The Review of Symbolic Logic
Volume 19, Number 2, June 2026

PARADOXICAL CONNECTIVES: PROOF-THEORETIC SEMANTICS, RECURSION, AND FIXED-POINT OPERATORS

ALBERTO NAIBO
IHPST (UMR 8590)
and
YUTA TAKAHASHI
Aomori University

Abstract. This paper analyses some connectives introduced by Stephen Read, known as 'Bullet connectives'. From the proof-theoretic semantics perspective, these connectives satisfy the requirement of harmony; however, they allow for the looping derivation of a contradiction. Namely, when the operation of detour reduction is applied to such a derivation, it leads to a non-terminating phenomenon. By appealing to Prawtiz's notion of validity, we argue that such connectives cannot be considered as logical ones. Still, it is possible to assign a computational meaning to such connectives, as several useful computation properties are induced by their harmonious behaviour. In particular, we show that a fixed point operator can be defined by using the rules of a generalised version of the Bullet connectives. Such a generalisation is an instance of (non-terminating) recursive types. Finally, we show that type-free (i.e., untyped) lambda-calculus is interpretable in simply typed lambda-calculus extended with yet another variant of the Bullet connectives, and that the introduction rules of the Bullet connectives are interpretable by Nakano's modality in typed lambda-calculus. We conclude that such a computational interpretation of the Bullet connectives is possible only if we separate the notion of types from that of propositions, which are meaningful and assertible linguistic entities (in fact, it is larger). Therefore, the computational interpretation we propose cannot be considered an instance of the Curry–Howard correspondence between propositions and types, and between proofs and programs.

§1. Introduction.

1.1. Background. The main objective of *proof-theoretic semantics* is to explain the meaning of sentences in terms of proof-conditions rather than truth-conditions [59]. Within the framework of Gentzen-style natural deduction, proof-theoretic semantics rests on the idea that the proof-condition of a compound sentence is explained in terms of what counts as a *canonical* (*closed*) *derivation*.[1] Accordingly, we know the meaning of a compound sentence when we know what counts as its canonical derivation.

Received: August 15, 2025.
2020 *Mathematics Subject Classification*: Primary 03A05, Secondary 03B38.
Key words and phrases: proof-theoretic semantics, typed lambda calculus, Curry-Howard correspondence, paradoxes.

[1] Following a common practice in proof-theoretic semantics [22, pp. 35–36], we consider that a proof corresponds to a closed derivation (see p. 8).

© The Author(s), 2026. Published by Cambridge University Press on behalf of The Association for Symbolic Logic. This is an Open Access article, distributed under the terms of the Creative Commons Attribution licence (https://creativecommons.org/licenses/by/4.0/), which permits unrestricted re-use, distribution, and reproduction in any medium, provided the original work is properly cited.

doi:10.1017/S1755020326101099

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

A canonical derivation of a compound sentence $\mathsf{C}(A_1, \dots, A_n)$, with an n-ary principal connective C, is a valid (i.e., correct) derivation whose last rule is an introduction rule of C. This indicates that the introduction rules of the connective C play a privileged role in fixing the meaning of a compound sentence whose principal connective is C. Indeed, as Gentzen remarks, the introduction rules of a connective represent the 'definitions' of this connective, while its elimination rules are the 'consequences' of these definitions [64, p. 80].[2]

To assign a more precise sense to Gentzen's remark, Prawitz formulated the *inversion principle*, which maintains that by an application of an elimination rule, one simply 'restores what had already been established if the major premise of the application was inferred by an application of an introduction rule' [49, p. 33]. Consider, for instance, the introduction and elimination rules for the connective $\wedge$:

$$\frac{A \quad B}{A \wedge B}\ \wedge_I \qquad \frac{A \wedge B}{A}\ \wedge_E \quad \frac{A \wedge B}{B}\ \wedge_E.$$

If $\wedge_I$ determines the meaning of $\wedge$, then by stating $A \wedge B$, we should not be allowed to deduce anything *more* than what we can already obtain from the derivations of the premises of $\wedge_I$. Therefore, Prawitz's inversion principle can be rephrased as follows: the application of an elimination rule $\wedge_E$ is dispensable when its major premise is the conclusion of the introduction rule $\wedge_I$, as shown by the (operation of) proof reduction below (noted with $\triangleright$):

$$\dfrac{\dfrac{\begin{matrix}\mathcal{D}_1 & \mathcal{D}_2 \\ A & B\end{matrix}}{A \wedge B}\ \wedge_I}{A}\ \wedge_E \quad \triangleright \quad \begin{matrix}\mathcal{D}_1 \\ A.\end{matrix}$$

The left-hand derivation is what Prawitz calls a 'detour' [49, p. 34] and Dummett a 'local peak' [19, p. 248].

Several subsequent works [23, 46, 58] attempted to improve Prawitz's analysis of Gentzen's remark by coupling the inversion principle with the *recovery principle*, according to Schroeder-Heister's terminology [58]. The recovery principle, which is often said to be the 'converse' of the inversion principle, consists in asking that what we can deduce by stating $A \wedge B$ should not be anything *less* than what we can already obtain from the derivations of the premises of $\wedge_I$. In particular, this requires that we can recover the $\wedge_I$ rule from the assertion $A \wedge B$, as shown by the (operation of) proof expansion below (noted with $\triangleleft$, which is the opposite of $\triangleright$):

$$\begin{matrix}\mathcal{D} \\ A \wedge B\end{matrix} \quad \triangleleft \quad \dfrac{\dfrac{\begin{matrix}\mathcal{D} \\ A \wedge B\end{matrix}}{A}\ \wedge_E \quad \dfrac{\begin{matrix}\mathcal{D} \\ A \wedge B\end{matrix}}{B}\ \wedge_E}{A \wedge B}\ \wedge_I.$$

[2] Dummett calls the 'verificationist approach' one that gives priority to introduction rules over elimination ones. However, one can also decide to start by giving priority to elimination rules. This is what Dummett calls the 'pragmatist approach' [19]. The BHK-interpretation of intuitionistic connectives somehow dictates the priority of introduction rules. The clauses of the BHK-interpretation can indeed be seen as establishing the canonicity conditions in terms of introduction rules [49, p. 32].

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

The connectives whose inference rules satisfy both the inversion and the recovery principle are said to be *harmonious* [23].[3] Asking for both the inversion and the recovery principle is a way of demanding that the meaning of a connective C is entirely determined by its introduction rules: to fix the meaning of C, and thus of a compound sentence whose principal connective is C, all we need are the inference rules for C; there is no need to look for any context (external to the rules of C) in which to fix the reference of this connective. This is why proof-theoretic semantics is considered a non-referentialist semantics [40], and harmony is intended as a condition for *logicality*, though it is considered only a necessary condition. §2 will clarify this point.

While harmony has been considered so far from a logical perspective, it can also be discussed from a computational point of view. In particular, it is possible to elaborate a computational notion of harmony using the *Curry–Howard correspondence*. This correspondence concerns the possibility of establishing a formal link between proof (or deductive) systems in logic and functional languages in computer programming: formulas (or propositions) are interpreted as types, proofs as programs, and proof transformations (reductions or expansions) as computational steps (see, e.g., [5, 53, 63]). For instance, the correspondence between the implicative fragment of natural deduction and simply typed λ-calculus, which can be seen as an abstract programming language, has been discussed in the literature.[4] In typed λ-calculus, there are typing rules which associate types with λ-terms (i.e., programs): a λ-term t, if it is typable, is classified into a type A by means of typing rules. We say that the term t belongs to the type A, or t is of type A, and write $t : A$. In addition, each type is accompanied by some computation rules specifying how to execute the λ-terms which belong to this type. In this sense, a type can be considered as a *classification* of a program, as it indicates the behaviour of a program and how the program acts when it interacts with other programs.[5] Hence, according to the Curry–Howard correspondence, one can consider a formula in a natural deduction system as a classification of programs, an inference rule as a typing rule, and a reduction rule for proofs as a computation rule for λ-terms. In particular, the inversion principle and the recovery principle, which

[3] In this paper, we consider the notion of harmony formulated in terms of the standard presentation of the elimination rules. Several authors, such as [23, 55], also discuss its variant formulated in terms of *general elimination rules*.

[4] In this paper, by 'simply typed λ-calculus' we mean the fragment having the arrow type as its only type constructor. Simply typed λ-calculus is not a concrete programming language but an idealised one obtained by abstraction from the particular syntax of functional programming languages.

[5] As stated by Cardelli and Wegner: 'Types arise informally in any domain to categorize objects according to their usage and behavior. The classification of objects in terms of the purposes for which they are used eventually results in a more or less well-defined type system. [...] The objective of a language for talking about types is to allow the programmer to name those types that correspond to interesting kinds of behavior' [11, pp. 473, 490]. As noticed by Turner, the classification allowed by types plays different roles. The first is to guide the programming practice: 'Objects are classified in order to provide structure to the programmer's universe and algorithm development has to proceed in harmony with this structure' [72, p. 102]. However, the classification of objects, and in particular programs, into types also has another role—that of allowing one to 'formulat[e] and prov[e] properties about them' [72, p. 103]. In other words, types allow for a formal classification of programs, and typing systems enable the possibility of formal reasoning about them.

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

are the ingredients of harmony, can be considered as the two computation rules called *β-reduction* and *η-expansion*, respectively.[6]

1.2. Aim and outline. Inspired by the Curry–Howard correspondence, in this paper we mainly focus on harmony from a computational point of view. However, our approach differs from a full-fledged interpretation of the Curry–Howard correspondence because, while the Curry–Howard correspondence states that a proposition corresponds to a type, and vice versa, our interpretation does not always require a type to correspond to a proposition.[7] This means that we emphasise the conception of type as a classification of a program rather than the proposition-as-type conception. In this respect, the computational systems we present in §3 have to be considered as typing systems which do not fully enjoy a Curry–Howard interpretation.

More specifically, we examine various computational properties of the 'Bullet connectives' previously introduced by Read [54, 55]. As he showed, these connectives are harmonious but present some problematic features. When they are used with other connectives, they allow for the derivation of a contradiction, and when detour reductions are applied to this derivation, such reductions are non-terminating. For this reason, the Bullet connectives are considered paradoxical connectives. By analysing the non-terminating phenomena, Roy Dyckhoff pointed out another problem, namely, a circularity in the explanation of the meaning of the Bullet connectives in terms of proof-conditions. In §2, we analyse Read's and Dyckhoff's arguments relying on Prawitz's notion of validity. We argue that, although harmonious, the Bullet connectives cannot be considered logical from the perspective of proof-theoretic semantics. The existence of harmonious connectives that are not logical suggests that harmony can be considered only a necessary condition for logicality, but not a sufficient one.

In §3, we consider harmony from a computational angle, arguing that the Bullet connectives are computationally meaningful, as they have several useful computational properties induced by their harmonious behaviour. To show this, we first reformulate the inference and the reduction rules of the Bullet connectives within a system inspired by the Curry–Howard correspondence. In particular, we consider the inference rules of this system as typing rules, and its proof-reduction rules as term-reduction rules (or computation rules). However, Read's and Dyckhoff's arguments against the logicality of the Bullet connectives show that the types defined in this system cannot be considered propositions. This is why our typing system ultimately deviates from enjoying a genuine Curry–Howard correspondence.

Once we have set our typing systems, we show that the behaviour of the general fixed-point operator fix, which has been discussed in the context of Martin-Löf type theory (e.g., [35, 43]), can be simulated by means of this computational reformulation of Read's *generalised Bullet connective* $\bullet^A$ [55]. The connective $\bullet^A$ is also shown to be an instance

[6] Strictly speaking, the inversion and recovery principles correspond to β-reduction and η-expansion in *typed* λ-calculus: the Curry–Howard correspondence applies indeed to (fragments of) typed λ-calculus and not to the (full) untyped λ-calculus. In §3.3, we will analyse untyped λ-calculus from the point of view of typed λ-calculus extended with a connective that does not behave well from the logical point of view.

[7] By a 'proposition' we do not mean a *mere proposition* (i.e., an *h-proposition*) in the sense of homotopy type theory [73], which is a type P such that for any $x, y : P$, the identity type $x = y$ is inhabited. We mean a meaningful and assertible linguistic entity, for which we can give evidence.

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

of the non-normalising recursive types (which are presented, for instance, in [24]). This allows us to compare $\bullet^A$ with the recursive types in terms of type isomorphisms [10, 16]. In particular, while $\bullet^A$ can only be proved to be isomorphic with $\bullet^A \to A$, we show that a recursive type Ref can be defined, allowing a stronger isomorphism—namely, the one between Ref and Ref $\to$ Ref. Next, we prove that type-free (i.e., untyped) λ-calculus is interpretable in simply typed λ-calculus extended with yet another variant of Bullet connectives denoted by $\bullet^*$ below. Finally, we discuss the computational meaning of the Bullet connectives in terms of Nakano's modality [41] in typed λ-calculus: one can derive the *type* of Nakano's fixed-point operator in simply typed λ-calculus with the $\bullet^A$-introduction rule only.[8] We show then that one can also interpret the $\bullet^A$-introduction rule in Nakano's type system. This provides the $\bullet^A$-introduction rule with a computational meaning in his type system.

Our main contributions in this paper can be summarised as follows:

- We provide a precise analysis of the circularity issue regarding the definition of the Bullet connectives raised by Dyckhoff by appealing to Prawitz's notion of validity.
- We show that the Bullet connectives are computationally meaningful as:
  - the general fixed-point operator fix can be defined by means of the computational reformulation of Read's generalised Bullet connective $\bullet^A$;
  - the connective $\bullet^A$ is an instance of recursive types and is isomorphic only with $\bullet^A \to A$, while it is possible to define a recursive type Ref such that Ref and Ref $\to$ Ref are isomorphic;
  - type-free λ-calculus is interpretable in simply typed λ-calculus extended with yet another variant of the Bullet connectives;
  - the type of Nakano's fixed-point operator is derivable in simply typed λ-calculus with the $\bullet^A$-introduction rule only, whereas the $\bullet^A$-introduction rule is interpretable in Nakano's type system.

§2. Read's Bullet connective. In this section, we first recall two formulations of Read's Bullet connective, $\bullet$, both yielding the derivation of a contradiction as well as the non-termination of detour reductions (§2.1). We then discuss Dyckhoff's remark about the circularity of $\bullet$, and we make this remark more precise by appealing to Prawitz's notion of validity (§2.2).

2.1. Two formulations of the Bullet connective. We adopt the following notational convention concerning derivational systems (in particular, natural deduction systems): when we write some derivations where the discharge of open assumptions and the composition of derivations are at play, as in this case

$$\dfrac{\dfrac{\overset{k}{[A]}\ \mathcal{D}_1}{B}}{\dfrac{A \to B}{B}}\to_{I\,k}\quad \dfrac{\mathcal{D}_2}{A}\to_E \qquad \triangleright \qquad \dfrac{\mathcal{D}_2}{A}\ \dfrac{\mathcal{D}_1}{B}$$

[8] We thank the anonymous referee for pointing this out and observing that the $\bullet^A$-introduction rule can be considered a variant of Löb's axiom, which corresponds to the type of Nakano's fixed-point operator.

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

we denote with the index k the fact that the application of $\rightarrow_I$ discharges n occurrences of an open assumption A ($n \geq 0$) annotated with k. In addition, $\mathcal{D}_1$ denotes a derivation from an open assumption A to the conclusion (or the end formula) B, possibly containing some other open assumptions. Similarly, $\mathcal{D}_2$ denotes a derivation of A, possibly containing some open assumptions. The right-hand derivation is then obtained by substituting n copies of $\mathcal{D}_2$ for n occurrences of an open assumption A in $\mathcal{D}_1$.

Let us now recall two formulations of the introduction and elimination rules of $\bullet$, which is a 0-ary connective, thus corresponding to an atomic formula. The first formulation is the pair $\bullet 1_I$ and $\bullet 1_E$ below (see [55, sec. 7]); if we then define $\neg A$ as $A \rightarrow \bot$, a second formulation of the rules of $\bullet$ can be given as the pair $\bullet 2_I$ and $\bullet 2_E$ (see [54, sec. 2.8])[9]:

$$\dfrac{\begin{array}{c}[\bullet]^n \\ \vdots \\ \bot\end{array}}{\bullet}\,\bullet 1_{I\,n} \qquad \dfrac{\bullet \quad \bullet}{\bot}\,\bullet 1_E \qquad\qquad \dfrac{\neg\bullet}{\bullet}\,\bullet 2_I \qquad \dfrac{\bullet}{\neg\bullet}\,\bullet 2_E.$$

One can easily see that these two formulations are equivalent, namely, each pair of introduction and elimination rules is derivable from the other pair by means of the inference rules of $\rightarrow$. The second formulation, however, gives a very intuitive idea of the behaviour of $\bullet$: it corresponds to a formula which is logically equivalent to its negation. This is why Read interprets $\bullet$ as a 'proof-conditional Liar sentence' [54, p. 142] or, alternatively, as corresponding to Russell's paradox formula $r \in r$, where $r = \{x \mid x \notin x\}$ ([55, p. 574]; cf. also Tranchini [67, sec. 2] about this second interpretation). For this reason, one can take $\bullet$ to be 'a paradoxical connective' (we will better clarify this point later).

Each of these two formulations satisfies the inversion principle, as already shown by Read in [54, 55] (in his terms, each satisfies harmony); also, each of them satisfies the recovery principle:

$$\mathcal{D}\;\bullet \;\lhd\; \dfrac{\mathcal{D}\;\bullet \quad [\bullet]^n}{\bot}\,\bullet 1_E\,\bullet 1_{I\,n} \qquad\qquad \mathcal{D}\;\bullet \;\lhd\; \dfrac{\dfrac{\mathcal{D}}{\bullet}\,\bullet 2_E}{\bullet}\,\bullet 2_I.$$

The fact that $\bullet$ satisfies both the inversion and the recovery principles makes it comparable to the standard connectives (like conjunction, disjunction, and implication), which also satisfy these two principles. Up to this point, $\bullet$ meets all the relevant conditions to be a 'good' candidate as a logical connective.[10]

[9] In fact, Read in [54] defined the elimination rule of $\bullet$ as an inference rule acting on the antecedent of a sequent, as he adopts a sequent-style presentation of natural deduction. Here, we adapt this elimination rule to usual natural deduction format.

[10] One might find it problematic that some other connectives appear in the introduction rules for $\bullet$: the 0-ary connective $\bot$ appears in the first formulation, while $\bot$ and $\rightarrow$ appear in the second formulation. This makes the meaning of $\bullet$ dependent on other connectives. To avoid any possible circularity, an order in the explanation of the meanings of these connectives must be imposed. This consideration is closely related to the condition of *separability* in proof-theoretic semantics (as discussed, for instance, in [12, chap. 1], where $\bullet$ is rejected as non-meaningful exactly because it does not respect some form of separability). Here, we

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

However, Read showed that each of the two formulations yields a contradiction:

$$\dfrac{\dfrac{\overset{1}{[\bullet]}\quad\overset{1}{[\bullet]}}{\dfrac{\bot}{\bullet}\,\bullet 1_{I\,1}}\,\bullet 1_E \qquad \dfrac{\overset{2}{[\bullet]}\quad\overset{2}{[\bullet]}}{\dfrac{\bot}{\bullet}\,\bullet 1_{I\,2}}\,\bullet 1_E}{\bot}\bullet 1_E$$

$$\dfrac{\dfrac{\overset{1}{[\bullet]}}{\neg\bullet}\,\bullet 2_E \quad [\bullet]}{\dfrac{\bot}{\neg\bullet}\,\to_{I\,1}}\,\to_E \qquad \dfrac{\dfrac{\overset{2}{[\bullet]}}{\neg\bullet}\,\bullet 2_E \quad \overset{2}{[\bullet]}}{\dfrac{\bot}{\neg\bullet}\,\to_{I\,2}}\,\to_E}{\dfrac{\bot}{\bullet}\,\bullet 2_I}\,\to_E .$$

Read also showed in [55] that the reduction of the first contradictory derivation is non-terminating, and it is not hard to see the non-termination of the reduction of the second as well. The absence of contradiction has been traditionally considered (since Aristotle) one of the characteristic features of any logical and meaningful discourse. The fact that • brings about a contradiction seems then to make a point against its logicality and meaningfulness, although it satisfies the harmony condition.[11]

2.2. The circularity of the Bullet connective. By analysing the problems of contradiction and non-termination we just exposed, Dyckhoff identifies another issue concerning • from the point of view of proof-theoretic semantics [20, p. 82]. He considers the first formulation of the inference rules for •, and following the meaning-explanation of propositions as given by [19, 34, 36], he claims that 'we understand a proposition when we understand what it means to have a *canonical* proof of it, i.e. what forms a canonical proof can take' [20, p. 82]. By a 'canonical proof', Dyckhoff means a derivation whose last rule is an introduction rule. According to him, the first formulation of the introduction rule for • attempts to explain what a canonical proof for • is: this rule says that a canonical proof for • is a proof obtained by applying $\bullet 1_I$-rule to a derivation from • to $\bot$. Dyckhoff treats the subderivation $\mathcal{D}$ as a method for transforming *arbitrary proofs of* • into proofs of $\bot$, and he finds a circularity in the alleged explanation. That is, the above explanation of canonical proofs for • presupposes the notion of proof of •, in particular the notion of canonical proof for •.

What stands behind Dyckhoff's remark is in fact a well-founded and hierarchical view of proofs. A typical instance of such a view is given by the *Brouwer–Heyting–Kolmogorov interpretation* (the BHK-interpretation, for short; see [70, pp. 9–10]). The BHK-interpretation is an informal explanation of the meaning of intuitionistic logic. In particular, proofs in this interpretation stand for any kind of objects provided by (idealised) mathematicians in order to give evidence to their claims. The domain of such proofs is constructed bottom-up, and this construction proceeds along a certain hierarchical structure. As explained by van Atten, this view essentially follows Brouwer's intuitionism:

> [...] in our mathematical acts we first construct certain basic objects, [...] and then to those apply the same mathematical acts to construct further objects. This activity thus has an iterative structure, and

will not discuss this condition, as it becomes harmless (or at least, hardly applicable) to the case of the generalised Bullet connective introduced in §3, where no explicit appeal to other connectives is made in its introduction and elimination rules.

[11] By following Tennant's proof-theoretic analysis of paradoxes [65], the fact that the two derivations of contradiction generate looping detours reductions corroborates the idea of looking at • as a paradoxical connective. We will address the question of the paradoxical character of Bullet connectives at p. 23 by adding another argument confirming this view.

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

> induces a linear order on the constructions that the Creating Subject has effected. [...] As the linear ordering of the Creating Subject's constructions is not only the order in which it becomes aware of the objects, but indeed the order in which they come into being, this order is a mathematical fact. [3, p. 1598]

If we read Dyckhoff's remark about the circularity of $\bullet$ in the light of this passage, we can interpret it as suggesting that a well-founded and hierarchical domain of proofs cannot be constructed—even by some idealised mathematician, like the Creative Subject—in the case of $\bullet$.

More precisely, we spell out Dyckhoff's idea by showing that if we try to define in a bottom-up way a domain of proofs by using the $\bullet$-rules, we eventually end up with a circular definition. Since the BHK-interpretation is an informal one, we replace it by appealing to a formal notion, that of Prawitz's *validity*. This notion indeed includes a well-founded and hierarchical perspective of proofs, as the one considered above, although proofs are taken here to be closed derivations in the natural deduction format (see the definition below). The formulation of validity which we adopt here is a simplified version of the one given in [50, 56].[12]

A derivation $\mathcal{D}$ is *closed* if it has no open assumption; otherwise, $\mathcal{D}$ is said to be *open*. An *atomic system* is a pair composed of an arbitrary set $\mathcal{P}$ of atomic formulas including $\bot$ and an arbitrary set $\mathcal{R}$ of inference rules of the form

$$\frac{A_1 \quad \cdots \quad A_n}{A}$$

such that

(i) $n \geq 0$,
(ii) $\{A, A_1, \ldots, A_n\} \subseteq \mathcal{P}$, and
(iii) no closed derivation of $\bot$ can be obtained by using the inference rules of $\mathcal{R}$.

When $n = 0$ holds, the inference rule is an axiom which states that A holds. In the rest of this subsection, we consider formulas to be given relative to an atomic system $\mathcal{S}$: formulas are constructed by a given set of logical connectives from atomic formulas in $\mathcal{S}$. For an atomic system $\mathcal{S}$, a *derivation in* $\mathcal{S}$ is a derivation which contains inference rules in $\mathcal{S}$ only. Closed derivations in an atomic system $\mathcal{S}$ are the basic proofs on which a domain of proofs is constructed in a bottom-up way. For a finite set $\{A_1, \ldots, A_n\}$ of formulas, we define its *complexity* as the number of all occurrences of connectives in this set. The complexity of a formula A is defined as the complexity of $\{A\}$.

In order to introduce Prawitz's notion of validity for $\bullet$, we first formulate the notion of validity for derivations which contain only the rules of a given atomic system $\mathcal{S}$ and the rules for $\rightarrow$.

[12] Specifically, $\mathcal{S}$-valid derivations in our sense correspond to $\mathcal{S}$-valid derivations in the sense of the *$\mathcal{S}$-validity (1)* defined in [56, p. 543], with the only exception that we omit the condition concerning the extension of atomic systems (see below). This simplification is harmless, since the notion of validity we are interested in only concerns the derivations defined in a fixed logical system, while we leave aside the discussion of the notion of validity for arbitrary *derivation structures* (as defined in [56, sec. 5]).

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

DEFINITION 2.1 (Validity of derivations with atomic systems). *Let $\mathcal{S}$ be an atomic system. The set of $\mathcal{S}$-valid derivations is defined by induction on the sum of the complexities of the open assumptions $\{A_1, \ldots, A_n\}$ and the end formula B*:

1. *A closed derivation of an atomic formula is $\mathcal{S}$-valid if it reduces to a derivation in $\mathcal{S}$.*
2. *A closed derivation of $A \to B$ is $\mathcal{S}$-valid if it reduces to a derivation of the form*

$$\dfrac{\begin{array}{c}\overset{n}{[A]}\\ \mathcal{D}_1\\ B\end{array}}{A \to B}\ {\to I}_n$$

and for any closed $\mathcal{S}$-valid derivation $\begin{array}{c}\mathcal{D}_2\\ A\end{array}$, the derivation

$$\begin{array}{c}\mathcal{D}_2\\ A\\ \mathcal{D}_1\\ B\end{array}$$

is $\mathcal{S}$-valid.

3. *An open derivation of the form*

$$\begin{array}{ccc} A_1 & \cdots & A_n\\ & \mathcal{D} & \\ & B & \end{array}$$

is $\mathcal{S}$-valid if for any closed $\mathcal{S}$-valid derivation $\begin{array}{c}\mathcal{D}_i\\ A_i\end{array}$ with $1 \leq i \leq n$, the derivation

$$\begin{array}{ccc} \mathcal{D}_1 & & \mathcal{D}_n\\ A_1 & \cdots & A_n\\ & \mathcal{D} & \\ & B & \end{array}$$

is $\mathcal{S}$-valid.

Using the notion of $\mathcal{S}$-validity, we can now explain better the circularity of the $\bullet$-rules. The inductive definition of validity breaks once we add the clause for defining the validity of a closed derivation of $\bullet$: by adding this clause, we get a circular definition. Consider the first formulation of the $\bullet$-rules, and suppose that we try to define the $\mathcal{S}$-validity of a derivation by induction on the sum of the complexities of its open assumptions and the end formula. The similarity between ${\bullet}1_I$ and ${\to}_I$ suggests considering the following clause:

(B) A closed derivation of $\bullet$ is $\mathcal{S}$-valid if it reduces to a derivation of the form

$$\dfrac{\begin{array}{c}\overset{n}{[\bullet]}\\ \mathcal{D}_1\end{array}}{\bullet}\ {\bullet}1_{I\ n}$$

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

and for any closed $\mathcal{S}$-valid derivation $\begin{array}{c}\mathcal{D}_2\\ \bullet\end{array}$, the derivation

$$\begin{array}{c}\mathcal{D}_2\\ \bullet\\ \mathcal{D}_1\\ \bot\end{array}$$

is $\mathcal{S}$-valid.

Here the $\mathcal{S}$-validity of a closed derivation of $\bullet$ depends on the validity of arbitrary closed $\mathcal{S}$-valid derivations having the same conclusion, i.e., $\begin{array}{c}\mathcal{D}_2\\ \bullet\end{array}$. This implies that the inductive clause (B) does not decrease the joint complexity of the open assumptions and the end formula, since the derivations considered are all closed, valid, and have the same complexity as the conclusion. Therefore, the inductive definition breaks down, and the clause leads to a circularity, as point out by Dyckhoff. The second formulation of the $\bullet$-rules breaks the inductive definition as well.

To sum up, the introduction and elimination rules for $\bullet$ satisfy both the inversion principle and the recovery principle (whenever we adopt the first formulation or the second formulation). However, these rules yield a contradiction, and derivations of such a contradiction do not admit a normal form. Moreover, as Dyckhoff remarks, when these rules are analysed from the point of proof-theoretic semantics, the $\bullet$-rules present a form of circularity, which can be made explicit by appealing to Prawtiz's notion of validity, as we have just shown.

Considering $\bullet$ through Prawitz's validity, we can specify that deriving a contradiction is highly undesirable and makes it problematic to consider $\bullet$ to be a (0-ary) logical connective. Although $\bullet$ is harmonious, and thus satisfies a necessary condition to be a logical connective, the derivation of a contradiction makes it possible to produce non-valid derivations in the sense of Prawitz. The derivations of $\bot$ that we have seen on p. 7 are closed and non-canonical; hence, by applying the rule of *ex falso quodlibet*, it is possible to obtain closed and non-canonical derivations of any formula. In particular, we can obtain a closed derivation for any atomic formula. Among these formulas, there is also $\bullet$ (since a 0-ary connective corresponds to an atomic proposition), and such a derivation cannot be reduced to a derivation in any atomic system. We are considering logical validity and no atomic rules are given. Moreover, the rules of $\bullet$ (in both formulations) cannot themselves belong to any atomic system: in the first formulation, the $\bullet$-introduction rule can discharge finitely many assumptions of the form $\bullet$, while in the second formulation the connective $\neg$ occurs in the $\bullet$-rules, so that in both cases we are not respecting the format of atomic rules given at p. 8.[13] Thus, this situation makes the first clause of Prawitz's validity false. In fact, the situation would be problematic also if we consider arbitrary formulas instead of atomic ones. The application of the *ex falso quodlibet* to the two closed derivations of $\bot$ considered at p. 7 makes it possible to obtain a closed derivation of $A \to B$, for any A and B. This

[13] One might wonder whether the first formulation of introduction and elimination rules for $\bullet$ can be treated as an atomic system when higher-level atomic systems in the sense of [47] are admitted. However, the condition (iii) in the definition of atomic systems prevents the first formulation from being a higher-level atomic system: if this formulation of $\bullet$-introduction and elimination rules is included in higher-level atomic rules, there would be a closed derivation of $\bot$ consisting only of atomic rules, as seen above.

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

is a derivation that cannot be reduced to a derivation ending with the $\rightarrow$-introduction rule, so that in this case it is the second clause of Prawitz's validity definition which would become false.

It could be objected that by dropping the *ex falso quodlibet* rule, the $\bullet$-rules do not yield non-valid derivations. However, the circularity of the validity clauses for $\bullet$ would still be problematic, as it prevents an inductive construction of the domain of proofs. The issue is whether such a conception of the domain of proofs is essential. Considering that this conception is motivated by an intuitionistic point of view, we might want to abandon it. However, if we do so, we will not only abandon one of the sources of inspiration for proof-theoretic semantics, but also one of the characteristic features shaping it. The adoption of proof-theoretic semantics is usually linked with intuitionistic logic.[14] Hence, to avoid the hierarchical and well-founded view of proofs that we mentioned, one should be able to show that proof-theoretic semantics can be defined without necessarily having to adopt an intuitionistic conception. In other words, one should show that it is possible to have a proof-theoretic semantics that does not validate intuitionistic logic. This is not the approach we want to pursue. Note that, originally, the very idea of proof-theoretic semantics rests on the BHK-interpretation (cf. [59]). We consider that such a hierarchical and well-founded notion of proofs is perfectly legitimate, and that $\bullet$ is not enough to question it.

The lesson we learn from the proof-theoretic analysis of $\bullet$ is actually another one, namely, that harmony, being only a necessary but not sufficient condition for logicality, is too weak as a requirement. Moreover, it is not even a sufficient condition for meaningfulness. It is only by satisfying Prawitz's validity that we can say that a certain connective is really meaningful from a proof-theoretic semantics point of view. More precisely, if the rules of a certain connective allow one to define a notion of proof that satisfies validity, we can then say that this connective is meaningful. In particular, this connective allows one to define propositions, that is, meaningful linguistic entities that are assertible, and for which we can give evidence.[15] The fact that the $\bullet$-rules do not allow one to define a notion of proof satisfying validity means that $\bullet$ does not allow one to form propositions.

The notion of validity is defined inductively; one of the induction parameters is the complexity of the conclusion of a proof. Now, according to what we just said, the conclusion of a proof is not simply a formula, but a proposition. This implies that the notion of proposition must be also inductively defined, in the sense that it must be well-founded. This is consistent with the notion of proof discussed above: proofs involve propositions, and both proofs and propositions are organised in a hierarchical and well-founded way.

We will revisit the notion of proposition in §5, comparing it with the notion of type, which we will discuss in the next section.

---

[14] See [59, secs. 1.2 and 2.8]. In this paper, we adopt this standard view, which holds that proof-theoretic semantics essentially justifies intuitionistic logic (see [19, 52]). Some authors, such as Read [54], argue that classical logic can also be justified in terms of a proof-theoretic semantics using, for example, multi-conclusion natural deduction systems. However, since intuitionistic logic is a subsystem of classical logic, it remains justified from such a perspective. More generally, these authors attempt to offer a proof-theoretic semantics justification of logical systems that are extensions of intuitionistic logic. Hence, intuitionistic logic still remains at the core of the proof-theoretic semantics project.

[15] The idea is that a correct assertion is supported by a proof.

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

$$A, B, C ::= P \mid \bot \mid A \to B \mid \bullet \qquad \text{types}$$
$$s, t, u ::= x \mid \lambda x.t \mid \mathsf{app}(t, s) \mid \mathsf{elim}_\bot(t) \mid \hat{x}.t \mid @(t, s) \qquad \text{terms}$$

Figure 1. Types and terms of $\lambda_1^{\to\bullet}$.

§3. The computational meaning of the Bullet connective. Read's Bullet connective $\bullet$ is problematic from the logical point of view. In this section, we argue that this connective is nevertheless *computationally* meaningful, as some variants of it have several computational properties. We start by reformulating the rules for $\bullet$, as well as its generalisation given in [55], within a framework inspired by the Curry–Howard correspondence (§3.1). Next, we show that the behaviour of the fixed-point operator fix, which has been discussed in the context of Martin-Löf type theory, can be simulated with this generalisation of the Bullet connective. Moreover, we note that the generalised Bullet connective is an instance of non-normalising recursive types (§3.2). We prove then that type-free λ-calculus is interpretable in simply typed λ-calculus with yet another variant $\bullet^*$ of the Bullet connective, exploiting the *type isomorphism* between $\bullet^*$ and $\bullet^* \to \bullet^*$ (§3.3). Finally, we discuss the computational meaning of Bullet connectives in terms of Nakano's modality [41] in typed λ-calculus (§3.4). One can derive the *type* of Nakano's fixed-point operator in simply typed λ-calculus with the $\bullet^A$-introduction rule only, and we show that one can also interpret the $\bullet^A$-introduction rule in Nakano's type system.

3.1. The Bullet connective in the light of a computational framework. We extend simply typed λ-calculus $\lambda^\to$ (see, e.g., [63]) by adding the Bullet connective to this calculus. Given infinitely many atomic types $P_0, P_1, \ldots$ and variables $x_1, x_2, \ldots$, types and (untyped) terms of the extended calculus are defined as in Figure 1. We denote this calculus by $\lambda_1^{\to\bullet}$, since the constructor $\hat{x}.t$ and the destructor $@(t, s)$ correspond to the first version of the $\bullet$-introduction and $\bullet$-elimination rules, respectively.[16] In what follows, we abbreviate $\mathsf{app}(t, s)$ as $t\ s$, and write $@(t, s)$ using the infix notation $t@s$. Note that, in the term $\hat{x}.t$, the variable x is bound as in the λ-abstraction term $\lambda x.t$. The expression $t[s/x]$ denotes the term obtained by substituting the term s for every free occurrence of the variable x in the term t. We often abbreviate $t[s/x]$ as $t[s]$ when it is obvious which variable is replaced. These notations can be extended to simultaneous substitution $t[s_1/x_1, \ldots, s_n/x_n]$ straightforwardly (which can thus be abbreviated as $t[s_1, \ldots, s_n]$).

A pair (x, A) composed of a variable and a type is called a *declaration*, and it is usually written as $x : A$. If the variables present in a set of declarations are pairwise distinct, we call this set a *context*, and a context is denoted by a capital Greek letter, such as Γ and Δ. Usual notations for contexts are adopted: for instance, a singleton $\{x : A\}$ is abbreviated as $x : A$, and we denote the union of two contexts Γ and Δ by Γ, Δ.

A typing judgement is an expression of the form $\Gamma \vdash t : A$, meaning that the term t is of type A with respect to the context Γ.[17] The typing rules of $\lambda_1^{\to\bullet}$ operate on this

[16] We borrow the technical terms 'constructor' and 'destructor' from [28, 61], respectively. In [28], a destructor is called a *selector*.

[17] In this paper, we do not consider Church-style typing, but only Curry-style typing, as we assign a type to a pure (or untyped) term (and vice versa—we decorate natural deduction

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A}\ \mathrm{Var} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ {\to_I} \qquad \frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash s : A}{\Gamma \vdash t\ s : B}\ {\to_E}$$

$$\frac{\Gamma \vdash t : \bot}{\Gamma \vdash \mathsf{elim}_\bot(t) : A}\ {\bot_E} \qquad \frac{\Gamma, x : \bullet \vdash t : \bot}{\Gamma \vdash \hat{x}.t : \bullet}\ {\bullet 1_I} \qquad \frac{\Gamma \vdash t : \bullet \quad \Gamma \vdash s : \bullet}{\Gamma \vdash t@s : \bot}\ {\bullet 1_E}$$

Figure 2. $\lambda_1^{\to\bullet}$-typing rules.

$$\frac{\Gamma \vdash t : \bullet \to \bot}{\Gamma \vdash \mathsf{fold}(t) : \bullet}\ {\bullet 2_I} \qquad \frac{\Gamma \vdash t : \bullet}{\Gamma \vdash \mathsf{unfold}(t) : \bullet \to \bot}\ {\bullet 2_E}$$

Figure 3. $\lambda_2^{\to\bullet}$-typing rules.

kind of judgement and are formulated in a natural deduction system, presented in a sequent-style format,[18] as in Figure 2. This allows for an alternative reading of the role of the terms, namely, that of keeping track of the natural deduction rules applied to form it, so that a judgement of the $\Gamma \vdash t : A$ can also be read as expressing the fact that t is the code of a derivation of A from the premises Γ. If we adopt the usual natural deduction format (instead of the sequent-style format), as well as the same notational conventions adopted in §2.1, the typing rules for $\bullet$ can be presented in the following way, where the introduction and elimination rules for $\bullet$ are decorated with appropriate terms:

$$\frac{\begin{array}{c}[x \overset{n}{:} \bullet] \\ \vdots \\ t : \bot\end{array}}{\hat{x}.t : \bullet}\ {\bullet 1_{I\,n}} \qquad \frac{t : \bullet \quad s : \bullet}{t@s : \bot}\ {\bullet 1_E}.$$

We can also present the second version of the $\bullet$-rules as typing rules. The syntax of the system $\lambda_2^{\to\bullet}$, which is an extension of $\lambda^{\to}$ with this second version of rules for $\bullet$, is obtained by replacing the constructor $\hat{x}.t$ and the destructor $t@s$ with $\mathsf{fold}(t)$ and $\mathsf{unfold}(t)$, respectively. The typing rules of $\lambda_2^{\to\bullet}$ are those of $\lambda^{\to}$ together with the ones in Figure 3. The reason for presenting the constructor of the $\bullet$-introduction rule (resp. the destructor of the $\bullet$-elimination rule) as fold (resp. as unfold) in the second version will be explained in §3.2.

We define the *reduction relation* $\rhd$ on the terms of $\lambda_1^{\to\bullet}$ and $\lambda_2^{\to\bullet}$ as in Figure 4. In addition to the standard rules (β), (η), (ξ), (ConL), and (ConR) coming from λ-calculus, the set of $\lambda_1^{\to\bullet}$-reduction rules contains the rules $(\beta_{\bullet 1})$, $(\eta_{\bullet 1})$, $(\xi_{\bullet 1})$, $(\mathrm{ConL}_{\bullet 1})$, and $(\mathrm{ConR}_{\bullet 1})$, while the set of $\lambda_2^{\to\bullet}$-reduction rules contains $(\beta_{\bullet 2})$, $(\eta_{\bullet 2})$, $(\mathrm{ConF}_{\bullet 2})$, and $(\mathrm{ConU}_{\bullet 2})$. A *$\beta$-redex* in $\lambda_1^{\to\bullet}$ (resp. $\lambda_2^{\to\bullet}$) is a term of the form $(\lambda x.t)s$ or $(\hat{x}.t)@s$ (resp. a term of the form $(\lambda x.t)s$ or $\mathsf{unfold}(\mathsf{fold}(t))$). Roughly speaking, we can thus identify

proofs with pure terms). For the distinction between Church-style and Curry-style ways of typing, see, e.g., [5].

[18] The idea is that judgements have the same format of sequents (as they are defined in sequent calculus).

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

$$\frac{}{(\lambda x.t)s \rhd t[s/x]}\ \beta \qquad \frac{}{\lambda x.tx \rhd t}\ \eta \text{ with } x \text{ not free in } t$$

$$\frac{t \rhd s}{\lambda x.t \rhd \lambda x.s}\ \xi \qquad \frac{t \rhd s}{t\ u \rhd s\ u}\ \mathrm{ConL} \qquad \frac{t \rhd s}{u\ t \rhd u\ s}\ \mathrm{ConR}$$

$$\frac{}{(\hat{x}.t)@s \rhd t[s/x]}\ \beta_{\bullet 1} \qquad \frac{}{\hat{x}.t@x \rhd t}\ \eta_{\bullet 1} \text{ with } x \text{ not free in } t$$

$$\frac{t \rhd s}{\hat{x}.t \rhd \hat{x}.s}\ \xi_{\bullet 1} \qquad \frac{t \rhd s}{t@u \rhd s@u}\ \mathrm{ConL}_{\bullet 1} \qquad \frac{t \rhd s}{u@t \rhd u@s}\ \mathrm{ConR}_{\bullet 1}$$

$$\frac{}{\mathsf{unfold}(\mathsf{fold}(t)) \rhd t}\ \beta_{\bullet 2} \qquad \frac{}{\mathsf{fold}(\mathsf{unfold}(t)) \rhd t}\ \eta_{\bullet 2}$$

$$\frac{t \rhd s}{\mathsf{fold}(t) \rhd \mathsf{fold}(s)}\ \mathrm{ConF}_{\bullet 2} \qquad \frac{t \rhd s}{\mathsf{unfold}(t) \rhd \mathsf{unfold}(s)}\ \mathrm{ConU}_{\bullet 2}$$

Figure 4. Reduction rules.

the β-redexes with those terms that can occur on the left-hand side of β-reduction rules. The notion of η-redex is defined in a similar way.

The β-reduction rule ($\beta_{\bullet 1}$) corresponds to the inversion principle for $\bullet$ following the first formulation (cf. §2.1), and the η-reduction rule ($\eta_{\bullet 1}$) is the symmetric counterpart of the recovery principle for $\bullet$ in the first formulation (since the recovery principle corresponds in general to the *η-expansion* rule):

$$\dfrac{\dfrac{\overset{n}{[x:\bullet]}\quad \mathcal{D}_1}{\hat{x}.t:\bullet}\ \bullet 1_{I\,n} \qquad \mathcal{D}_2}{(\hat{x}.t)@s:\bot}\ \bullet 1_E \quad \rhd \quad \dfrac{\mathcal{D}_2}{s:\bullet}\quad \mathcal{D}_1[s/x] \qquad t[s/x]:\bot \qquad \dfrac{\dfrac{\mathcal{D}}{t:\bullet} \qquad \overset{n}{[x:\bullet]}}{t@x:\bot}\ \bullet 1_E}{\hat{x}.t@x:\bullet}\ \bullet 1_{I\,n} \quad \rhd \quad \mathcal{D}\ t:\bullet.$$

Similarly, in the second formulation of $\bullet$ in §2.1, the rule ($\beta_{\bullet 2}$) corresponds to the inversion principle for $\bullet$, while the rule ($\eta_{\bullet 2}$) is the symmetric counterpart of the recovery principle for $\bullet$:

$$\dfrac{\dfrac{\mathcal{D}}{t:\bullet\to\bot}\ \bullet 2_I}{\mathsf{unfold}(\mathsf{fold}(t)):\bullet\to\bot}\ \bullet 2_E \quad \rhd \quad \mathcal{D}\ t:\bullet\to\bot \qquad \dfrac{\dfrac{\mathcal{D}}{t:\bullet}\ \bullet 2_E}{\mathsf{fold}(\mathsf{unfold}(t)):\bullet}\ \bullet 2_I \quad \rhd \quad \mathcal{D}\ t:\bullet.$$

As it has already been discussed in the literature (see [57, p. 80] and [67, p. 598]), the derivations of $\bot$ we presented in §2.1 can be decorated by means of a language expanding that of λ-calculus, so that we can have a term corresponding to the proof of a contradiction. This term contains β-redexes, but its reduction does not terminate.

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

In particular, this term cannot be reduced to a β-normal form; it cannot be reduced to another term that does not contain any β-redex. For example, the contradictory derivation in the first formulation of $\bullet$ can be represented in $\lambda_1^{\rightarrow\bullet}$ as follows:

$$\dfrac{\dfrac{\dfrac{\overset{1}{[x:\bullet]}\quad\overset{1}{[x:\bullet]}}{x@x:\bot}\bullet 1_E}{\hat{x}.x@x:\bullet}\bullet 1_{I\,1}\qquad\dfrac{\dfrac{\overset{2}{[x:\bullet]}\quad\overset{2}{[x:\bullet]}}{x@x:\bot}\bullet 1_E}{\hat{x}.x@x:\bullet}\bullet 1_{I\,2}}{(\hat{x}.x@x)@(\hat{x}.x@x):\bot}\bullet 1_E \quad\rhd\quad \dfrac{\dfrac{\dfrac{\overset{1}{[x:\bullet]}\quad\overset{1}{[x:\bullet]}}{x@x:\bot}\bullet 1_E}{\hat{x}.x@x:\bullet}\bullet 1_{I\,1}\qquad\dfrac{\dfrac{\overset{2}{[x:\bullet]}\quad\overset{2}{[x:\bullet]}}{x@x:\bot}\bullet 1_E}{\hat{x}.x@x:\bullet}\bullet 1_{I\,2}}{(\hat{x}.x@x)@(\hat{x}.x@x):\bot}\bullet 1_E.$$

One can define a non-terminating term in $\lambda_2^{\rightarrow\bullet}$ in a similar way. Now, if terms are interpreted here as computable functions (or programs), the fact of reaching a normal form can be interpreted as the fact of obtaining a value. Terms like those we just presented, which keep on reducing without reaching a normal form, can thus be interpreted as *partial* (computable) functions (or partial programs, namely, programs that do not always give an output). Partial functions are not problematic from a purely computational point of view, which is the point of view that we want to adopt here: the standard models of computation—like recursive functions, λ-calculus, Turing computability, etc.—take partiality into account. Partiality becomes problematic when we want to assign a semantic role to computational processes by interpreting them as ways of fixing the denotation of linguistic expressions.[19] According to such an interpretation, values are associated with denotations (see [33, p. 160]),[20] and partiality is taken to be a symptom of paradoxical situations: when we have a paradox, we keep on computing without being able to reach a value, that is, the denotation of an expression (see [67, secs. 2 and 3] for more details, especially the idea that a canonical proof can be seen as the denotation of a proposition). Imposing totality of computable functions (or programs) would thus be a way of blocking some possible paradoxical situation; however, in this way, many interesting computational procedures could be lost.[21]

Another approach would be to analyse computation independently of its possible semantics use. This is indeed something which naturally emerges from the analysis we are offering by considering typing systems. Consider the term $(\hat{x}.x@x)@(\hat{x}.x@x)$, which we constructed above. This keeps track of all the rules applied in the first derivation of $\bot$ at p. 7. However, we cannot say that such a term codifies a proof since, as we have shown in the previous section, this derivation of $\bot$ is not valid in Prawitz's sense (although it is closed). As a consequence, we cannot consider that the types involved in the derivation of the judgement $\vdash (\hat{x}.x@x)@(\hat{x}.x@x):\bot$ correspond to

[19] More precisely, according to some authors influenced by a Fregean theory of meaning, the sense (*Sinn*) of an expression must be interpreted as a computational process allowing one to fix (or to reach) the reference or the denotation (*Bedeutung*) of such an expression. See [18, 39, 66] for more details.

[20] As noticed by Martin-Löf in [33], the German term 'Bedeutung' can be translated not only as 'denotation' but also as 'value'. See also [25].

[21] Here, we take partiality as the lack of β-normal form. However, this is not the only way of defining partiality for functions in λ-calculus. Another way is to use the notion of unsolvability (see [74]), which is discussed by [44, p. 616, fn. 18], in relation to looping paradoxical derivations. A definition of solvable and unsolvable λ-terms can also be found in [4, p. 41].

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

$$A, B, C ::= P \mid \bot \mid A \to B \mid \bullet^A \qquad \text{types}$$
$$s, t, u ::= x \mid \lambda x.t \mid \mathsf{app}(t, s) \mid \mathsf{elim}_\bot(t) \mid \hat{x}^A.t \mid @^A(t, s) \qquad \text{terms in the first version}$$
$$s, t, u ::= x \mid \lambda x.t \mid \mathsf{app}(t, s) \mid \mathsf{elim}_\bot(t) \mid \mathsf{fold}^A(t) \mid \mathsf{unfold}^A(t) \qquad \text{terms in the second version}$$

Figure 5. Types and terms of $\lambda_1^{\to\bullet}$ and $\lambda_2^{\to\bullet}$.

$$\frac{\Gamma, x : \bullet^A \vdash t : A}{\Gamma \vdash \hat{x}^A.t : \bullet^A}\ \bullet^A 1_I \qquad \frac{\Gamma \vdash t : \bullet^A \quad \Gamma \vdash s : \bullet^A}{\Gamma \vdash t@^A s : A}\ \bullet^A 1_E$$

$$\frac{}{(\hat{x}^A.t)@^A s \rhd t[s/x]}\ \beta_{\bullet A1} \qquad \frac{}{\hat{x}^A.t@^A x \rhd t}\ \eta_{\bullet A1}$$

Figure 6. Generalised $\lambda_1^{\to\bullet}$-typing and reduction rules.

propositions. At best, they can be interpreted as purely syntactic formulas, devoid of any meaning (if we consider that meaning is fixed by proof-conditions).[22]

In what follows, we will consider terms solely from a computational point of view, and we will show that there are interesting computational situations we can account for by means of the typing rules for the $\bullet$ connective, where partiality is involved. To do this, we consider a generalisation of the $\bullet$ connective given by Read in [55, pp. 574–575].[23] Note that he described this generalised $\bullet$ connective as a kind of proof-conditional Curry's paradox. Although Read considers the generalisation of its first version only, one can generalise the second version in a similar way. His idea is to replace $\bot$ with an arbitrary formula (or type) A in the formulation of $\bullet$. Hereafter the expression $\lambda_1^{\to\bullet}$ (resp. $\lambda_2^{\to\bullet}$) is used to denote the system obtained by extending simply typed λ-calculus with the first version (resp. the second version) of the generalised Bullet connective. The syntax is defined as in Figure 5, while the inference and reduction rules for the first version of $\bullet^A$ are formulated in Figure 6. The congruence rules ($\mathrm{ConL}_{\bullet 1}$) and ($\mathrm{ConR}_{\bullet 2}$) are generalised accordingly. For each type A, a non-terminating term of type A is obtained as in the case of $\bot$. Note that this non-terminating term of type A is closed; hence, we have a closed term (i.e., a closed derivation) of A. A similar generalisation of $\bullet$ in the second formulation can be obtained as in Figure 7.

It is straightforward to prove a weakening lemma for both $\lambda_1^{\to\bullet}$ and $\lambda_2^{\to\bullet}$. This lemma allows us to simulate the weakening rule in the case of natural deduction rules presented in the sequent-style format.

[22] The typing systems we are considering are inspired by the Curry–Howard correspondence but do not fully enjoy it. A term here does not correspond to a proof in a strict sense, but only to a formal (not necessarily valid, in the sense of Prawitz) derivation. In the rest of this section, we will try to offer further arguments to better specify why we do not have a proper Curry–Howard correspondence for the typing systems involving the Bullet connectives.

[23] An exercise in [73, chap. 5] formulates a type denoted by L_A with an arbitrary type A; this type corresponds to Read's generalised Bullet connective with the *general* elmination rule. We thank Ansten Klev for having pointed to this reference. For a formulation of general elimination rules, see, e.g., [55].

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

$$\frac{\Gamma \vdash t : \bullet^A \to A}{\Gamma \vdash \mathsf{fold}^A(t) : \bullet^A}\ \bullet^A 2_I \qquad \frac{\Gamma \vdash t : \bullet^A}{\Gamma \vdash \mathsf{unfold}^A(t) : \bullet^A \to A}\ \bullet^A 2_E$$

$$\frac{}{\mathsf{unfold}^A(\mathsf{fold}^A(t)) \rhd t}\ \beta_{\bullet A2} \qquad \frac{}{\mathsf{fold}^A(\mathsf{unfold}^A(t) \rhd t}\ \eta_{\bullet A2}$$

Figure 7. Generalised $\lambda_2^{\to\bullet}$-typing and reduction rules.

LEMMA 3.1 (Weakening lemma). *In both $\lambda_1^{\to\bullet}$ and $\lambda_2^{\to\bullet}$, if a typing judgement $\Gamma, \Delta \vdash t : A$ is derivable and a variable x does not occur in Γ, Δ, then for any type B, we have $\Gamma, x : B, \Delta \vdash t : A$.*

We will now assign an explicit computational interpretation to the rules of this generalised version of $\bullet$. We will show that this generalisation of $\bullet$ makes it possible to define a general version of a fixed-point operator.

3.2. Fixed-point operator and recursive types. The computational behaviour of the generalised Bullet connective $\bullet^A$ reminds one of the *general fixed-point operator* fix discussed in the literature on Martin-Löf type theory (see, e.g., [35, 43]).[24] The typing and reduction rules for fix can be formulated as follows[25]:

$$\frac{\Gamma, x : A \vdash f : A}{\Gamma \vdash \mathsf{fix}((x)f) : A}\ \mathsf{fix}_I \qquad \mathsf{fix}((x)f) \rhd f[\mathsf{fix}((x)f)/x].$$

The variable x in the term f is bound in the term $\mathsf{fix}((x)f)$. As shown by these rules, the role of the operator fix is to provide such a term f of type A with its 'fixed-point' $\mathsf{fix}((x)f)$.

By using the typing rule for fix, we have the following closed term of type A:

$$\frac{\dfrac{x : A \in \{x : A\}}{x : A \vdash x : A}\ \mathrm{Var}}{\vdash \mathsf{fix}((x)x) : A}\ \mathsf{fix}_I$$

for each type A. Hence, fix provides a closed derivation for each formula A, in the same way as the generalised Bullet connectives do. The resulting term $\mathsf{fix}((x)x)$ is non-terminating as follows, where we denote syntactic identity by $\equiv$.

$$\mathsf{fix}((x)x) \rhd x[\mathsf{fix}((x)x)/x] \equiv \mathsf{fix}((x)x) \rhd x[\mathsf{fix}((x)x)/x] \equiv \mathsf{fix}((x)x) \rhd \cdots .$$

Here, we show that $\lambda_1^{\to\bullet}$ allows one to deduce the typing rule for fix so that its reduction rule is also derivable. In Appendix 1, we show that $\lambda_2^{\to\bullet}$ allows one to do the

[24] The version of Martin-Löf's type theory containing fix is not the standard one, since partial functions are definable (as we will show). In particular, this non-standard version of Martin-Löf's type theory goes against some of the principles exposed in [37], where totality of functions and termination of computations play a crucial role. For the early works to reconcile partial functions with constructive type theory, see, e.g., [13, 14].

[25] Note that in [35, 43], the reduction rule for $\mathsf{fix}((x)f)$ is formulated by means of definitional equality. Definitional equality (sometimes also called judgemental equality) is an equality between typed terms which includes the relation of syntactic identity and also an equivalence relation induced by the reduction relation. A thorough discussion of definitional and judgemental equality in the framework of Martin-Löf's type theory can be found in [68].

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

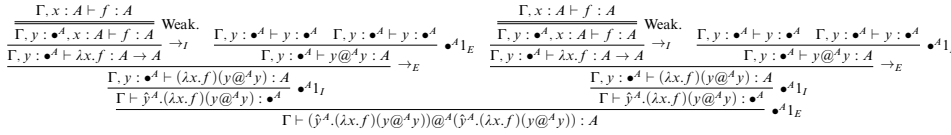


Figure 8. Interpretation of fix in $\lambda_1^{\to\bullet}$.

same. Let a variable y be fresh for a context $\Gamma, x : A$, then we have a $\lambda_1^{\to\bullet}$-derivation as the one shown in Figure 8 (for the sake of brevity, we omit the application of the Var-rule). We define

$$\mathsf{fix}((x)f) :\equiv (\hat{y}^A.(\lambda x.f)(y@^A y))@^A(\hat{y}^A.(\lambda x.f)(y@^A y)),$$

where $:\equiv$ indicates that the term on the left is a syntactic abbreviation for the term on the right. The derivation in Figure 8 then shows that the typing rule for fix is derivable in $\lambda_1^{\to\bullet}$. Moreover, we have

$$\begin{aligned}\mathsf{fix}((x)f) &\equiv (\hat{y}^A.(\lambda x.f)(y@^A y))@^A(\hat{y}^A.(\lambda x.f)(y@^A y))\\ &\rhd (\lambda x.f)((\hat{y}^A.(\lambda x.f)(y@^A y))@^A(\hat{y}^A.(\lambda x.f)(y@^A y)))\\ &\rhd f[((\hat{y}^A.(\lambda x.f)(y@^A y))@^A(\hat{y}^A.(\lambda x.f)(y@^A y)))/x]\\ &\equiv f[\mathsf{fix}((x)f)/x],\end{aligned}$$

so the reduction rule for fix is derivable.

The derivability of fix enables to solve recursive equations which have several applications. Let $\overset{*}{\rhd}$ be the reflexive and transitive closure of a given reduction relation $\rhd$, while $\overset{*}{\bowtie}$ is defined as the reflexive, symmetric, and transitive closure of $\rhd$. Then, the following proposition holds by a standard argument (cf., e.g., [5, theorem 2.1.9]).

PROPOSITION 3.2. *In $\lambda_1^{\to\bullet}$ or $\lambda_2^{\to\bullet}$, let a term $\psi[f, x]$ be given with $\Gamma, f : B \to A, x : B \vdash \psi[f, x] : A$. Then, there is a term φ such that $\Gamma \vdash \varphi : B \to A$ holds and we have $\varphi\, u \overset{*}{\rhd} \psi[\varphi, u]$ for any u with $\Gamma \vdash u : B$.*

Proof. We consider only the case of $\lambda_1^{\to\bullet}$ here, since the case of $\lambda_2^{\to\bullet}$ is similar. Let a term $\psi[f, x]$ be given with $\Gamma, f : B \to A, x : B \vdash \psi[f, x] : A$, so that $\Gamma, f : B \to A \vdash \lambda x.\psi[f, x] : B \to A$ holds. By the typing rule of fix, we have $\Gamma \vdash \mathsf{fix}((f)\lambda x.\psi[f, x]) : B \to A$.

We define $\varphi :\equiv \mathsf{fix}((f)\lambda x.\psi[f, x])$. Then, for any $\Gamma \vdash u : B$,

$$\varphi\, u \equiv (\mathsf{fix}((f)\lambda x.\psi[f, x]))u \overset{*}{\rhd} (\lambda x.\psi[\varphi, x])u \rhd \psi[\varphi, u]$$

holds. □

To observe an example of the proposition above, we consider an extension of $\lambda_1^{\to\bullet}$ or $\lambda_2^{\to\bullet}$ with the integer type Int and the Boolean type Bool, and put ψ in this proposition as follows:

$$\psi[f, x] :\equiv \texttt{if}\ \ x \leq 0\ \ \texttt{then}\ \ 1\ \ \texttt{else}\ \ x \times f\ (x - 1)$$

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

$$A, B, C ::= X \mid A \to B \mid \mu X.A \qquad \text{types}$$
$$s, t, u ::= x \mid \lambda x.t \mid \mathsf{app}(t, s) \mid \mathsf{fold}(t) \mid \mathsf{unfold}(t) \qquad \text{terms}$$

Figure 9. Syntax of $\lambda^{\to\mathsf{rec}}$.

$$\frac{\Gamma \vdash t : A[\mu X.A/X]}{\Gamma \vdash \mathsf{fold}(t) : \mu X.A}\ \mu_I \qquad \frac{\Gamma \vdash t : \mu X.A}{\Gamma \vdash \mathsf{unfold}(t) : A[\mu X.A/X]}\ \mu_E$$

$$\frac{}{\mathsf{unfold}(\mathsf{fold}(t)) \rhd t}\ \beta_\mu \qquad \frac{}{\mathsf{fold}(\mathsf{unfold}(t)) \rhd t}\ \eta_\mu$$

Figure 10. $\lambda^{\to\mathsf{rec}}$-typing and reduction rules.

with $f : \mathsf{Int} \to \mathsf{Int}$ and $x : \mathsf{Int}$. We then have a term $\mathsf{fac} : \mathsf{Int} \to \mathsf{Int}$ satisfying the 'equation'

$$\mathsf{fac}\, l \overset{*}{\rhd} \mathtt{if}\ \ l \leq 0\ \ \mathtt{then}\ \ 1\ \ \mathtt{else}\ \ l \times \mathsf{fac}\,(l-1),$$

which provides the factorial function whenever l is a positive integer.

Next, we show that the second version of the generalised Bullet connective is an instance of non-normalising recursive types. This implies that its first version is definable as a recursive type as well, because the inference rules of the first version are derivable from those of the second version. Below, we consider the non-normalising recursive types discussed, for instance, in [24, chap. 7] and [48, chap. IV]. These recursive types have several applications; they allow one to define not only the general fixed-point operator fix discussed above, but also many examples of datatypes, such as the type of natural numbers and the one of lists of natural numbers. Thus, one can provide the recursive definition of a function on these datatypes by solving the corresponding recursive equation with fix.

Our base theory is simply typed λ-calculus, and we denote by $\lambda^{\to\mathsf{rec}}$ the system obtained by adding recursive types to simply typed λ-calculus. Types and terms of $\lambda^{\to\mathsf{rec}}$ are defined in Figure 9, where the vocabulary contains type variables $X_0, X_1, \ldots$ instead of atomic types. In particular, a type variable X is bound in a type of the form $\mu X.A$, which is a recursive type. Similarly to what we did for the substitution of a term variable, we denote by $A[B/X]$ the type resulting from the substitution of a type B for all occurrences of a type variable X in a type A.

The system $\lambda^{\to\mathsf{rec}}$ has the typing and reduction rules for recursive types in Figure 10. These rules go together with the suitable congruence rules.

Let $\bullet^A :\equiv \mu X.X \to A$, where the variable X is not free in A.[26] Then, we have

$$\bullet^A \to A \equiv (\mu X.X \to A) \to A \equiv (X \to A)[\mu X.X \to A/X],$$

so that it is easy to see that the typing and reduction rules for the recursive type $(\mu X.X \to A) \to A$ are exactly those of the second version of the generalised Bullet

[26] Discussing Russell's paradox in the framework of combinatory logic, Curry and Feys construct a formula having a similar behaviour to the type $\bullet^A$ by means of a fixed-point combinator (see [17, p. 258]). We thank Ansten Klev for pointing to this reference.

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

connective. This means that the typing and reduction rules for the recursive type $(\mu X.X \to A) \to A$ also provide a non-terminating (or non-normalising) term.[27] Furthermore, this explains why, in the second version of the Bullet connective, the introduction and elimination rules are called fold and unfold, respectively. If one considers the second version as a recursive type, its introduction rule 'folds' a type into the Bullet connective, and the elimination rule 'unfolds' the Bullet connective.

The comparison with recursive types sheds light on another feature of the Bullet connectives, corroborating the idea that they cannot be considered as logical connectives forming propositions. Given a recursive type $\mu X.A$, its corresponding rules make it possible to pass from it to $A[\mu X.A/X]$, and vice versa. One can thus analyse a recursive type $\mu X.A$ with respect to the position of the variable X in A, that is, the variable on which the substitution (with the recursive types themselves) is operated. More precisely, we first define a positive/negative occurrence of a type variable in a type of $\lambda^{\to \mathsf{rec}}$ (cf. [1]).

Definition 3.3 (Positive/negative occurrences). *Let a sign* s *be either* $+$ *or* $-$. *Putting* $-+ = -$ *and* $--= +$, *we define an occurrence of a type variable* X *to be in a* s*-position of a type* A *inductively*:

1. X *occurs in a* $+$*-position of* X.
2. X *occurs in an* s*-position of* $A \to B$ *if* X *occurs in an* s*-position of* B.
3. X *occurs in a* $(-\,\mathsf{s})$*-position of* $A \to B$ *if* X *occurs in an* s*-position of* A.
4. X *occurs in an* s*-position of* $\mu Y.A$ *if* X *occurs in an* s*-position of* A.

When X *occurs in a* $+$*-position* (*resp. a* $-$*-position*) *of* A, *we say that* X *occurs positively* (*resp. negatively*) *in* A.

Note that a type variable X may occur in A both positively and negatively: for instance, X occurs in $X \to X$ both positively and negatively. Next, we define a narrower notion of positive occurrence, namely, the notion of strictly positive occurrence.[28]

Definition 3.4 (Strict positivity). *We say that a type variable* X *occurs* strictly positively *in a type* A *if* A *is of the form* $B_1 \to (\cdots \to (B_n \to X)\ldots)$ *and* B_i *does not contain any occurrence of* X *for each* i $(1 \leq i \leq n)$.

Note that a type variable X does not occur negatively in a type A whenever X occurs strictly positively in A.

The notion of strictly positive recursive type can then be defined by requiring that in a type $\mu X.A$ the variable X occurs strictly positively in A. Strict positiveness is usually imposed as a condition on a system with recursive types in order to guarantee the termination of the reduction relation in such a system (see, e.g., [15]). Note, however, that in the definition of $\bullet^A$ as $\mu X.X \to A$, the variable X does not occur in a strictly positive way in $X \to A$ (in fact, it occurs negatively). As Dyckhoff [20, p. 83] remarks, this can be seen as yet another symptom of the circularity of the Bullet connective:

[27] While our base theory is simply typed λ-calculus, the normalisation problem of monotone recursive types in the setting of Girard's System F has been discussed in [38].

[28] Here, we follow the definition given by [7], and we are only considering the case in which A is formed by using $\to$. For a more general notion of strictly positive occurrence of a variable in a formula, see [69, pp. 5–6].

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

by unfolding $\mu X.X \to A$, one substitutes $\mu X.X \to A$ for X in $X \to A$, and because of the position of X, this allows one to obtain the type $(\mu X.X \to A) \to A$, which not only folds to $\mu X.X \to A$, but can also be applied to $\mu X.X \to A$ itself (which is indeed the symptom of circularity). In this way, in particular, it is possible to provide a non-terminating closed term t for any arbitrary type A, which means that every type A is inhabited.[29] As Pierce remarks:

> This fact makes systems with recursive types useless as logics: if we interpret types as logical propositions following the Curry–Howard correspondence [...] and read "type T is inhabited" as "proposition T is provable," then the fact that every type is inhabited means that every proposition in the logic is provable—that is, the logic is inconsistent. [48, p. 273]

Here, again, the problem is considering that a type can be interpreted as a proposition: $\vdash t : A$ (where t is closed) would mean that there is a closed derivation, and thus a proof, of A coded by t.[30] The solution is to drop the identification between types and propositions, and to look at a term simply as a program. In this sense, as we already mentioned in §1.1, a type would correspond to the *classification* of a program. By assigning a type to a term (standing for a program), we describe its behaviour from a formal point of view. Accordingly, a typable term does not require being terminating, because non-terminating terms or programs are often useful as the general fixed-point operator fix is, and recursive types can be used to type such programs.

In fact, we can push the analysis of the Bullet connective further via recursive types. As we have seen, the rules for recursive types allow one to pass from a recursive type to its unfolded version, and vice versa. However, what is the exact relation between a recursive type and its unfolded version? One way of looking at it is to consider that they are *isomorphic types* (see [48, pp. 276–277]). We will use this notion to compare $\bullet^A$ with other recursive types, and show that they allow for different instances of type isomorphisms. We denote the identity $\lambda x.x$ of type $A \to A$ by 1_A, and define the *composition* $s \circ t$ of two terms $t : A \to B$ and $s : B \to C$ as $\lambda x.s\,(t\,x) : A \to C$. In proof-theoretic terms, the notion of type isomorphism is defined as follows (cf. [10, 16]): two types A and B are *provably isomorphic* in a type system if, in this system, there are closed terms $t : A \to B$ and $s : B \to A$ such that

$$s \circ t \overset{*}{\rhd\!\lhd} 1_A, \qquad t \circ s \overset{*}{\rhd\!\lhd} 1_B$$

hold. Intuitively, the idea is that the term t is an 'isomorphism' from A to B.

While the system $\lambda^{\to\mathsf{rec}}$ allows one to show that there is a type A provably isomorphic to $A \to A$, the systems $\lambda_1^{\to\bullet}$ and $\lambda_2^{\to\bullet}$ allow one only to show that there is a provable isomorphism between $\bullet^A$ and $\bullet^A \to A$, for an arbitrary type A. The reason is that the inference rules for $\bullet^A$ relate it with the implication (or the arrow type) $\bullet^A \to A$, where the antecedent $\bullet^A$ is always distinct from the succedent A. Let us first show that there is a type A provably isomorphic to $A \to A$ in $\lambda^{\to\mathsf{rec}}$. Let $\mathsf{Ref} :\equiv \mu X.X \to X$ and consider

---

[29] Such a term t corresponds indeed to $(\mathsf{unfold}(\mathsf{fold}(\lambda x(\mathsf{unfold}(x)x))))(\mathsf{fold}(\lambda x(\mathsf{unfold}(x)x)))$.

[30] Actually, when t is non-terminating, this means that the detours present in the closed derivation coded by t cannot be eliminated, and thus this derivation is not valid in Prawitz's sense. Therefore, strictly speaking, it is not a proof.

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

then the following proposition.[31] Note that Ref is defined by means of the formula $X \to X$ with two occurrences of X, one of which is negative and the other is positive.

PROPOSITION 3.5. *In* $\lambda^{\to\mathsf{rec}}$, Ref *and* $\mathsf{Ref} \to \mathsf{Ref}$ *are provably isomorphic.*

Proof. Take the following $\lambda^{\to\mathsf{rec}}$-derivations, where the Var-rule is omitted:

$$\dfrac{\dfrac{x : \mathsf{Ref} \to \mathsf{Ref} \vdash x : \mathsf{Ref} \to \mathsf{Ref}}{x : \mathsf{Ref} \to \mathsf{Ref} \vdash \mathsf{fold}(x) : \mathsf{Ref}}\,\mu_I}{\vdash \lambda x.\mathsf{fold}(x) : (\mathsf{Ref} \to \mathsf{Ref}) \to \mathsf{Ref}}\,{\to_I} \qquad \dfrac{\dfrac{y : \mathsf{Ref} \vdash y : \mathsf{Ref}}{y : \mathsf{Ref} \vdash \mathsf{unfold}(y) : \mathsf{Ref} \to \mathsf{Ref}}\,\mu_E}{\vdash \lambda y.\mathsf{unfold}(y) : \mathsf{Ref} \to (\mathsf{Ref} \to \mathsf{Ref})}\,{\to_I}.$$

Define then $\psi :\equiv \lambda x.\mathsf{fold}(x)$ and $\varphi :\equiv \lambda y.\mathsf{unfold}(y)$. These closed terms ψ, φ satisfy the following reduction relations:

$$\begin{aligned}
\varphi \circ \psi \equiv \lambda z.\varphi\ (\psi\ z) &\equiv \lambda z.(\lambda y.\mathsf{unfold}(y))((\lambda x.\mathsf{fold}(x))z) \\
&\rhd \lambda z.(\lambda y.\mathsf{unfold}(y))(\mathsf{fold}(z)) && [\text{by } \beta] \\
&\rhd \lambda z.\mathsf{unfold}(\mathsf{fold}(z)) && [\text{by } \beta] \\
&\rhd \lambda z.z && [\text{by } \beta_\mu] \\
\psi \circ \varphi \equiv \lambda w.\psi\ (\varphi\ w) &\equiv \lambda w.(\lambda x.\mathsf{fold}(x))((\lambda y.\mathsf{unfold}(y))w) \\
&\rhd \lambda w.(\lambda x.\mathsf{fold}(x))(\mathsf{unfold}(w)) && [\text{by } \beta] \\
&\rhd \lambda w.\mathsf{fold}(\mathsf{unfold}(w)) && [\text{by } \beta] \\
&\rhd \lambda w.w. && [\text{by } \eta_\mu]
\end{aligned}$$

The type of the term $\lambda z.z$ is $(\mathsf{Ref} \to \mathsf{Ref}) \to (\mathsf{Ref} \to \mathsf{Ref})$, thus this term corresponds to $1_{\mathsf{Ref}\to\mathsf{Ref}}$, while the term $\lambda w.w$ is of type $\mathsf{Ref} \to \mathsf{Ref}$ and corresponds to 1_{Ref}. Therefore, Ref and $\mathsf{Ref} \to \mathsf{Ref}$ are provably isomorphic. □

Here in $\lambda^{\to\mathsf{rec}}$ we use both β-reduction and η-reduction, which correspond to the inversion principle and the (converse of) recovery principle, respectively. An argument similar to the previous one allows us to prove the fact that in $\lambda_2^{\to\bullet}$, $\bullet^A$ and $\bullet^A \to A$ are provably isomorphic for any type A. This statement also holds in $\lambda_1^{\to\bullet}$, as shown below.

PROPOSITION 3.6. *In* $\lambda_1^{\to\bullet}$, $\bullet^A$ *and* $\bullet^A \to A$ *are provably isomorphic for any type* A.

Proof. We construct an isomorphism as follows. Consider first the two derivations below:

$$\dfrac{\dfrac{\dfrac{y : \bullet^A \to A, x : \bullet^A \vdash y : \bullet^A \to A \quad y : \bullet^A \to A, x : \bullet^A \vdash x : \bullet^A}{y : \bullet^A \to A, x : \bullet^A \vdash y\ x : A}\,{\to_E}}{y : \bullet^A \to A \vdash \hat{x}^A.y\ x : \bullet^A}\,{\bullet^A 1_I}}{\vdash \lambda y.\hat{x}^A.y\ x : (\bullet^A \to A) \to \bullet^A}\,{\to_I}$$

$$\dfrac{\dfrac{\dfrac{v : \bullet^A, z : \bullet^A \vdash v : \bullet^A \quad v : \bullet^A, z : \bullet^A \vdash z : \bullet^A}{v : \bullet^A, z : \bullet^A \vdash v@^A z : A}\,{\bullet^A 1_E}}{v : \bullet^A \vdash \lambda z.v@^A z : \bullet^A \to A}\,{\to_I}}{\vdash \lambda v.\lambda z.v@^A z : \bullet^A \to \bullet^A \to A}\,{\to_I}.$$

[31] We abbreviate the type $\mu X.X \to X$ as Ref by analogy with the *reflexive objects* discussed in [2, chap. 4].

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

Then, we have the following reductions, which preserve types:

$$\begin{aligned}
(\lambda v.\lambda z.v@^A z) \circ (\lambda y.\hat{x}^A.y\ x) &\equiv \lambda w.(\lambda v.\lambda z.v@^A z)((\lambda y.\hat{x}^A.y\ x)w) \\
&\rhd \lambda w.(\lambda v.\lambda z.v@^A z)(\hat{x}^A.w\ x) && [\text{by } \beta] \\
&\rhd \lambda w.\lambda z.(\hat{x}^A.w\ x)@^A z && [\text{by } \beta] \\
&\rhd \lambda w.\lambda z.w\ z && [\text{by } \beta_{\bullet 1}] \\
&\rhd \lambda w.w && [\text{by } \eta] \\
&\equiv 1_{\bullet^A \to A}
\end{aligned}$$

$$\begin{aligned}
(\lambda y.\hat{x}^A.y\ x) \circ (\lambda v.\lambda z.v@^A z) &\equiv \lambda w.(\lambda y.\hat{x}^A.y\ x)((\lambda v.\lambda z.v@^A z)w) \\
&\rhd \lambda w.(\lambda y.\hat{x}^A.y\ x)(\lambda z.w@^A z) && [\text{by } \beta] \\
&\rhd \lambda w.\hat{x}^A.(\lambda z.w@^A z)x && [\text{by } \beta] \\
&\rhd \lambda w.\hat{x}^A.w@^A x && [\text{by } \beta] \\
&\rhd \lambda w.w && [\text{by } \eta_{\bullet 1}] \\
&\equiv 1_{\bullet^A.}
\end{aligned}$$

This is sufficient to obtain an isomorphism between $\bullet^A \to A$ and $\bullet^A$. □

As we said, $\bullet$ is just a special case of $\bullet^A$ (i.e., when $A \equiv \bot$). This result provides another reason for considering $\bullet$ as a paradoxical connective. This connective is not only paradoxical in that it mimics, at the proposition level, the Liar paradox or Russell's paradox. This connective meets a stronger criterion of paradoxicality, as exposed by Schroeder-Heister and Tranchini (see also Petrolo and Pistone's exposition [44, p. 607]):

> What triggers a genuine paradox is not simply the assumption that a sentence is *interderivable* with its own negation [...]. A genuine paradox is a sentence A such that there are proofs from A to $\neg A$ and from $\neg A$ to A that composed with each other give us the identity proof A (i.e., the formula A considered a proof of A from A). Such a notion, which is stricter than just interderivability, and which is well known in general (in particular categorial) proof theory as *isomorphism* of formulas (see Došen [5]), must be given a much more prominent role in proof-theoretic semantics than it currently enjoys. [60, p. 578]

Although the type isomorphism between $\bullet^A$ and $\bullet^A \to A$ allows us to have a deeper understanding of the Bullet connective (in particular by meeting the condition of paradoxality formulated by Schroeder-Heister and Tranchini), one can ask whether it is possible to make the behaviour of the Bullet connective more similar to that of recursive types in general. We show that it is possible to define a simple variant of the Bullet connective behaving like Ref with respect to type isomorphism. Moreover, thanks to this variant of the Bullet connective, it is possible to interpret type-free (i.e., pure) λ-calculus, a framework which is powerful enough to characterise a Turing-complete model of computation.

3.3. Interpretation of type-free λ-calculus. Recall that the original formulation of the inference rules for $\bullet$—that is before considering the generalisation presented at

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

§3.1—is the following:

$$\frac{\Gamma, x : \bullet \vdash t : \bot}{\Gamma \vdash \hat{x}.t : \bullet}\ \bullet 1_I \qquad \frac{\Gamma \vdash t : \bullet \quad \Gamma \vdash s : \bullet}{\Gamma \vdash t@s : \bot}\ \bullet 1_E$$

$$\frac{\Gamma \vdash t : \bullet \to \bot}{\Gamma \vdash \mathsf{fold}(t) : \bullet}\ \bullet 2_I \qquad \frac{\Gamma \vdash t : \bullet}{\Gamma \vdash \mathsf{unfold}(t) : \bullet \to \bot}\ \bullet 2_E.$$

In order to obtain a variant of $\bullet$ which makes it possible to interpret type-free λ-calculus, it is sufficient to replace $\bot$ with $\bullet$ itself in these rules. We denote this variant by $\bullet^*$, and the system $\lambda_{1*}^{\to\bullet}$ is formulated by extending simply typed λ-calculus with the typing rules in Figure 11. Similarly, the system $\lambda_{2*}^{\to\bullet}$ is obtained by extending simply typed λ-calculus with the typing rules in Figure 12. The reduction rules of $\lambda_{1*}^{\to\bullet}$ and $\lambda_{2*}^{\to\bullet}$ are those of Figure 4.

One can prove the following proposition about type isomorphisms in these systems by adapting the proof of Proposition 3.5 to the case of $\lambda_{2*}^{\to\bullet}$, and that of Proposition 3.6 to the case of $\lambda_{1*}^{\to\bullet}$. Note that, in fact, these proofs allow one to show a stronger statement than the one corresponding to the definition of type isomorphism. They allow one to show the existence of two closed terms whose composition *reduces to* the identity.

PROPOSITION 3.7. *In both $\lambda_{1*}^{\to\bullet}$ and $\lambda_{2*}^{\to\bullet}$, there are two closed terms $\psi : (\bullet^* \to \bullet^*) \to \bullet^*$ and $\varphi : \bullet^* \to (\bullet^* \to \bullet^*)$ with*

$$\varphi \circ \psi \overset{*}{\triangleright} 1_{\bullet^* \to \bullet^*}, \qquad \psi \circ \varphi \overset{*}{\triangleright} 1_{\bullet^*}.$$

COROLLARY 3.8. *In both $\lambda_{1*}^{\to\bullet}$ and $\lambda_{2*}^{\to\bullet}$, the types $\bullet^*$ and $\bullet^* \to \bullet^*$ are provably isomorphic.*

By applying a general result presented in [24, chap. 8, proposition 8.1] to the case of $\bullet^*$, and by using Proposition 3.7, we can provide an interpretation of type-free $\lambda\beta\eta$-calculus in $\lambda_{1*}^{\to\bullet}$, and in $\lambda_{2*}^{\to\bullet}$ as well. First, we define by induction a translation $(\cdot)^*$ from terms of type-free λ-calculus into terms of $\lambda_{1*}^{\to\bullet}$ and $\lambda_{2*}^{\to\bullet}$:

- $x^* := x$;
- $(\lambda x.t)^* := \psi(\lambda x.t^*)$;
- $(t\ u)^* := \varphi\ t^*\ u^*$.

The lemma below guarantees that terms of type-free λ-calculus are translated into terms which are typable with type $\bullet^*$. It is straightforward to prove the weakening lemma for $\lambda_{1*}^{\to\bullet}$ and $\lambda_{2*}^{\to\bullet}$ (cf. Lemma 3.1 above), which will be used implicitly below.

$$\frac{\Gamma, x : \bullet^* \vdash t : \bullet^*}{\Gamma \vdash \hat{x}.t : \bullet^*}\ *1_I \qquad \frac{\Gamma \vdash t : \bullet^* \quad \Gamma \vdash s : \bullet^*}{\Gamma \vdash t@s : \bullet^*}\ *1_E$$

Figure 11. $\lambda_{1*}^{\to\bullet}$-typing rules.

$$\frac{\Gamma \vdash t : \bullet^* \to \bullet^*}{\Gamma \vdash \mathsf{fold}(t) : \bullet^*}\ *2_I \qquad \frac{\Gamma \vdash t : \bullet^*}{\Gamma \vdash \mathsf{unfold}(t) : \bullet^* \to \bullet^*}\ *2_E$$

Figure 12. $\lambda_{2*}^{\to\bullet}$-typing rules.

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

LEMMA 3.9. *Let* t *be an arbitrary term of type-free* λ*-calculus such that the set of all free variables of* t *are* $\{x_1, x_2, \dots, x_n\}$*. The translation* t^* *of* t *satisfies the typing judgement*

$$x_1 : \bullet^*, x_2 : \bullet^*, \dots, x_n : \bullet^* \vdash t^* : \bullet^*$$

in both $\lambda^{\to\bullet}_{1*}$ *and* $\lambda^{\to\bullet}_{2*}$.

Proof. By induction on t. The case of variables is trivial, so we consider the cases of abstraction and application only. By Proposition 3.7, in $\lambda^{\to\bullet}_{1*}$ and $\lambda^{\to\bullet}_{2*}$, respectively, we have two closed terms $\psi : (\bullet^* \to \bullet^*) \to \bullet^*$ and $\varphi : \bullet^* \to (\bullet^* \to \bullet^*)$ satisfying

$$\varphi \circ \psi \overset{*}{\rhd} 1_{\bullet^* \to \bullet^*}, \qquad \psi \circ \varphi \overset{*}{\rhd} 1_{\bullet^*}.$$

Let t be of the form $\lambda x.u$. Using the induction hypothesis with the weakening lemma, we can derive the desired typing judgement as follows:

$$\dfrac{\dfrac{\vdots}{x_1 : \bullet^*, x_2 : \bullet^*, \dots, x_n : \bullet^* \vdash \psi : (\bullet^* \to \bullet^*) \to \bullet^*} \qquad \dfrac{\dfrac{\vdots}{x_1 : \bullet^*, x_2 : \bullet^*, \dots, x_n : \bullet^*, x : \bullet^* \vdash u^* : \bullet^*}}{x_1 : \bullet^*, x_2 : \bullet^*, \dots, x_n : \bullet^* \vdash \lambda x.u^* : \bullet^* \to \bullet^*} \to_I}{x_1 : \bullet^*, x_2 : \bullet^*, \dots, x_n : \bullet^* \vdash \psi(\lambda x.u^*) : \bullet^*} \to_E .$$

Let t be an application term $u\ s$. Using the induction hypothesis with the weakening lemma, we have

$$\dfrac{\dfrac{\vdots}{x_1 : \bullet^*, x_2 : \bullet^*, \dots, x_n : \bullet^* \vdash \varphi : \bullet^* \to (\bullet^* \to \bullet^*)} \quad \dfrac{\vdots}{x_1 : \bullet^*, x_2 : \bullet^*, \dots, x_n : \bullet^* \vdash u^* : \bullet^*}}{x_1 : \bullet^*, x_2 : \bullet^*, \dots, x_n : \bullet^* \vdash \varphi\, u^* : \bullet^* \to \bullet^*} \to_E \qquad \dfrac{\vdots}{x_1 : \bullet^*, x_2 : \bullet^*, \dots, x_n : \bullet^* \vdash s^* : \bullet^*}}{x_1 : \bullet^*, x_2 : \bullet^*, \dots, x_n : \bullet^* \vdash \varphi\, u^*\, s^* : \bullet^*} \to_E .$$

This completes the proof. □

We can now prove that type-free $\lambda\beta\eta$-calculus is interpretable in $\lambda^{\to\bullet}_{1*}$ and $\lambda^{\to\bullet}_{2*}$, respectively.

PROPOSITION 3.10. *The reduction rules of type-free* $\lambda\beta\eta$*-calculus are derivable in* $\lambda^{\to\bullet}_{1*}$ *and* $\lambda^{\to\bullet}_{2*}$*, respectively.*

Proof. We consider the β-reduction rule and the η-reduction rule only. The cases of the remaining reduction rules are straightforward.

For the β-reduction rule,

$$(\lambda x.t)\ s \rhd t[s/x],$$

we have the terms $((\lambda x.t)\ s)^*$ and $(t[s/x])^*$ of type $\bullet^*$, by Lemma 3.9. It is sufficient to verify that the former term reduces to the latter in $\lambda^{\to\bullet}_{1*}$ and $\lambda^{\to\bullet}_{2*}$, respectively. We have

$$((\lambda x.t)\ s)^* = (\varphi(\psi(\lambda x.t^*)))s^* \overset{*}{\rhd} (\lambda x.t^*)s^* \rhd t^*[s^*/x] = (t[s/x])^*,$$

where $t^*[s^*/x]$ is well-typed by Lemma 3.9. So the β-reduction rule is derivable.

In the case of the η-reduction rule, the reduction relation below

$$(\lambda x.t\ x)^* = \psi(\lambda x.(\varphi\ t^*)\ x) \rhd \psi(\varphi\ t^*) \overset{*}{\rhd} t^*$$

holds, so the η-reduction rule is derivable as well. □

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

Note that the interpretation of type-free $\lambda\beta\eta$-calculus into $\lambda_{2*}^{\to\bullet}$ we just presented can be reformulated so as to obtain an interpretation of type-free $\lambda\beta\eta$-calculus into $\lambda^{\to\mathsf{rec}}$, simply by replacing $\bullet^*$ with Ref.

3.4. Bullet connectives as a modality. As mentioned above, the generalised Bullet connective $\bullet^A$ can be regarded as a proof-conditional Curry's paradox (cf. Footnote 26). On the one hand, Curry's paradox is known to have an argument structure similar to that of a proof of Löb's theorem. Whereas the former provides any given formula A with a formula C such that $C \leftrightarrow (C \to A)$ holds, the latter shows that for any recursively enumerable extension T of first-order Peano arithmetic, and for any number-theoretic sentence A, there exists a sentence C such that $C \leftrightarrow (\mathrm{Prov}_T(\ulcorner C \urcorner) \to A)$ is derivable in T, where Prov_T is a provability predicate of T (see, e.g., [6]). On the other hand, Löb's axiom, which is an axiom of Gödel–Löb provability logic corresponding to Löb's theorem, is known to be satisfied by Nakano's modality in typed λ-calculus with recursion [41]. We argue that these analogies can help shed light on the computational meaning of the Bullet connectives further. Specifically, we first consider the fragment of $\lambda_1^{\to\bullet}$ and $\lambda_2^{\to\bullet}$ obtained by omitting the $\bullet^A$-elimination rules, and show that one can derive the *type* corresponding to Nakano's fixed-point operator in this fragment. We then note that this fragment can be interpreted in Nakano's type system, which we denote by $\lambda^{\mu\bullet}$; the interpretation of $\bullet^A$ in Nakano's system not only demonstrates the consistency of the $\bullet^A$-introduction rules, but also provides them with a computational meaning.

Nakano's modality, which allows one to form the type $\bullet A$ for a given type A, approximates a *specification* for a recursive program. A successive sequence $S_0, S_1, \ldots$ of specifications approximating a given one is described by possible worlds and the $\bullet$-accessibility relation in the sense of Kripke semantics.[32] It is this approximation in terms of the modality $\bullet A$ that enables Nakano to define recursive programs including the fixed-point operators in $\lambda^{\mu\bullet}$ without rendering the system logically inconsistent.

The notion of 'specification' invoked by Nakano can be clarified by appealing to the characterisation found in [72, pp. 232–233]. According to this characterisation, a specification for a program is a formal condition about the way in which the inputs and outputs of a program are related. Thus, the specification of a program describes both its behaviour and its associated properties. In this sense, a specification should allow us to state what happens for each input, namely, whether the program terminates on that input and which output it produces if it does terminate. One can then consider a program's type as expressing its specification. For instance, the type $\mathsf{Int} \to \mathsf{Int} \to \mathsf{Int}$ specifies that addition and multiplication on integers take two inputs of type Int and return an output of type Int. Moreover, in a type system satisfying the strong normalisation property, that is, the property that all typable terms terminate, assigning a type to a program guarantees its termination.

In [41], Nakano takes types to express specifications and refines Scott's construction of solutions to recursive equations [61, 62]. In Scott's construction, the solution or function f satisfying a recursive equation $f = F(f)$ is approximated by the sequence $\bot, F(\bot), F(F(\bot)), \ldots$, where F is a Scott-continuous function on a directed-complete partial order with the least element $\bot$ and so we have $\bot \sqsubseteq F(\bot) \sqsubseteq F(F(\bot)) \sqsubseteq \ldots$.

[32] Note that here the modality $\bullet$ is considered a unary connective, whereas we considered the generalised Bullet connective $\bullet^A$ a 0-ary connective.

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

Here, functions can be viewed extensionally as sets of input-output pairs, so that each approximation $F^n(\bot)$ is a finite set of such pairs. Nakano similarly approximates the type (i.e., the specification) S of the solution f using $S_0, S_1, ...$, and the modality is crucial for this latter approximation, as noted above. However, treating types as expressing program specifications involves more than simply claiming that types capture program behaviour. In this respect, Nakano's notion of type is more demanding than the one adopted here.[33] Our perspective in this paper (as explained further in the conclusion) is that a type is merely a way of classifying programs with respect to their behaviour; this does not entail that a type conveys additional properties enabling a more exact description of the program, such as termination on every input. Nakano's conception of type as specification is closer to the notion of type employed in logic (see the Concluding Remarks for a more detailed discussion of the different meanings of the notion of type).

According to Nakano's semantics, the modality $\bullet A$ informally means that A holds at the 'previous time'. The system $\lambda^{\mu\bullet}$ includes the following typing rules (see [41] for details of the $\lambda^{\mu\bullet}$-typing rules):

$$\frac{\bullet\Gamma \vdash t : \bullet A}{\Gamma \vdash t : A}\ \bullet \qquad \frac{\Gamma \vdash t : \bullet^n(A \to B) \quad \Gamma \vdash s : \bullet^n A}{\Gamma \vdash ts : \bullet^n B}\ \to_E,$$

where the notation $\bullet^n$ denotes the n-fold consecutive applications of the modality $\bullet$ with $n \geq 0$, and the context $\bullet\Gamma$ is of the form $x_1 : \bullet A_1, \ldots, x_n : \bullet A_n$ with $\Gamma \equiv x_1 : A_1, \ldots, x_n : A_n$. In addition, $\lambda^{\mu\bullet}$ contains recursive types $\mu X.A$ in its type expressions, but the occurrences of X in A must ensue within the scope of some occurrence of $\bullet$ (i.e., the occurrences of X must be *guarded* by $\bullet$).

In this system, Nakano defined the fixed-point operator

$$\xi :\equiv \lambda f.(\lambda x.f(xx))(\lambda x.f(xx)) : (\bullet X \to X) \to X$$

for any type variable X, using the subtyping rule below:

$$\frac{\Gamma \vdash t : A \quad \vdash A \preceq B}{\Gamma \vdash t : B}\ \preceq.$$

Here the relation $\preceq$ is a subtype relation, and this rule states that if A is a subtype of B then a term of type A is also a term of type B. Subtyping judgements such as $\vdash A \preceq B$ are governed by the subtype relation rules of $\lambda^{\mu\bullet}$, which derive subtyping judgements without assigning terms in the judgements. Although the fixed-point operator is definable in $\lambda^{\mu\bullet}$, this system is logically consistent: there exists a type that cannot be inhabited in $\lambda^{\mu\bullet}$. This contrasts with our definition of the fixed-point operator by means of the introduction and elimination rules for the generalised Bullet connective $\bullet^A$; the presence of both the $\bullet^A$-introduction and $\bullet^A$-elimination rules leads to an unrestricted recursive type that entails a logical contradiction. Nakano imposed a guardedness condition on recursive types so as to obtain a version of Bullet connectives (i.e., his modality $\bullet A$) that does not entail a logical contradiction.

[33] Nakano sees programs as functions, and that is why he considers types as specifications (since a specification should be able to explain the way in which inputs and outputs are related). According to our less-demanding interpretation of the notion of type, we leave open the possibility that programs do not necessarily behave as functions. A program is more generally a procedure allowing one to interact with a certain environment.

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

As an example of a recursive program defined using this fixed-point operator, we consider streams of data of type X, which can be presented as

$$\lambda x.\xi(\lambda y.\langle x, y\rangle) : X \to \mu Y.X \times \bullet Y,$$

where $\times$ denotes the usual product type. This stream unfolds as

$$\lambda x.\xi(\lambda y.\langle x, y\rangle) \rhd \lambda x.\zeta\zeta \rhd \lambda x.(\lambda y.\langle x, y\rangle)(\zeta\zeta) \rhd \lambda x.\langle x, \zeta\zeta\rangle \rhd \cdots$$

with $\zeta :\equiv \lambda x_1.(\lambda y.\langle x, y\rangle)(x_1 x_1)$.

Treating a modal type $\bullet A$ as $\bullet^A$, one can derive $(\bullet A \to A) \to A$ in the fragment of $\lambda_1^{\to\bullet}$ and $\lambda_2^{\to\bullet}$ obtained by dropping the $\bullet^A$-elimination rules (below we omit the application of the Var-rule):

$$\dfrac{\dfrac{f : \bullet^A \to A \vdash f : \bullet^A \to A \quad \dfrac{f : \bullet^A \to A, x : \bullet^A \vdash f : \bullet^A \to A \quad f : \bullet^A \to A, x : \bullet^A \vdash x : \bullet^A}{f : \bullet^A \to A, x : \bullet^A \vdash fx : A}\to_E}{f : \bullet^A \to A \vdash \hat{x}^A.fx : \bullet^A}\bullet^A 1_I}{\vdash \lambda f.f(\hat{x}^A.fx) : (\bullet^A \to A) \to A}\to_E \to_I$$

$$\dfrac{f : \bullet^A \to A \vdash f : \bullet^A \to A \quad \dfrac{f : \bullet^A \to A \vdash f : \bullet^A \to A}{f : \bullet^A \to A \vdash \mathsf{fold}^A(f) : \bullet^A}\bullet^A 2_I}{\vdash \lambda f.f(\mathsf{fold}^A(f)) : (\bullet^A \to A) \to A}\to_E \to_I .$$

In contrast to $\lambda^{\bullet}$, this does not allow us to obtain a recursive program, because we lack the $\bullet^A$-elimination rules and therefore cannot compute the terms constructed by the $\bullet^A$-introduction rules.

Nevertheless, we can assign a computational meaning to the $\bullet^A$-introduction rules by interpreting the current fragment of $\lambda_1^{\to\bullet}$ and $\lambda_2^{\to\bullet}$ in $\lambda^{\bullet}$. Since simply typed λ-calculus is a subsystem of $\lambda^{\bullet}$, it suffices to derive the $\bullet^A$-introduction rules in $\lambda^{\bullet}$. Here, we consider only the $\bullet^A$-introduction rule of $\lambda_1^{\to\bullet}$, which is formulated using type variables (see below). First, let $B :\equiv \mu Y.\bullet Y \to X$. We omit the derivations of subtyping judgements used below, as they are immediately derivable using the $\lambda^{\bullet}$-subtype relation rules. In particular, the judgements $\vdash \bullet B \to X \preceq \bullet B$ and $\vdash \bullet B \preceq \bullet(\bullet B \to X)$ are derivable from the facts that $\bullet B \to X \preceq B \preceq \bullet B$ and $B \preceq \bullet B \to X$ hold, respectively. Defining $\chi :\equiv \lambda x.(\lambda y.t)(xx)$, we have

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, y : \bullet X \vdash t : X}{\Gamma, x : \bullet B, y : \bullet X \vdash t : X}\text{Weak.}}{\Gamma, x : \bullet B \vdash \lambda y.t : \bullet X \to X}\to_I \quad \dfrac{\Gamma, x : \bullet B \vdash x : \bullet B \quad \vdash \bullet B \preceq \bullet(\bullet B \to X)}{\Gamma, x : \bullet B \vdash x : \bullet(\bullet B \to X)}\preceq \quad \dfrac{\Gamma, x : \bullet B \vdash x : \bullet B \quad \vdash \bullet B \preceq \bullet\bullet B}{\Gamma, x : \bullet B \vdash x : \bullet\bullet B}\to_E}{\Gamma, x : \bullet B \vdash xx : \bullet X}\preceq}{\Gamma, x : \bullet B \vdash (\lambda y.t)(xx) : X}\to_E}{\Gamma \vdash \lambda x.(\lambda y.t)(xx) : \bullet B \to X}\to_I$$

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash \chi : \bullet B \to X} \quad \dfrac{\dfrac{\vdots}{\Gamma \vdash \chi : \bullet B \to X} \quad \vdash \bullet B \to X \preceq \bullet B}{\Gamma \vdash \chi : \bullet B}\to_E}{\Gamma \vdash \chi\chi : X} \quad \vdash X \preceq \bullet X}{\Gamma \vdash \chi\chi : \bullet X}\preceq .$$

In short, we have derived the $\bullet^A$-introduction rule of $\lambda_1^{\to\bullet}$ as

$$\dfrac{\Gamma, y : \bullet X \vdash t : X}{\Gamma \vdash \chi\chi : \bullet X} .$$

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

Thus, we not only establish the consistency of the $\bullet^A$-introduction rules, but also provide them with a computational meaning in terms of Nakano's fixed-point operator.

However, it is not clear what this argument establishes from the perspective of proof-theoretic semantics. Does the interpretation of the above fragment of $\lambda_1^{\rightarrow\bullet}$ and $\lambda_2^{\rightarrow\bullet}$ in $\lambda^{\mu\bullet}$ provide the $\bullet^A$-introduction rules not only with a computational meaning but also with a proof-conditional meaning? The difficulty lies in the fact that $\lambda^{\mu\bullet}$ includes the $\preceq$-rule, which allows the use of subtyping judgements, and the rules governing subtype relations in [41] are not formulated in terms of introduction and elimination rules. If we consider the inversion principle, it clearly holds for $\rightarrow$ in $\lambda^{\mu\bullet}$, and one can establish the corresponding recovery principle as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash t : \bullet^n(A \rightarrow B)}}{\Gamma, x : \bullet^n A \vdash t : \bullet^n(A \rightarrow B)}\ \text{Weak.} \qquad \Gamma, x : \bullet^n A \vdash x : \bullet^n A}{\Gamma, x : \bullet^n A \vdash tx : \bullet^n B}\ \rightarrow_E}{\Gamma \vdash \lambda x.tx : \bullet^n A \rightarrow \bullet^n B}\ \rightarrow_I \qquad \vdash \bullet^n A \rightarrow \bullet^n B \preceq \bullet^n(A \rightarrow B)}{\Gamma \vdash \lambda x.tx : \bullet^n(A \rightarrow B)}\ \preceq .$$

However, the $\preceq$-rule plays a crucial role in this derivation.

In fact, providing a precise definition of what constitutes an introduction or elimination rule and determining their admissible forms is a non-trivial task. Only a small number of works in the literature address it explicitly—for instance, Prawitz [51], Dummett [19, pp. 256–258], and Martin-Löf [34].[34] The formulations of introduction and elimination rules proposed by Prawitz and Martin-Löf are such that they satisfy the inversion and recovery principles by definition. Consequently, as stated in the Introduction, they meet the necessary conditions for logicality. Because the rules governing the subtype relation do not follow this format, it is difficult to formulate inversion and recovery principles for them, and therefore difficult to assess whether they can be regarded as logical. We cannot pursue this issue further here, as doing so would require engaging in a separate and considerably more ambitious project, namely, an investigation into the general format of introduction and elimination rules and the manner in which this format shapes the formulation of the condition of logicality.

§4. Related work. To better appreciate the scope and novelty of our work, we compare it with related studies.

In a standard textbook on types and programming languages, such as that by Pierce [48], a fixed-point operator is defined by explicitly considering the recursive type $\mu X.X \rightarrow A$, which he calls an 'infinite type', precisely because of the circularity discussed in §3.2. The remark by Pierce quoted in this section directly concerns such an infinite type. Pierce's focus in [48] is on the foundations of programming languages rather than on proof-theoretic semantics, it is therefore not surprising that connectives such as Bullet connectives are not studied explicitly there.

[34] In [34], the introduction and elimination formats are given in connection with inductive types: the former correspond to the way in which the elements of a type are constructed, whereas the latter guarantee that there is no other way of constructing elements of this type; in other words, the elimination rules express the exhaustivity of the modes of construction expressed by the introduction rules (see [71, p. 598]).

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

Schroeder-Heister [57] and Tranchini [67], using their respective notations, considered the Bullet connective $\bullet$ in typed λ-calculus and obtained a looping combinator, that is, a closed term whose reduction does not terminate. Tranchini further explained how such a looping combinator can capture the notion of partial functions. These works, however, do not discuss generalised Bullet connectives, by means of which we interpreted the general fixed-point operator fix and type-free λ-calculus.

By defining the $\bullet$ connective as $\bullet :\equiv \mu X.(X \to \bot)$, Dyckhoff [20] observed that this definition is not positive because of the negative occurrence of X in $X \to \bot$. He then argued that this negative occurrence of X gives rise to the circularity of $\bullet$, which we discussed in §§2.2 and 3.2. However, as in [57, 67], generalised Bullet connectives are not considered.

Building on Dyckhoff's observation, Pezlar [45] argued from the perspective of Martin-Löf type theory that what is paradoxical about the $\bullet$ connective is its formation rule: it cannot be justified that the $\bullet$ connective constitutes a proposition. This aligns with our claim that the $\bullet$ connective is not meaningful from a proof-theoretic semantics perspective. Although Pezlar also mentioned generalised Bullet connectives, they were not investigated further in that work; in particular, they were not analysed in terms of recursive types.

Petrolo and Pistone [44] obtained a polymorphic fixed-point operator by exploiting Curry's paradox. Their argument is similar to ours in Appendix 1. Moreover, they considered the notion of isomorphism between types, which is a special case of provable isomorphism, and showed that in $\lambda_2^{\to\bullet}$, $\bullet^A$ and $\bullet^A \to A$ are isomorphic in their sense (and hence in the sense of provable isomorphism as well). However, Petrolo and Pistone did not show that type-free λ-calculus can be interpreted using a variant of the Bullet connectives.

In addition, Klev [27] analysed Curry's paradox from the perspective of Martin-Löf's explanation of intuitionistic type theory. According to Klev, the operators that give rise to Curry's paradox are not operators for constructing inductive types in Martin-Löf type theory.[35] Consequently, he argued that it is impossible to assign a Curry–Howard interpretation to the derivation corresponding to Curry's paradox. This supports our own view that the rules for Bullet connectives do not admit a Curry–Howard interpretation. Moreover, Klev maintains, as we do, that there exist types that do not correspond to propositions. Our analysis goes further by showing that, using Prawitz's notion of validity and the notion of non-well-founded recursive types, one can give an explicit explanation of why Bullet connectives do not admit a Curry–Howard interpretation.[36]

In a logical approach to hardware verification, Fairtlough and Mendler introduced the lax modality $\bigcirc$ into intuitionistic propositional logic [21], and the generalised Bullet connective $\bullet^A$ can also be understood in terms of this modality.[37] The lax modality represents the notion of correctness up to constraints: intuitively, $\bigcirc A$ means

[35] Note that inductive types are well-founded, that is, functions defined by means of their induction principles are terminating, whereas recursive types are not necessarily so. For the distinction between inductive and recursive types, see, e.g., [26].

[36] Nakano's guarded recursion is a method of taming recursion to prevent it from having a non-well-founded recursion entailing logical inconsistencies. Thus, Nakano's guarded recursion is, in a sense, a middle way between the inductive and recursive types.

[37] We thank the anonymous referee for pointing out the link between $\bullet^A$ and $\bigcirc A$.

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

that A holds under some constraint. The axiom schemes of the lax modality are as follows:

$$A \to \bigcirc A \qquad \bigcirc \bigcirc A \to \bigcirc A \qquad \bigcirc A \wedge \bigcirc B \to \bigcirc(A \wedge B).$$

These schemes can be derived using the $\bullet^A$-rules by treating $\bigcirc A$ as $\bullet^A$. In $\lambda_1^{\to\bullet}$, the first scheme is derivable by a vacuous application of the $\bullet^A$-introduction rule, the second by applying the elimination rule for $\bullet^{\bullet^A}$, and the third by using both the $\bullet^A$-introduction and elimination rules. Of course, the presence of both the $\bullet^A$-introduction and elimination rules allows one to derive any formula via a fixed-point argument; however, the third scheme is provable directly, that is, without appealing to this argument.

With respect to our contributions, none of the works discussed in this section analyse Bullet connectives using either Prawitz's notion of validity or Nakano's modality.

§5. Concluding remarks. This paper analysed a family of 0-ary connectives called Bullet connectives from a proof-theoretic and a computational perspective. First, we showed that such connectives do not meet some basic requirements imposed by proof-theoretic semantics. In particular, the inference rules of these connectives are not compatible with Prawitz's notion of validity of proofs, which is defined inductively: the validity of a proof having a complex proposition as conclusion must rest on the validity of proofs having less complex conclusions to avoid circularities and make the definition well-founded. Given that this is not the case for the Bullet connectives, they cannot be considered meaningful from the point of view of proof-theoretic semantics. A meaningful connective is, instead, a connective allowing one to define propositions, namely, linguistic entities for which we can define a (well-founded) notion of valid proof.

However, the fact that the Bullet connectives are not meaningful from the point of view of proof-theoretic semantics does not imply that they are not meaningful at all. We argued that they are still meaningful from a computational point of view. We defined an extension of simply typed λ-calculus allowing us to decorate the rules of the Bullet connectives with functional terms. Based on the Curry–Howard correspondence, this allows us to look at the Bullet connectives as types, and at the corresponding terms as programs. We showed, in particular, that the extended system can simulate the general fixed-point operator. Moreover, by defining some variants of the Bullet connectives, we also showed that the Bullet connectives are instances of recursive types, and that one of these variants enables us to interpret type-free λ-calculus, that is, to obtain a framework suitable for defining a Turing-complete model of computation. Finally, we derived the type of Nakano's fixed-point operator in simply typed λ-calculus with the $\bullet^A$-introduction rule only, and then interpreted the $\bullet^A$-introduction rule in Nakano's type system.

Among the programs definable by using the rules for the Bullet connectives, there are non-terminating ones. These programs do not correspond to valid proofs in the sense of Prawitz. Hence, the types associated with these programs cannot be considered as propositions. This contrasts with the idea that our way of decorating the rules of the Bullet connectives with terms follows a genuine Curry–Howard correspondence. What is imposed by the Curry–Howard correspondence is indeed a very strict notion of type, where types serve to ensure the termination of programs (i.e., the totality of

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

the function corresponding to such programs) and thus to check their correctness (i.e., to verify whether the right value/output has been reached). This notion of type comes from logical and foundational frameworks, such as Martin-Löf type theory [37],[38] and is of a conceptual significance, as it allows us to connect constructive logic and mathematics with computer programming [42].

However, in computer science practice, there are some kinds of programs whose characteristic feature is exactly that of being non-terminating. Operating systems, for example, must run continuously to make it possible to call another program at any time. Computer viruses are also conceived to indefinitely reproduce themselves; indeed, they have been analysed in terms of fixed-point operators (see [8, 9, 30]), such as those we defined using the Bullet connectives. This kind of non-terminating program is typable in the sense that we can associate a formal description of its behaviour: to act in a non-well-founded or circular way.

Our analysis shows that it is possible to develop a computational account of Bullet connectives, but it requires distinguishing the notion of type from that of proposition, thus abandoning one of the characteristic features of the Curry–Howard perspective. The notion of type which we adopt here comes from computer science practice, and it is broader than the notion of proposition. In fact, as shown by Martini [31, 32], the notion of type used in computer science is neither univocal nor stable. Certainly, there is a (restrictive) use of the notion of type that comes from the logic tradition, and this notion is the one found in the definition of types as propositions that characterises the Curry–Howard correspondence. However, in computer science practice, there is another, less restrictive (and thus broader) notion of type, which considers types as ways of classifying programs in an abstract manner (i.e., independently of a particular representation or implementation of data[39]) by indicating the behaviour of such programs and how they act. Treating types in this way does not require them to be hierarchically structured nor well-founded: it is only required that they are completely defined by the operations fixed by the rules characterising them.[40]

Paradoxical connectives that do not well behave from a proof-theoretic and logical point of view are not necessarily devoid of any interest. They can still be computationally meaningful. The possibility of assigning them a computational content does not have to pass through a full-fledged Curry–Howard correspondence, as the notion of computation has to satisfy fewer constraints than those of logic.

§A. Appendix: Interpretation of the general fixed-point operator in the system $\lambda_2^{\rightarrow\bullet}$. In this appendix, we show that the system $\lambda_2^{\rightarrow\bullet}$ is able to derive the typing and reduction rules for the general fixed-point operator fix. Recall that $\lambda_2^{\rightarrow\bullet}$ was obtained by extending simply typed λ-calculus with the second version of the generalised Bullet connective. First, we have a $\lambda_2^{\rightarrow\bullet}$-derivation as the one given in Figure A1.

[38] Here, we consider the intensional version of Martin-Löf's type theory, because the extensional version with a universe type does not allow the termination of typed terms (cf. [42]). For the distinction between the two versions, see [42, 68].

[39] This is why they are usually called *abstract data types*.

[40] This notion of type is very similar to the notion of abstract data type presented by Liskov and Zilles [29, p. 51]: 'An abstract data type defines a class of abstract objects which is completely characterized by the operations available on those objects. This means that an abstract data type can be defined by defining the characterizing operations for that type'.

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

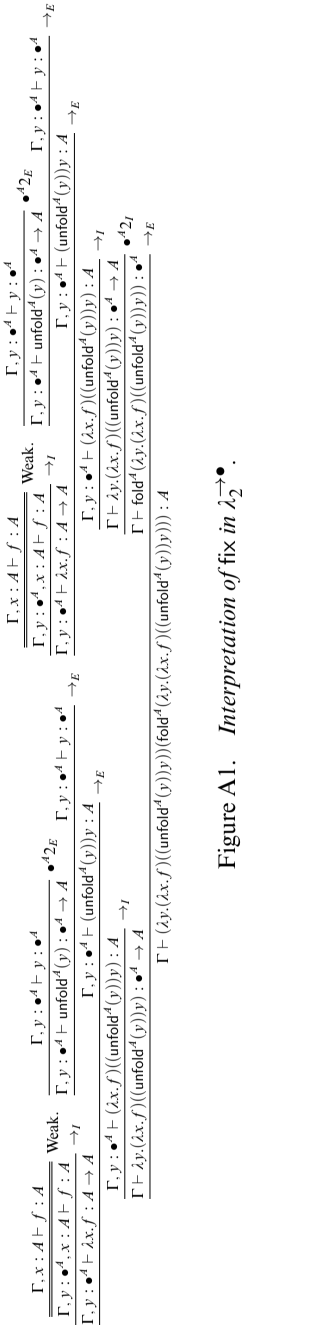


Figure A1. *Interpretation of* fix *in* $\lambda_2^{\rightarrow \bullet}$.

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

If we define $\mathsf{fix}((x)f)$ as

$$\mathsf{fix}((x)f) :\equiv (\lambda y.(\lambda x.f)((\mathsf{unfold}^A(y))y))(\mathsf{fold}^A(\lambda y.(\lambda x.f)((\mathsf{unfold}^A(y))y))),$$

then we obtain the typing rule for fix from this derivation. Moreover, its reduction rule is derivable as well by using the reduction rules of $\lambda_2^{\to\bullet}$:

$$\begin{aligned}\mathsf{fix}((x)f) &\equiv (\lambda y.(\lambda x.f)((\mathsf{unfold}^A(y))y))t\\ &\rhd (\lambda x.f)((\mathsf{unfold}^A(\mathsf{fold}^A(\lambda y.(\lambda x.f)((\mathsf{unfold}^A(y))y))))t)\\ &\rhd (\lambda x.f)((\lambda y.(\lambda x.f)((\mathsf{unfold}^A(y))y))t)\\ &\rhd f[(\lambda y.(\lambda x.f)((\mathsf{unfold}^A(y))y))t/x]\\ &\equiv f[\mathsf{fix}((x)f)/x],\end{aligned}$$

where $t :\equiv \mathsf{fold}^A(\lambda y.(\lambda x.f)((\mathsf{unfold}^A(y))y))$.

Acknowledgments. We are extremely grateful to the anonymous referee for their generous comments, which greatly helped to improve the paper. A first version of this paper was presented in June 2021 at the EXPRESS-IHPST (online) workshop *Truth, Proof and Communication*, co-organised by Luca Incurvati and Francesca Poggiolesi. We thank them for giving us the opportunity to present this talk and for their valuable comments and suggestions, which helped us to further develop our ideas. We also thank Michele Contente and Ansten Klev for their interest in our paper and for the very useful discussions about it.

Funding. The work of the first author was partially supported by the ANR project *GoA—The Geometry of Algorithms* (ANR-20-CE27-0004).

BIBLIOGRAPHY

[1] Altenkirch, T. (1998). *Logical relations and inductive/coinductive types*. In Gottlob, G., Grandjean, E., & Seyr, K., editors. *Computer Science Logic, 12th International Workshop, CSL '98, Annual Conference of the EACSL, Brno, Czech Republic, August 24-28, 1998, Proceedings*, Lecture Notes in Computer Science, 1584. Berlin: Springer, pp. 343–354.

[2] Amadio, R. M., & Curien, P.-L. (1998). *Domains and Lambda-Calculi*, Cambridge Tracts in Theoretical Computer Science. Cambridge: Cambridge University Press.

[3] van Atten, M. (2018). The creating subject, the Brouwer-Kripke schema, and infinite proofs. *Indagationes Mathematicae*, **29**(6), 1565–1636.

[4] Barendregt, H. P. (1984). *The Lambda Calculus: Its Syntax and Semantics* (revised edition). Amsterdam: North-Holland.

[5] ———. (1992). *Lambda calculi with types*. In Abramsky, S., Gabbay, D. M., & Maibaum, S. E., editors. *Handbook of Logic in Computer Science*, Vol. 2. New York: Oxford University Press, Inc., pp. 117–309.

[6] van Benthem, J. F. A. K. (1978). Four paradoxes. *Journal of Philosophical Logic*, **7**(1), 49–72.

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

[7] Blanqui, F., Jouannaud, J.-P., & Okada, M. (2002). Inductive-data-type systems. *Theoretical Computer Science*, **272**(1–2), 41–68.

[8] Bonfante, G., Kaczmarek, M., & Marion, J.-Y. (2006). On abstract computer virology from a recursion theoretic perspective. *Journal in Computer Virology*, **1**(3–4), 45–54.

[9] ———. (2007). *A classification of viruses through recursion theorems*. In Cooper, S. B., Löwe, B., & Sorbi, A., editors. *Computation and Logic in the Real World, Third Conference on Computability in Europe, CiE 2007, Siena, Italy, June 18–23, 2007, Proceedings*, Lecture Notes in Computer Science, 4497. Berlin: Springer, pp. 73–82.

[10] Bruce, K. B., & Longo, G. (1985). *Provable isomorphisms and domain equations in models of typed languages (preliminary version)*. In Sedgewick, R., editors. *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6–8, 1985, Providence, Rhode Island, USA*. New York: ACM, pp. 263–272.

[11] Cardelli, L., & Wegner, P. (1985). On understanding types, data abstraction, and polymorphism. *ACM Computing Surveys*, **17**(4), 471–522.

[12] Ceragioli, L. (2020). *Pluralism in proof-theoretic semantics*. PhD Thesis, Università degli Studi di Firenze.

[13] Constable, R. L., & Mendler, N. P. (1985). *Recursive definitions in type theory*. In Parikh, R., editor. *Logics of Programs, Conference, Brooklyn College, New York, NY, USA, June 17-19, 1985, Proceedings*, Lecture Notes in Computer Science, 193. Berlin: Springer, pp. 61–78.

[14] Constable, R. L., & Smith, S. F. (1987). Partial objects in constructive type theory. In *Proceedings of the Symposium on Logic in Computer Science (LICS '87), Ithaca, New York, USA, June 22-25, 1987*, Washington, D.C.: Computer Society Press of the IEEE, pp. 183–193.

[15] Coquand, T., & Paulin, C. (1990). *Inductively defined types*. In Martin-Löf, P., & Mints, G., editors. *COLOG-88. COLOG 1988*, Lecture Notes in Computer Science, 417. Berlin: Springer, pp. 50–66.

[16] di Cosmo, R. (2005). A short survey of isomorphisms of types. *Mathematical Structures in Computer Science*, **15**(5), 825–838.

[17] Curry, H. B., & Feys, R. (1974). *Combinatory Logic. Volume I. With Two Sections by William Craig*. Amsterdam: North-Holland.

[18] Dummett, M. (1978). *Frege's distinction between sense and reference*. In *Truth and Other Enigmas*. London: Duckworth, pp. 116–144.

[19] ———. (1991). *The Logical Basis of Metaphysics*. London: Duckworth.

[20] Dyckhoff, R. (2016). *Some remarks on proof-theoretic semantics*. In Piecha, T., & Schroeder-Heister, P., editors. *Advances in Proof-Theoretic Semantics* Trends in Logic, 43. Berlin: Springer, pp. 79–93.

[21] Fairtlough, M., & Mendler, M. (1997). Propositional lax logic. *Information and Computation*, **137**(1), 1–33.

[22] Francez, N. (2015). *Proof-Theoretic Semantics*. London: College Publications.

[23] Francez, N., & Dyckhoff, R. (2012). A note on harmony. *Journal of Philosophical Logic*, **41**(3), 613–628.

[24] Gunter, C. A. (1992). *Semantics of Programming Languages*, Foundations of Computing. Cambridge (Mass.): The MIT Press.

[25] Hancock, P., & Martin-Löf, P. (1975). Syntax and semantics of the language of primitive recursive functions. Technical Report, Tech. Rep. 3. Stockholm, Sweden: University of Stockholm.

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

[26] Harper, R. (2016). *Practical Foundations for Programming Languages* (second edition). New York: Cambridge University Press.

[27] Klev, A. (2025). A type-theoretical Curry paradox and its solution. *The Philosophical Quarterly*, **75**(2), 763–774.

[28] Landin, P. J. (1964). The mechanical evaluation of expressions. *The Computer Journal*, **6**(4), 308–320.

[29] Liskov, B. H., & Zilles, S. N. 1974. *Programming with abstract data types*. In Leavenworth, B. M. editors. *Proceedings of the ACM SIGPLAN Symposium on Very High Level Languages, Santa Monica, California, USA, March 28-29, 1974*. New York: ACM, pp. 50–59.

[30] Marion, J.-Y. (2012). From Turing machines to computer viruses. *Philosophical Transactions of the Royal Society A*, **370**, 3319–3339.

[31] Martini, S. (2015). *Several types of types in programming languages*. In Gadducci, F., & Tavosanis, M., editors. *History and Philosophy of Computing—Third International Conference, HaPoC 2015, Pisa, Italy, October 8-11, 2015, Revised Selected Papers*, IFIP Advances in Information and Communication Technology, 487. Berlin: Springer, pp. 216–227.

[32] ———. (2016). *Types in programming languages, between modelling, abstraction, and correctness—Extended abstract*. In Beckmann, A., Bienvenu, L., & Jonoska, N., editors. *Pursuit of the Universal—12th Conference on Computability in Europe, CiE 2016, Paris, France, June 27—July 1, 2016, Proceedings*, Lecture Notes in Computer Science, 9709. Berlin: Springer, pp. 164–169.

[33] Martin-Löf, P. (1982). *Constructive mathematics and computer programming*. In Jonathan Cohen, L., Łoś, J., Pfeiffer, H., & Podewski, K.-P., editors. *Logic, Methodology and Philosophy of Science VI*, Studies in Logic and the Foundations of Mathematics, 104. Amsterdam: Elsevier, pp. 153–175.

[34] ———. (1984). *Intuitionistic Type Theory: Notes by Giovanni Sambin of a Series of Lectures Given in Padua, June 1980*. Naples: Bibliopolis.

[35] ———. (1990). *Mathematics of infinity*. In Martin-Löf, P., & Mints, G., editors. *Colog-88, International Conference on Computer Logic Tallinn, USSR, December 12–16, 1988 Proceedings, (Berlin, Heidelberg)*, Lecture Notes in Computer Science, 417. Berlin: Springer, pp. 146–197.

[36] ———. (1996). On the meanings of the logical constants and the justifications of the logical laws. *Nordic Journal of Philosophical Logic*, **1**(1), 11–60.

[37] ———. (1998). *An intuitionistic theory of types*. In Sambin, G., & Smith, J. M., editors. *Twenty-Five Years of Constructive Type Theory*, Oxford Logic Guides, 36. Oxford: Clarendon Press, pp. 127–172.

[38] Matthes, R. (1998). *Monotone fixed-point types and strong normalization*. In Gottlob, G., Grandjean, E., & Seyr, K., editors. *Computer Science Logic, 12th International Workshop, CSL '98, Annual Conference of the EACSL, Brno, Czech Republic, August 24-28, 1998, Proceedings*, Lecture Notes in Computer Science, 1584. Berlin: Springer, pp. 298–312.

[39] Moschovakis, Y. N. (1993). *Sense and denotation as algorithm and value*. In Oikkonen, J., & Väänänen, J., editors. *Logic Colloquium '90: ASL Summer Meeting in Helsinki*, Lecture Notes in Logic. Berlin: Springer-Verlag, pp. 210–249.

[40] Murzi, J., & Steinberger, F. (2017). *Inferentialism*. In Hale, B., Wright, C., & Miller, A., editors. *A Companion to the Philosophy of Language* (second edition). Oxford: Wiley-Blackwell, pp. 197–224.

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

[41] Nakano, H. (2000). A modality for recursion. In *15th Annual IEEE Symposium on Logic in Computer Science, Santa Barbara, California, USA, June 26–29*, Vol. 2000. Los Alamitos (Calif.): IEEE Computer Society, pp. 255–266.

[42] Nordström, B., Petersson, K., & Smith, J. M. (1990). *Programming in Martin-Löf's Type Theory. An Introduction*. Oxford: Oxford University Press.

[43] Palmgren, E. (1991). A construction of type: Type in Martin-Löf's partial type theory with one universe. *The Journal of Symbolic Logic*, **56**(3), 1012–1015.

[44] Petrolo, M., & Pistone, P. (2019). On paradoxes in normal form. *Topoi*, **38**, 605–617.

[45] Pezlar, I. (2021). A note on paradoxical propositions from an inferential point of view. In Blicha, M., & Sedlár, I., editors. *The Logica Yearbook 2020*. London: College Publications, pp. 183–199.

[46] Pfenning, F., & Davies, R. (2001). A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, **11**(4), 511–540.

[47] Piecha, T., de Campos Sanz, W., & Schroeder-Heister, P. (2015). Failure of completeness in proof-theoretic semantics. *Journal of Philosophical Logic*, **44**, 321–335.

[48] Pierce, B. C. (2002). *Types and Programming Languages*. Cambridge: The MIT Press.

[49] Prawitz, D. (1965). *Natural Deduction: A Proof-Theoretical Study*. Stockholm: Almqvist & Wiksell.

[50] ———. (1971). *Ideas and results in proof theory*. In Fenstad, J. E., editors. *Proceedings of the Second Scandinavian Logic Symposium*, Studies in Logic and the Foundations of Mathematics, 63. Amsterdam: Elsevier, pp. 235–307.

[51] ———. (1973). *Towards a foundation of a general proof theory*. In Suppes, P., Henkin, L., Joja, A., & Moisil, G. C., editors. *Proceedings of the Fourth International Congress for Logic, Methodology and Philosophy of Science, Bucharest, 1971*, Studies in Logic and the Foundations of Mathematics, 74. Amsterdam: Elsevier, pp. 225–250.

[52] ———. (1977). Meaning and proofs: On the conflict between classical and intuitionistic logic. *Theoria*, **43**(1), 2–40.

[53] de Queiroz, R. J. G. B., de Oliveira, A. G., & Gabbay, D. M. (2011). *The Functional Interpretation of Logical Deduction*. Singapore: World Scientific.

[54] Read, S. (2000). Harmony and autonomy in classical logic. *Journal of Philosophical Logic*, **29**(2), 123–154.

[55] ———. (2010). General-elimination harmony and the meaning of the logical constants. *Journal of Philosophical Logic*, **39**(5), 557–576.

[56] Schroeder-Heister, P. (2006). Validity concepts in proof-theoretic semantics. *Synthese*, **148**(3), 525–571.

[57] ———. (2012). Proof-theoretic semantics, self-contradiction, and the format of deductive reasoning. *Topoi*, **31**, 77–85.

[58] ———. (2014). The calculus of higher-level rules, propositional quantification, and the foundational approach to proof-theoretic harmony. *Studia Logica*, **102**(6), 1185–1216.

[59] ———. (2024). *Proof-theoretic semantics*. In Zalta, E. N., & Nodelman, U., editors. *The Stanford Encyclopedia of Philosophy* (Summer 2024 edition). Metaphysics Research Lab, Stanford University. URL: https://plato.stanford.edu/archives/sum2024/entries/proof-theoretic-semantics/.

[60] Schroeder-Heister, P., & Tranchini, L. (2017). Ekman's paradox. *Notre Dame Journal of Formal Logic*, **58**(4), 567–581.

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press

[61] Scott, D. S. (1976). Data types as lattices. *SIAM Journal on Computing*, **5**(3), 522–587.

[62] ———. (1982). *Domains for denotational semantics*. In Nielsen, M., & Schmidt, E. M., editors. *Automata, Languages and Programming, 9th Colloquium, Aarhus, Denmark, July 12–16, 1982, Proceedings*, Lecture Notes in Computer Science, 140. Berlin: Springer, pp. 577–613.

[63] Sørensen, M. H., & Urzyczyn, P. (2006). *Lectures on the Curry-Howard Isomorphism*, Studies in Logic and the Foundations of Mathematics, 149. Amsterdam: Elsevier.

[64] Szabo, M. E. (1969). *The Collected Papers of Gerhard Gentzen*. Amsterdam: North-Holland.

[65] Tennant, N. (1982). Proof and paradox. *Dialectica*, **36**, 265–296.

[66] Tichý, P. (1988). *The Foundations of Frege's Logic*. Berlin: De Gruyter.

[67] Tranchini, L. (2019). Proof, meaning and paradox: Some remarks. *Topoi*, **38**, 591–603.

[68] Tranchini, L., & Naibo, A. (2024). *Judgmental and definitional equality from a Fregean perspective*. In Klev, A., editor. *The Architecture and Archaeology of Modern Logic: Studies Dedicated to Göran Sundholm*. Berlin: Springer Nature, pp. 145–179.

[69] Troelstra, A. S., & Schwichtenberg, H. (2000). *Basic Proof Theory* (second edition). Cambridge: Cambridge University Press.

[70] Troelstra, A. S., & van Dalen, D. (1988). *Constructivism in Mathematics: An Introduction*, Vol. 1, Studies in Logic and the Foundations of Mathematics, 121. Amsterdam: Elsevier.

[71] ———. (1988). *Constructivism in Mathematics: An Introduction*, Vol. 2, Studies in Logic and the Foundations of Mathematics, 123. Amsterdam: Elsevier.

[72] Turner, R. (1991). *Constructive Foundations for Functional Languages*. London: McGraw-Hill.

[73] The Univalent Foundations Program. (2013). *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study. https://homotopytypetheory.org/book.

[74] Wadsworth, C. P. (1971). *Semantics and pragmatics of the lambda-calculus*. PhD Thesis, University of Oxford.

IHPST (UMR 8590)
UNIVERSITÉ PARIS 1 PANTHÉON-SORBONNE
PARIS, FRANCE
E-mail: alberto.naibo@univ-paris1.fr

AOMORI UNIVERSITY
AOMORI CITY AOMORI 030-0943, JAPAN
E-mail: y.takahashi@aomori-u.ac.jp

https://doi.org/10.1017/S1755020326101099 Published online by Cambridge University Press