the clause takes the form $c : \beta$, where c is atomic. I will use this formal notation, since it is concise and intuitive.

2. Preliminaries on type theory

In the current philosophical literature, type theory is often equated with some version of simple type theory. There are a small number of ground types, including a type of individuals, and function types generated from these. For instance, in the

system of Church (1940), there is a type ι of individuals, a type o of propositions, and a type $(\alpha)\beta$ of functions from α to β, whenever α and β are types. The type ι is meant to encompass *all* individuals, including the natural numbers, the booleans, binary strings, and sets in the cumulative hierarchy—in short, every object that can be in the range of a first-order variable. (Throughout this paper my interest will lie with mathematical and logical objects.)

Under a different conception of type theory, there are many, perhaps infinitely many, types of individuals, each of which is a homogeneous domain of objects. For instance, each of the familiar number domains—natural numbers, integers, rational numbers, real numbers—might count as a separate type. There are different ways of realizing type theory in this sense. I shall identify types of individuals with inductively defined domains. Examples will be given in Sections 3 and 5 below.

Besides the types of individuals, there are function types. Whenever α and β are types, there is a type, $(\alpha)\beta$, of unary functions from α to β. The following terminology will be used in Sections 6 and 7 below: α is the *domain* and β the *codomain* of the function type $(\alpha)\beta$.

I will write $a : \alpha$ to mean that a is an object of type α. The defining rule of the function type $(\alpha)\beta$ is the rule of function application,

$$\text{(App)} \qquad \frac{f : (\alpha)\beta \qquad a : \alpha}{f(a) : \beta}$$

In prose, this rule says that a function f from α to β may be applied to every element a of α, and the result $f(a)$ is an object of type β. Since (App) does not have the form of an inductive definition, the function type $(\alpha)\beta$, being defined by this rule, is not an inductively defined type.

In mathematical writing, a function is often expressed in terms of free variables, each variable corresponding to an argument of the function. For example, x^2 is the squaring function, and $x + y$ is the addition function. A function in this sense will be written schematically as $e[x]$ or $e[x, y]$. By giving values to the variables in such a function, we obtain a function value. For instance, if x is given the value 3, we obtain 3^2 as value of the squaring function. Function values are written schematically as $e[a]$ or $e[a, b]$.

If the variable x is of type α and the values of $e[x]$ are of type β, we naturally speak of $e[x]$ as a function from α to β. It is, however, not yet an object of type $(\alpha)\beta$ (Klev 2019b). In order to obtain an object of this type we must apply functional abstraction on $e[x]$, the result of which I shall write, following Curry, as $[x].e[x]$. The Greek letter lambda, often used to indicate functional abstraction, will be used for a somewhat different purpose below. The function $[x].e[x]$ is defined by the following equation:

$$[x].e[x](a) := e[a]$$

In prose: the result of applying the function $[x].e[x]$ to a is $e[a]$. (This is the beta rule of the typed lambda calculus.)

3. Inductively defined types: three examples

The type $\mathbb{N}$ of natural numbers is inductively defined by the two clauses that (i) 0 is a natural number and (ii) if n is a natural number, then so is its successor,

$\mathrm{s}(n)$. In formal notation these two clauses are written as follows:

$$0 : \mathbb{N} \qquad \frac{n : \mathbb{N}}{\mathrm{s}(n) : \mathbb{N}}$$

We take it for granted that neither clause is redundant. More specifically, we take it for granted that the object 0, postulated by the first clause, is not also postulated by the second clause—that is, 0 is not of the form $\mathrm{s}(n)$. That $\mathbb{N}$ is defined by these clauses means that every object of type $\mathbb{N}$ is equal to a number constructed by means of these two clauses. For instance, since $2 + 2$ is equal to $\mathrm{s}(\mathrm{s}(\mathrm{s}(0)))$, it is a natural number. These remarks apply, mutatis mutandis, as well to the two following examples.

The type $\mathbb{B}$ of booleans is inductively defined by the two clauses that (i) $\mathtt{t}$ is a boolean and (ii) $\mathtt{f}$ is a boolean. In formal notation,

$$\mathtt{t} : \mathbb{B} \qquad \mathtt{f} : \mathbb{B}$$

The type $\mathbb{HC}$ of hereditarily countable sets is defined by the two clauses that (i) the empty set is hereditarily countable, and (ii) if a_i is a sequence of hereditarily countable sets indexed by $\mathbb{N}$, then the set $\{a_i \mid i : \mathbb{N}\}$ is also hereditarily countable. In formal notation,

$$\emptyset : \mathbb{HC} \qquad \frac{f : (\mathbb{N})\mathbb{HC}}{\mathrm{set}(f) : \mathbb{HC}}$$

Think of $\mathrm{set}(f)$ as the set of all the sets enumerated by the function f. For instance, if $f(i) = \emptyset$ for all $i : \mathbb{N}$, then $\mathrm{set}(f)$ is the singleton set $\{\emptyset\}$.

These three examples serve to illustrate that an inductive definition of a domain D shows how the elements of D can be built up, or constructed, from below. The natural numbers are built up, starting from 0, by iterated application of the successor function. Booleans are built up in the trivial way that two base elements, $\mathtt{t}$ and $\mathtt{f}$, are stipulated. Hereditarily countable sets are built up from the empty set by means of functions into the hereditarily countable sets: from any such function, f, one can construct the set of sets enumerated by it, written $\mathrm{set}(f)$.

To this gloss on the nature of inductive definitions one might object that mathematical objects are not in fact constructed or built up from below. Mathematical objects exist independently of us, and we can neither construct nor destruct them. An alternative gloss avoids any appeal to construction: an inductive definition of D displays the compositional structure of the elements of D. A natural number either has the structure 0 or the structure $\mathrm{s}(m)$, where m is a natural number. A boolean has the structure $\mathtt{t}$ or $\mathtt{f}$. A hereditarily countable set has the structure $\emptyset$ or $\mathrm{set}(f)$ where f is a sequence of hereditarily countable sets. This more static, and philosophically less committing, view of inductive definitions will be assumed here. The dynamic view of inductive definitions, invoking the notion of construction, is nevertheless useful to keep in mind as a heuristic.

4. Functions defined by induction

Over any inductively defined domain one is justified in defining functions by induction. Suppose that the variable x is of type $\mathbb{N}$ and that t and all the values of $u[x]$ are of type A. A basic definition by induction of a function $f : (\mathbb{N})A$ takes

the form of two equations,

$$f(0) := t$$
$$f(\mathrm{s}(x)) := u[x]$$

To see that this form of definition is justified, note that every natural number is equal either to 0 or to some $\mathrm{s}(m)$, where m is a natural number, hence for every natural number, n, one of the two equations above determines the value of $f(n)$. Moreover, since 0 is not of the form $\mathrm{s}(m)$, a unique value is determined. An example is the definition of the predecessor function, from $\mathbb{N}$ to $\mathbb{N}$,

$$\mathrm{pd}(0) := 0$$
$$\mathrm{pd}(\mathrm{s}(x)) := x$$

A stronger form of definition by induction involves recursion and allows the defined function to have any finite number of arguments, not only one argument. The definitions of the binary functions of addition and multiplication are well-known examples. Since recursion is not needed in what follows, I shall not go into the details.

To define a function, f, by induction on $\mathbb{B}$, one must specify the value of $f(\mathtt{t})$ and the value of $f(\mathtt{f})$.

To define a function, f, by induction on $\mathbb{HC}$, one must specify the value of $f(\emptyset)$ and the value of $f(\mathrm{set}(g))$, for any $g : (\mathbb{N})\mathbb{HC}$.

The following remark is intended for readers familiar with the type-theoretical terminology of introduction, elimination, and equality rules (Martin-Löf 1984). The introduction rule for a type A is an inductive definition in the sense of this paper. Its elimination rule may be formulated as the permission to define functions by induction on A (Martin-Löf 1975), in which case the defining equations of any such function are instances of the equality rule for A.

5. Propositions as types

Under the Curry–Howard correspondence, propositions are inductively defined types: a proposition is identified with the type of its truthmakers. Let us consider conjunction, implication, and absurdity as examples. (Conjunction will not be needed later.)

Assume that A and B are propositions. The proposition $A \wedge B$ is inductively defined by the following clause:

$$\frac{a : A \qquad b : B}{\langle a, b\rangle : A \wedge B}$$

The notation "$a : A$" is now to be read as saying that a is a truthmaker of the proposition A (Sundholm 1994). That this clause defines the proposition $A \wedge B$ means that each of its truthmakers is equal to a truthmaker of the form $\langle a, b\rangle$, where a is a truthmaker of A and b is a truthmaker of B.

Since $A \wedge B$ is an inductively defined type, we may define a function by induction on it, namely by specifying the function value for all truthmakers of the form $\langle a, b\rangle$.

Examples are the definitions of the two projection functions:

$$\mathrm{fst}(\langle a, b\rangle) := a$$
$$\mathrm{snd}(\langle a, b\rangle) := b$$

Whenever c is a truthmaker of $A \wedge B$, then $\mathrm{fst}(c)$ is a truthmaker of A, and $\mathrm{snd}(c)$ is a truthmaker of B. The two rules of conjunction elimination are therefore justified:

$$\frac{c : A \wedge B}{\mathrm{fst}(c) : A} \qquad \frac{c : A \wedge B}{\mathrm{snd}(c) : B}$$

If f is of type $(A)B$, then $\lambda(f)$ is a truthmaker of $A \supset B$, and any truthmaker of $A \supset B$ is equal to a truthmaker of the form $\lambda(f)$. In other words, the proposition $A \supset B$ is inductively defined by the following clause:

$$\frac{f : (A)B}{\lambda(f) : A \supset B}$$

One may think of $\lambda(f)$ as the graph of the function f, or more abstractly as an individual coding this function (Klev 2019a: 286).

To define a function by induction on $A \supset B$ it suffices to specify its value for every truthmaker of the form $\lambda(f)$. In particular, we may define, for every $a : A$, the function ap_a,

$$\mathrm{ap}_a(\lambda(f)) := f(a)$$

The binary application function, ap, may then be explicitly defined as follows:

$$\mathrm{ap}(c, a) := \mathrm{ap}_a(c)$$

Modus ponens is therefore justified:

$$\frac{c : A \supset B \qquad a : A}{\mathrm{ap}(c, a) : B}$$

The proposition $\bot$ (absurdity) is inductively defined by the empty set of clauses: there is no clause postulating a truthmaker of $\bot$. It follows that $\bot$ has no truthmakers. To define a function by induction on $\bot$, no values at all need to be specified, hence for any type C, there is a function

$$\mathrm{exfalso} : (\bot)C$$

Ex falso quodlibet is therefore justified:

$$\frac{a : \bot}{\mathrm{exfalso}(a) : C}$$

6. A Curry Paradox

For any inductively defined domain, A, let $\Gamma(A)$ be defined by the following clause:

$$\frac{f : (\Gamma(A))A}{\gamma(f) : \Gamma(A)}$$

Whenever f is a function from $\Gamma(A)$ to A, we may form $\gamma(f)$ of type $\Gamma(A)$. That this clause constitutes the definition of $\Gamma(A)$ means that every object of type $\Gamma(A)$ is equal to an object of the form $\gamma(f)$, where $f : (\Gamma(A))A$.

To define a function by induction on $\Gamma(A)$ it suffices to specify its value for every $\gamma(f)$. In particular, we may define a function H with domain $\Gamma(A)$ and codomain

the proposition, alias type, $\Gamma(A) \supset A$ by the following equation:

$$H(\gamma(f)) := \lambda(f)$$

Since H is defined for all objects of type $\Gamma(A)$, the following rule is justified:

$$\frac{a : \Gamma(A)}{H(a) : \Gamma(A) \supset A}$$

Using the function H as well as the functions λ and ap, associated with implication, we can show the existence of an object of type A. First we construct an object of type $(\Gamma(A))A$:

$$\frac{\dfrac{\dfrac{x : \Gamma(A)}{H(x) : \Gamma(A) \supset A} \qquad x : \Gamma(A)}{\mathrm{ap}(H(x), x) : A}}{[x].\mathrm{ap}(H(x), x) : (\Gamma(A))A}$$

The object constructed here will be called h:

$$h := [x].\mathrm{ap}(H(x), x)$$

Since h is of type $(\Gamma(A))A$, we may apply both λ and γ to it:

$$\lambda(h) : \Gamma(A) \supset A \qquad\qquad \gamma(h) : \Gamma(A)$$

Applying the function ap to these as arguments gives us an object of type A,

$$\mathrm{ap}(\lambda(h), \gamma(h)) : A$$

The proof is complete: we have constructed an object of type A.

Let us call the object of type A that we have constructed $\Omega(A)$,

$$\Omega(A) := \mathrm{ap}(\lambda(h), \gamma(h))$$

The construction of $\Omega(A)$ involves, besides functional abstraction, only the functions H, λ, and ap. No function associated with A is involved, hence we have constructed an object of type A without relying on any features of A other than its being an inductively defined type. In particular, we may infer that the type $\bot$ is inhabited, contrary to its definition.

What is more, we are not able to determine the value of $\Omega(A)$. Its calculation gets caught in a loop:

$$\begin{aligned}
& \Omega(A) \\
= \;& \mathrm{ap}(\lambda(h), \gamma(h)) \\
= \;& \mathrm{ap}\big(\lambda\big([x].\mathrm{ap}(H(x), x)\big), \gamma(h)\big) \\
= \;& \mathrm{ap}(H(\gamma(h)), \gamma(h)) \\
= \;& \mathrm{ap}(\lambda(h), \gamma(h)) \\
= \;& \Omega(A)
\end{aligned}$$

As a consequence, we cannot determine, for instance, whether $\Omega(\mathbb{N})$ is 0 or a successor, nor whether $\Omega(\mathbb{B})$ is $\mathtt{t}$ or $\mathtt{f}$.

Under the Curry–Howard correspondence, this reasoning translates into the logical Curry paradox, whose conclusion is that the arbitrary proposition A is true.

7. Solution of the paradox

An object of type $\Gamma(A)$ is, by definition, an object that is equal to an object of the form $\gamma(f)$, where $f : (\Gamma(A))A$. In order to determine whether a given object is of type $\Gamma(A)$, one must therefore have access to a function, f, from $\Gamma(A)$ to A. Since the function type $(\Gamma(A))A$ is defined by the application rule (App), knowing f to be such a function means knowing $f(a)$ to be of type A for an arbitrary object, a, of type $\Gamma(A)$. Knowing this, however, requires understanding what an arbitrary object of type $\Gamma(A)$ is, and that requires mastering the definition of $\Gamma(A)$. This is a vicious circularity.

The second clauses in the inductive definitions of $\mathbb{N}$ and $\mathbb{HC}$ also exhibit a form of circularity:

$$\frac{n : \mathbb{N}}{\mathrm{s}(n) : \mathbb{N}} \qquad \frac{f : (\mathbb{N})\mathbb{HC}}{\mathrm{set}(f) : \mathbb{HC}}$$

Here, just as in the putative definition of $\Gamma(A)$, the type to be defined is mentioned in the premiss. These clauses are, however, not viciously circular. In order to draw an inference according to the first of these clauses, it is not necessary to have complete knowledge of what the objects of type $\mathbb{N}$ are. The clause may be applied whenever we know that a given object is a natural number. In order to draw an inference according to the second clause, it is not necessary to have complete knowledge of what the objects of type $\mathbb{HC}$ are. The clause may be applied whenever we know that a given object, f, is a function from $\mathbb{N}$ to $\mathbb{HC}$, and that requires knowing merely that all the $f(n)$'s are hereditarily countable sets.

It is not the presence of the base cases, $0 : \mathbb{N}$ and $\emptyset : \mathbb{HC}$, that renders the circularity in the definitions of $\mathbb{N}$ and $\mathbb{HC}$ harmless. These clauses ensure that $\mathbb{N}$ and $\mathbb{HC}$ are non-empty. Even without a base case, we would be left with a correct inductive definition, although the type defined thereby would be empty. The putative definition of $\Gamma(A)$, by contrast, fails as a definition altogether.

Comparison with the definition of $\mathbb{HC}$ reveals where the putative definition of $\Gamma(A)$ goes wrong. The circularity in the definition of $\mathbb{HC}$ is harmless because the type being defined appears in the premiss as the codomain of a function type. In the putative definition of $\Gamma(A)$, by contrast, $\Gamma(A)$ appears in the premiss as the domain of a function type. (This is the negative occurrence of $\Gamma(A)$ mentioned in Section 1.) Quite generally, in order to know that f is a function from α to β one must have complete knowledge of what the objects of type α are, but one does not need complete knowledge of what the objects of type β are: it suffices to know that all the $f(a)$'s are β. Understanding a judgement of the form $f : (\alpha)\beta$ therefore requires complete knowledge of the objects of type α, but, in general, only partial knowledge of the objects of type β. A judgement of the form $f : (\alpha)\beta$ therefore cannot appear in the premiss of a clause meant to define α, though it can appear in the premiss of a clause meant to define β.

I propose to solve the type-theoretical Curry paradox by denying that $\Gamma(A)$ is a well-defined type. Because of the vicious circularity just noted, the putative definition of $\Gamma(A)$ fails to determine a type. Without recourse to $\Gamma(A)$, the reasoning in Section 6, showing the existence of an object of type A, does not go through, and the paradox disappears. This solution agrees with that advocated by Curry

himself, who took the paradox to show the need for restrictions on the means of defining propositions. However, whereas Curry merely pointed to the need for a more refined notion of proposition, we have given a precise and philosophically well-motivated restriction on the form of inductive definitions. Under the Curry–Howard correspondence, this becomes a restriction on the form of the definitions of propositions, just as Curry demanded.

Acknowledgements

I am grateful to Luca Tranchini for introducing me to the proof theory of paradoxes. While writing this paper I was financially supported by a *Lumina* fellowship, LQ300092101, from the Czech Academy of Sciences.

References

Aczel, P. (1977) 'An introduction to inductive definitions', in J. Barwise (ed.) *Handbook of Mathematical Logic*, 739–782. Amsterdam: North-Holland.

Church, A. (1940) 'A formulation of the simple theory of types', *Journal of Symbolic Logic*, 5: 56–68.

Coquand, T. (1986) 'An analysis of Girard's Paradox', in *Proceedings of the First Annual IEEE Symposium on Logic in Computer Science (LICS 1986)*, 227–236. Cambridge, MA: IEEE Computer Society Press.

Coquand, T. (1992) 'The paradox of trees in type theory', *BIT*, 32: 10–14.

Curry, H. B. and Feys, R. (1958) *Combinatory Logic*. Amsterdam: North-Holland.

Dybjer, P. (1994) 'Inductive families', *Formal Aspects of Computing*, 6: 440–465.

Dybjer, P. (2000) 'A general formulation of simultaneous inductive-recursive definitions in type theory', *Journal of Symbolic Logic*, 65: 525–549.

Dyckhoff, R. (2016) 'Some remarks on proof-theoretic semantics', in T. Piecha and P. Schroeder-Heister (eds.) *Advances in Proof-Theoretic Semantics*, 79–93. Cham: Springer.

Feferman, S. and Sieg, W. (1981) 'Inductive definitions and subsystems of analysis', in *Iterated Inductive Definitions and Subsystems of Analysis: Recent Proof-Theoretical Studies*, 16–77. Berlin: Springer.

Howard, W. A. (1980) 'The formulae-as-types notion of construction', in J. P. Seldin and J. R. Hindley (eds.) *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, 479–490. London: Academic Press.

Klev, A. (2019a) 'A comparison of type theory with set theory', in S. Centrone, D. Kant, and D. Sarikaya (eds.) *Reflections on the Foundations of Mathematics*, 271–292. Dordrecht: Springer.

Klev, A. (2019b) 'The name of the sinus function', in I. Sedlár and M. Blicha (eds.) *Logica Yearbook 2018*, 149–159. London: College Publications.

Martin-Löf, P. (1975). 'An intuitionistic theory of types: Predicative part', in H. E. Rose and J. C. Shepherdson (eds.) *Logic Colloquium '73*, 73–118. Amsterdam: North-Holland.

Martin-Löf, P. (1982) 'Constructive mathematics and computer programming' in J. L. Cohen *et al.* (eds.) *Logic, Methodology and Philosophy of Science VI, 1979*, 153–175. Amsterdam: North-Holland.

Martin-Löf, P. (1984) *Intuitionistic Type Theory*. Naples: Bibliopolis.

Martin-Löf, P. (1993) *Philosophical Aspects of Intuitionistic Type Theory*. Transcript of a lecture course given at Leiden University in the autumn semester of 1993. Available at `https://pml.flu.cas.cz/`.

Martin-Löf, P. (1998) 'An intuitionistic theory of types', in G. Sambin and J. Smith (eds.) *Twenty-five Years of Constructive Type Theory*, 127–172. Oxford: Clarendon Press.

Pezlar, I. (2021) 'A note on paradoxical propositions from an inferential point of view', in I. Sedlár and M. Blicha (eds.) *The Logica Yearbook 2020*, 183–199. London: College Publications.

Shapiro, L. and Beall, J. (2021) 'Curry's paradox', in E. N. Zalta (ed.) *The Stanford Encyclopedia of Philosophy*, winter 2021 edition. Stanford University: Metaphysics Research Lab.

Sundholm, B. G. (1994) 'Existence, proof and truth-making: A perspective on the intuitionistic conception of truth', *Topoi*, 13: 117–126.

Institute of Philosophy, Czech Academy of Sciences, Czechia