About Opposition and Duality in Paraconsistent Type Theory

Juan C. Agudelo-Agudelo
Universidad de Antioquia
Medellín, Colombia
juan.agudelo9@udea.edu.co

Andrés Sicard-Ramírez
Universidad EAFIT
Medellín, Colombia
asr@eafit.edu.co

A paraconsistent type theory (an extension of a fragment of intuitionistic type theory by adding 'opposite types') is here extended by adding co-function types. It is shown that, in the extended paraconsistent type system, the opposite type constructor can be viewed as an involution operation that transforms each type into its dual type. Moreover, intuitive interpretations of opposite and co-function types under different interpretations of types are discussed.

Introduction

A *paraconsistent type theory*, called here PTT_0, was introduced by the authors by extending ITT (the $\to, \times, +, \Pi, \Sigma$ intensional fragment of intuitionistic type theory, as presented in [9], with two universes U_0 and U_1) with the addition of *opposite types* in [1]. In PTT_0, for each type A, there is an opposite type $\overline{A}$. The introduction and elimination rules for opposite types were defined for each one of the type constructors (including the opposite type constructor itself), and such rules were based on the rules for *constructible falsity* [12, 2, 8]. A propositions-as-types correspondence between $\mathsf{PL}_0^{\mathsf{S}}$ (a many-sorted version of the *refutability calculus*–introduced by López-Escobar in [8]–presented in natural deduction style) and PTT_0 was proven.[1] Under such propositions-as-types correspondence, the opposite type constructor corresponds to negation in $\mathsf{PL}_0^{\mathsf{S}}$, the correspondence for the other type constructors are the same than for ITT with respect to intuitionistic logic.

Differently from how it is done in intuitionist type theory, where negation is formalised by means of the function type and the empty type, negation in PTT_0 is formalised by the primitive opposite type constructor, without need of the empty type. Under such formalisation, an inhabitant of a type $\overline{A}$ can be understood as a proof term for $\neg A$ or as a 'refutation term' for A, and proofs and refutations are treated in a symmetric and constructive way.

As $\mathsf{PL}_0^{\mathsf{S}}$ is a paraconsistent logic (i.e. for some set of formulae Δ and some formulae A and B we have that $\Delta \vdash_{\mathsf{PL}_0^{\mathsf{S}}} A$, $\Delta \vdash_{\mathsf{PL}_0^{\mathsf{S}}} \neg A$ and $\Delta \nvdash_{\mathsf{PL}_0^{\mathsf{S}}} B$), the propositions-as-types correspondence with PTT_0 leads to the existence of logically contradictory but not trivial contexts (i.e. contexts Γ for which there exist a type A and terms t and s such that $\Gamma \vdash_{\mathsf{PTT}_0} t : A$ and $\Gamma \vdash_{\mathsf{PTT}_0} r : \overline{A}$, but $\Gamma \nvdash_{\mathsf{PTT}_0} s : B$, for some type B and every term s). Because of that, PTT_0 is considered a 'paraconsistent type theory' (for details see [1]).

We wrote the following observation in the section of concluding remarks and future work of [1]:

> The opposite type constructor can be viewed as an operation that transforms types into their 'duals'. Under a logical (or propositions-as-types) interpretation, the duality is between truth and falsity of propositions (since the habitation of a type $\overline{A}$, interpreted as a proposition, can be understood as A is false). If we interpret types as problems (or tasks), the duality is between solvability and unsolvability of the respective problems. A more difficult understanding of the duality is when types are interpreted as sets. However, also under the

[1] The logic $\mathsf{PL}_0^{\mathsf{S}}$ is denoted in [1] by PL_{S}.

Mauricio Ayala-Rincón, Eduardo Bonelli (Eds.): 16th Logical and Semantic Frameworks with Applications (LSFA 2021)
EPTCS 357, 2022, pp. 25–37, doi:10.4204/EPTCS.357.3

set interpretation, opposite types can be viewed as an operation that establishes a duality between some set operations, although it is difficult to understand in what sense this duality occurs.

Despite such observation, the notion of duality in PTT_0 was neither rigorously defined nor discussed in depth in [1] and this is just the aim of this article.

In PL_0^S, the logical constants $\supset, \wedge, \vee, \forall$ and $\exists$ have the same introduction and elimination rules than in intuitionistic logic, but negation is a primitive logical constant, not definable in terms of the other logical constants, and its behaviour is established by defining introduction and elimination rules for the negation of each one of the logical constants (including negation itself). Moreover, equivalence $\Leftrightarrow$ can be defined in the usual way (i.e. $A \Leftrightarrow B \stackrel{\text{def}}{=} (A \supset B) \wedge (B \supset A)$), but substitution by equivalents fails. However, a *strong equivalence* $\Leftrightarrow_\mathsf{s}$ can be defined by $A \Leftrightarrow_\mathsf{s} B \stackrel{\text{def}}{=} (A \Leftrightarrow B) \wedge (\neg A \Leftrightarrow \neg B)$ and substitution by strong equivalents is valid. The following strong equivalences are derivable in PL_0^S (where superscripts on quantified variables denote their sorts):

$$\neg(A \wedge B) \Leftrightarrow_\mathsf{s} \neg A \vee \neg B, \qquad \neg(\forall x^s)A \Leftrightarrow_\mathsf{s} (\exists x^s)\neg A,$$
$$\neg(A \vee B) \Leftrightarrow_\mathsf{s} \neg A \wedge \neg B, \qquad \neg(\exists x^s)A \Leftrightarrow_\mathsf{s} (\forall x^s)\neg A, \qquad \neg\neg A \Leftrightarrow_\mathsf{s} A.$$

These strong equivalences show that in PL_0^S the negation is an involutive operation under which there is duality between $\wedge$ and $\vee$, and also between $\forall$ and $\exists$. It is important to highlight here that, although in PL_0^S the De Morgan's laws, the 'classical' equivalences for the negation of quantifiers and the double negation law are valid, this does not make this logic collapse to classical logic. There are many formulae valid in classic logic which are not valid in PL_0^S including $A \vee \neg A$, $A \supset (\neg A \supset B)$ and $\neg(A \wedge \neg A)$. The non-validity of this formulae can be proven by using the Kripke style semantics provided for López-Escobar's refutability calculus in [8] (which can be naturally adapted to PL_0^S).

For implication, although $\vdash_{\mathsf{PL}_0^\mathsf{S}} \neg(A \supset B) \Leftrightarrow (A \wedge \neg B)$, we have that $\nvdash_{\mathsf{PL}_0^\mathsf{S}} \neg\neg(A \supset B) \Leftrightarrow \neg(A \wedge \neg B)$, thus $\nvdash_{\mathsf{PL}_0^\mathsf{S}} \neg(A \supset B) \Leftrightarrow_\mathsf{s} (A \wedge \neg B)$. Consequently, if we require that the dual of a formula be strongly equivalent to its negation, so that the negation of the formula and its dual be inter-substitutable (which seems to be a reasonable requirement), then $A \wedge \neg B$ cannot be considered a dual of $A \supset B$ in PL_0^S. Moreover, under the previous requirements, there is no dual logical constant to $\supset$ in PL_0^S, but in order to every logical constant have its dual, the logic PL_0^S is extended in Section 2 by including a co-implication logical constant whose rules are taking from [14] (where a kind of bi-intuitionistic logic, named $\mathsf{2Int}$, is defined by dualising the natural deduction rules of intuitionistic propositional logic). A rigorous definition of dual logical constants is given in Definition 3.1, and under such definition the dualities previously mentioned are justified.

As the introduction and elimination rules for opposite types in PTT_0 are based on the rules of the (constructible) negation of PL_0^S, the opposite type constructor works as an involutive operation under which there is duality between $\times$ and $+$, and between Π and Σ. As in PL_0^S there is no dual type constructor to $\rightarrow$, this type system is extended in Section 2 by adding a new constructor of 'co-function types', whose rules are based on the rules for co-implication in the extension of PL_0^S. The system obtained is called PTT_1 and one of its characteristics is that every type constructor have a dual, which allows a total symmetric formalisation of proofs and refutations. A rigorous definition of dual type constructors is given in Definition 3.3, and under such definition the dualities previously mentioned are justified.

This article is structured as follows: The type system PTT_0 is described in Section 1. In Section 2, PTT_0 is extended to PTT_1. In Section 3, it is shown that the opposite type constructor in PTT_1 can be viewed as an involution operation that transforms each type into its dual type, and it is stated a 'principle of duality' in PTT_1. Finally, some observations about the intuitive interpretation of opposite and co-function types in PTT_1 are presented in Section 4. With respect to the interpretation of types as sets, we only provide a vague description of how opposite types can be generally understood. Although a reviewer reasonably suggested that duality is not set-theoretic and we should just forget about this interpretation, we do not want to abandon this possibility for the moment, and we think that the vague idea that we present may shed some light in further developments of this possible interpretation.

1 The type system PTT_0

We shall give a brief description of system PTT_0, for further details see [1]. As we mentioned in the introduction, PTT_0 is introduced by extending ITT (the $\to, \times, +, \Pi, \Sigma$ intensional fragment of intuitionistic type theory with two universes U_0 and U_1) with the addition of *opposite types*. The formation rule for opposite types in PTT_0 is:

$$\frac{A : \mathtt{Set}}{\overline{A} : \mathtt{Set}}\;.$$

Taking into consideration that the introduction and elimination rules for the constructors $\to, \times, +, \Pi, \Sigma$ in ITT are well-known, only the introduction and elimination rules for opposite types in PTT_0 are described in Table 1 (where the term constructors are the same of ITT as presented in [9]).

The choice of terms for opposite types in PTT_0 is based on López-Escobar's extension of the so-called BHK-interpretation of intuitionistic logic [8], where the definition of what constitutes a construction to refute a formula is made using the same objects that constitute proofs of formulae in the BHK-interpretation, taking advantage of the duality between logical constants and the assumption that a construction c refutes a formula A iff c proves $\neg A$. For instance, a construction c that refutes a formula $A \vee B$ is a pair $c = (a,b)$, where a and b are constructions that refute A and B, respectively (in an analogous and dual way as a construction c that proves a formula $A \wedge B$ is a pair $c = (a,b)$, where a and b are constructions that prove A and B, respectively). Consequently, the term constructors for opposite types in PTT_0 already exists in ITT and their computation (or equality) rules in PTT_0 are defined just in the same way as in ITT. [2]

Universes U_0 and U_1, with $\mathrm{U}_0 : \mathrm{U}_1$, are introduced in PTT_0 in order to prove a propositions-as-types correspondence between this type system and the logic $\mathsf{PL}_0^{\mathsf{S}}$. Universe U_0 is closed under $\to, \times, +, \Pi, \Sigma$ and opposite types, while U_1 is closed only under $\to$. The only aim of U_1 is to allow the creation of types corresponding to predicates, for which it is enough to have in U_1 the type constructor $\to$. Thus the restriction of U_1 to be closed only under $\to$ is to facilitate the proof of the propositions-as-types correspondence. Consequently, the idea of opposite types is developed in PTT_0 only in universe U_0, although it can be naturally extended to other universes. Using widespread terminology, types in U_0 are called *small types*.

2 Extending PTT_0 with co-function types

The familiar introduction and elimination rules for intuitionistic propositional logic are dualised by employing a primitive notion of *dual proof* by Wansing in [14]. Here, we prefer to use *refutation* instead of

[2] This choice of terms leads to non-uniqueness of types in PTT_0, but this does not seem to be an inconvenience in this type system [1, § 6]. However, several causes of the non-uniqueness of types in PTT_0 are avoided by the inclusion of the equality rules for opposite types in Table 4.

$\dfrac{a : A}{a : \overline{\overline{A}}}$	$\dfrac{a : \overline{\overline{A}}}{a : A}$
$\dfrac{a : A \qquad b : \overline{B}}{(a,b) : \overline{A \to B}}$	$\dfrac{c : \overline{A \to B}}{\mathsf{p}(c) : A} \qquad \dfrac{c : \overline{A \to B}}{\mathsf{q}(c) : \overline{B}}$
$\dfrac{a : \overline{A}}{\mathsf{i}(a) : \overline{A \times B}}$ $\dfrac{b : \overline{B}}{\mathsf{j}(b) : \overline{A \times B}}$	$\dfrac{c : \overline{A \times B} \qquad \begin{matrix}(x : \overline{A}) \\ \vdots \\ d(x) : C(\mathsf{i}(x))\end{matrix} \qquad \begin{matrix}(y : \overline{B}) \\ \vdots \\ e(y) : C(\mathsf{j}(y))\end{matrix}}{\mathsf{D}(c, (x)d(x), (y)e(y)) : C(c)}$
$\dfrac{a : \overline{A} \qquad b : \overline{B}}{(a,b) : \overline{A + B}}$	$\dfrac{c : \overline{A + B}}{\mathsf{p}(c) : \overline{A}} \qquad \dfrac{c : \overline{A + B}}{\mathsf{q}(c) : \overline{B}}$
$\dfrac{a : A \qquad b : \overline{B(a)}}{(a,b) : \overline{(\Pi x : A)B(x)}}$	$\dfrac{c : \overline{(\Pi x : A)B(x)} \qquad \begin{matrix}(x : A, y : \overline{B(x)}) \\ \vdots \\ d(x,y) : C((x,y))\end{matrix}}{\mathsf{E}(c, (x,y)d(x,y)) : C(c)}$
$\dfrac{\begin{matrix}(x : A) \\ \vdots \\ b(x) : \overline{B(x)}\end{matrix}}{(\lambda x)b(x) : \overline{(\Sigma x : A)B(x)}}$	$\dfrac{b : \overline{(\Sigma x : A)B(x)} \qquad a : A}{\mathsf{Ap}(b,a) : \overline{B(a)}}$

Table 1: Introduction and elimination rules for opposite types in PTT_0.

dual proof. By using single-line rules for proofs and dotted-line rules for refutations, and using $\supset$ and $\prec$ as the logical constants for implication and co-implication, respectively,[3] the dual rules for $\supset$ are:

$$\dfrac{\begin{matrix}[A] \\ \vdots \\ \overline{B}\end{matrix}}{A \supset B} \quad \rightsquigarrow \quad \dfrac{\begin{matrix}\cdot[A]\cdot \\ \vdots \\ \dot{\overline{B}}\end{matrix}}{B \prec A}\,, \qquad\qquad \dfrac{\overline{A} \qquad \overline{A \supset B}}{B} \quad \rightsquigarrow \quad \dfrac{\dot{\overline{A}} \qquad \dot{\overline{B \prec A}}}{B}\,,$$

where $\rightsquigarrow$ means 'dualises to', $[A]$ denotes the cancellation of the assumption A in the conclusion and $\cdot[A]\cdot$ denotes the cancellation of counter-assumption A (or the assumption of the falsity of A) in the conclusion. Formula $B \prec A$ may be read as 'A co-implies B' or as 'B excludes A' (see [13, Footnote 2]).

As the process of dualisation does not induce rules for the falsification of implications nor for the

[3] In [14], the logical constants for implication and co-implication are $\to$ and $\prec$, respectively, but we shall reserve these symbols for function and co-function types. Moreover, we changed the notation for refutations and counter-assumptions, in order to avoid confusions with our notation for opposite types.

verification of co-implications, these rules are defined in [14] by taking an orthodox stance as follows:[4]

$$\dfrac{\overline{A} \qquad \overline{B}}{A \supset B}\,, \qquad\qquad \dfrac{\overline{A \supset B}}{A}\,, \qquad \dfrac{\overline{A \supset B}}{B}\,,$$

$$\dfrac{\overline{A} \qquad \overline{B}}{B \prec A}\,, \qquad\qquad \dfrac{\overline{B \prec A}}{A}\,, \qquad \dfrac{\overline{B \prec A}}{B}\,.$$

As in $\mathsf{PL}_0^{\mathsf{S}}$ negation represents falsity, and a proof of $\neg A$ can be also understood as a refutation of A, we can extend $\mathsf{PL}_0^{\mathsf{S}}$ with co-implication and define the introduction and elimination rules without using dotted-lined rules for refutations and dotted brackets for counter-assumptions as follows.

$$\dfrac{\neg A \qquad B}{B \prec A}\,, \qquad\qquad \dfrac{B \prec A}{\neg A}\,, \qquad \dfrac{B \prec A}{B}\,,$$

$$\dfrac{\begin{matrix}[\neg A]\\ \vdots \\ \neg B\end{matrix}}{\neg(B \prec A)}\,, \qquad\qquad \dfrac{\neg A \qquad \neg(B \prec A)}{\neg B}\,.$$

The extension of $\mathsf{PL}_0^{\mathsf{S}}$ with co-implication will be called $\mathsf{PL}_1^{\mathsf{S}}$.[5] [6]

---

[4]We changed the order of formulae in the co-implication rules for a better understanding of these rules as duals of implication rules.

[5]In [13], two ways of formalising co-implication are presented. In one way, co-implication is strongly equivalent to negated implication (i.e. $A \prec B \Leftrightarrow_{\mathsf{s}} \neg(A \to B)$). In the other way, co-implication is strongly equivalent to negated contrapose implication (i.e. $A \prec B \Leftrightarrow_{\mathsf{s}} \neg(\neg B \to \neg A)$). In $\mathsf{PL}_0^{\mathsf{S}}$, contraposition is not valid and the two ways of formalising co-implication are not equivalent. Under our formalisation of co-implication in $\mathsf{PL}_1^{\mathsf{S}}$, it is possible to prove that $A \prec B \Leftrightarrow_{\mathsf{s}} \neg(\neg B \to \neg A)$, which shows that the convincing process of dualisation in [14] leads co-implication to behave as negated contraposed implication, instead of behaving as negated implication.

[6]From Negri and von Plato's generalisation of the *inversion principle*, which states that 'whatever follows from the direct grounds for deriving a proposition must follow from that proposition' [11, p. 6], general elimination rules are uniquely determined by each introduction rule. For the case of co-implication, the general elimination rule would be:

$$\dfrac{B \prec A \qquad \begin{matrix}[\neg A, B]\\ \vdots \\ C\end{matrix}}{C}$$

This general elimination rule differs from the two standard elimination rules presented above. However, as in the case of intuitionistic logic, if we change in $\mathsf{PL}_1^{\mathsf{S}}$ the standard elimination rules by the general elimination rules, the system obtained is equivalent with respect to deductibility. In the case of intuitionistic logic, the difference emerges in the correspondence with sequent calculus: Normal derivations in the natural deduction system with general elimination rules can be isomorphically translated into cut-free derivations in the sequent calculus with independent contexts, which is not possible with the standard elimination rules [11, Ch. 8]. As we are not interested in establishing a correspondence of the natural deduction system for $\mathsf{PL}_1^{\mathsf{S}}$ with a sequent calculus, we choose the standard elimination rules for co-implication (and all the other logical constants).

In comparison with linear logic, which contains a fully involutive negation and can be seen 'as a bold attempt to reconcile the beauty and symmetry of the systems for classical logic with the quest for constructive proofs that had led to intuitionistic logic' [4], the same can be said for $\mathsf{PL}_1^{\mathsf{S}}$. However, linear logic is obtained by eliminating the contraction and weakening rules of a sequent calculus for classical logic, allowing the formalisation of two different versions of each logical constant: An *additive*

Now, we shall extend PTT_0 by adding co-function types, whose rules will be based on the co-implication rules. Symbol $\prec$ will be used to denote the co-function type constructor and $B \prec A$ may be read as 'the type of co-functions from A to B'. The formation rule for co-function types is:

$$\frac{A : \mathtt{Set} \qquad B : \mathtt{Set}}{B \prec A : \mathtt{Set}} \,.$$

The introduction and elimination rules for co-function types and their opposites are presented in Table 2.

$\dfrac{a : \overline{A} \qquad b : B}{(a,b) : B \prec A}$	$\dfrac{c : B \prec A}{\mathsf{p}(c) : \overline{A}} \qquad \dfrac{c : B \prec A}{\mathsf{q}(c) : B}$
$\dfrac{\begin{array}{c}(x : \overline{A}) \\ \vdots \\ b(x) : \overline{B}\end{array}}{(\lambda x)b(x) : \overline{B \prec A}}$	$\dfrac{c : \overline{B \prec A} \qquad a : \overline{A}}{\mathsf{Ap}(c,a) : \overline{B}}$

Table 2: Introduction and elimination rules for co-function types and their opposites.

As the terms used in the rules for co-function types already exits in ITT, their computation rules are defined in the same way as in ITT. The extension of PTT_0 with co-function types will be called PTT_1.

With some laborious but not difficult work, the propositions-as-types correspondence between $\mathsf{PL}_0^{\mathsf{S}}$ and PTT_0 can be extended to obtain a propositions-as-types correspondence between $\mathsf{PL}_1^{\mathsf{S}}$ and PTT_1. Under such extended correspondence, the co-function type constructor will correspond to co-implication.

3 Duality in PTT_1

As it is pointed out in [5, p. 187]:

> Duality is an important general theme that has manifestations in almost every area of mathematics. Over and over again, it turns out that one can associate with a given mathematical object a related, 'dual' object that helps one to understand the properties of the object one started with. Despite the importance of duality in mathematics, there is no single definition that covers all instances of the phenomenon.

Although there is not a general definition of duality in mathematics, we shall take the description in [3], which is clear and general enough for our purposes here.

---

version (where the contexts of the premises are the same) and a *multiplicative* version (where the contexts of the premises can be different); while the symmetry in $\mathsf{PL}_1^{\mathsf{S}}$ is obtained by the formalisation of a primitive constructive negation whose rules are based on the understanding of negation as falsity, and on the notion of refutation which is dual to the notion of proof. The distinction between additive and multiplicative logical constants in $\mathsf{PL}_1^{\mathsf{S}}$ is at least not evident. Although a reviewer pointed us out that the unique general elimination rule for co-implication leads to a multiplicative version of the connective, while the two standard elimination rules leads to an additive version of the connective, the connection between multiplicativity/additivity (which are concepts usually defined when working with sequent calculus) and general-elimination-rules/standard-elimination-rules (which are concepts usually defined in natural deduction systems presented in standard format, that is, where the rules are presented without entailment relations nor contexts) is not clear for us. In [10], where a natural deduction system (in standard format) for intuitionistic linear logic is proposed, all elimination rules (for additive and multiplicative logical constants) are general. The additivity of logical constants is formalised by adding labels to the assumptions in context-sharing rules.

> In mathematics, a *duality* translates concepts, theorems or mathematical structures into other concepts, theorems or structures, in a one-to-one fashion, often (but not always) by means of an involution operation: if the dual of A is B, then the dual of B is A. Such involutions sometimes have fixed points, so that the dual of A is A itself.

Before explaining duality in PTT_1, we shall provide a rigorous definition of duality between logical constants in a logical system, and based on such definition we shall state the dualities between logical constants in $\mathsf{PL}_1^{\mathsf{S}}$.

Definition 3.1. Let L be a logical system with consequence relation $\Vdash_{\mathsf{L}}$:

(i) Two formulae A and B are *inter-substitutable in* L, which will be denoted by $A \triangleq B$, if for every formula C of L, when C' is the result of substituting some occurrences of A by B (or vice versa) in C, then $C \Vdash_{\mathsf{L}} C'$ and $C' \Vdash_{\mathsf{L}} C$.

(ii) A unary logical constant $*$ is an *involution in* L if $A \triangleq *(*A)$, for every formula A of L.

(iii) Two unary logical constants $\circ$ and $\bullet$ are *dual in* L, under an involution $*$, if $*(\circ A) \triangleq \bullet(*A)$ and $*(\bullet A) \triangleq \circ(*A)$, for every formula A of L.

(iv) Two binary logical constants $\square$ and $\blacksquare$ are *dual in* L, under an involution $*$, if $*(A \square B) \triangleq *A \blacksquare *B$ and $*(A \blacksquare B) \triangleq *A \square *B$, or if $*(A \square B) \triangleq *B \blacksquare *A$ and $*(B \blacksquare A) \triangleq *A \square *B$, for every pair of formulae A and B of L.

Quantifiers are considered unary logical constants, and the binding variables will be considered parameters that are part of the logical constant. For instance, $\forall x$ will be considered a logical constant, parametrised by x. When we say that quantifiers $\forall$ and $\exists$ are dual, we mean that they are dual under every parameter (or binding variable) x.

Theorem 3.2. *In* $\mathsf{PL}_1^{\mathsf{S}}$*:*

(i) If $\vdash_{\mathsf{PL}_1^{\mathsf{S}}} A \Leftrightarrow_{\mathsf{s}} B$*, then* $A \triangleq B$*.*

(ii) $\neg$ *is an involution.*

(iii) $\wedge$ *and* $\vee$ *are dual logical constants under* $\neg$*.*

(iv) $\supset$ *and* $\prec$ *are dual logical constants under* $\neg$*.*

(v) $\forall$ *and* $\exists$ *are dual logical constants under* $\neg$*.*

Now, we shall explain why the opposite type constructor in PTT_1 can be viewed as an involution operation that transforms types into their dual types. Firstly, we provide a rigorous definition of duality between type constructors in a type theory.

Definition 3.3. Let T be a type system with consequence relation $\Vdash_{\mathsf{T}}$, and let U be a universe of T:

(i) Two types A and B are *inter-substitutable in* U, which will be denoted by $A \triangleq_{\mathrm{U}} B$, if A and B are in U and for every type C in U, when C' is the result of substituting some occurrences of A by B (or vice versa) in C, then $x : C \Vdash_{\mathsf{T}} x : C'$ and $x : C' \Vdash_{\mathsf{T}} x : C$, for every variable x that is not in C.

(ii) A unary type constructor $*$ is an *involution in* U if $A \triangleq_{\mathrm{U}} *(*A)$, for every type A in U.

(iii) Two unary type constructors $\circ$ and $\bullet$ are *dual in* U, under an involution $*$, if $*(\circ A) \triangleq_{\mathrm{U}} \bullet(*A)$ and $*(\bullet A) \triangleq_{\mathrm{U}} \circ(*A)$, for every type A in U.

(iv) Two binary type constructors $\square$ and $\blacksquare$ are *dual in* U, under an involution $*$, if $*(A \square B) \triangleq_{\mathrm{U}} *A \blacksquare *B$ and $*(A \blacksquare B) \triangleq_{\mathrm{U}} *A \square *B$, or if $*(A \square B) \triangleq_{\mathrm{U}} *B \blacksquare *A$ and $*(B \blacksquare A) \triangleq_{\mathrm{U}} *A \square *B$, for every pair of types A and B in U.

Similarly as quantifiers are considered unary logical constants, we shall consider the type constructors Π and Σ unary type constructors, and the binding variables and their types will be considered parameters that are part of the type constructor. For instance, $\Pi x : A$ will be considered a unary type constructor parametrised by x and A. When we say that Π and Σ are dual, we mean that they are dual under every pair of parameters x and A.

In [1], equivalence and strong equivalence relations between types of PTT_0 are defined. We adapt these definitions for PTT_1.

Definition 3.4. Let A and B be two small types of PTT_1.

(i) A and B are *equivalent in* U_0, which will be denoted by $A \equiv_{\mathrm{U}_0} B$, if for every context Γ we have that $\Gamma \vdash_{\mathsf{PTT}_1} t : A$ iff $\Gamma \vdash_{\mathsf{PTT}_1} t : B$.

(ii) A and B are *strongly equivalent*, which will be denoted by $A \equiv^{\mathsf{s}}_{\mathrm{U}_0} B$, if $A \equiv_{\mathrm{U}_0} B$ and $\overline{A} \equiv_{\mathrm{U}_0} \overline{B}$.

Taking into account that for types $\overline{A \to B}$ and $A \times \overline{B}$ apply the same introduction and elimination rules, we could think that $\overline{A \to B} \equiv_{\mathrm{U}_0} A \times \overline{B}$ is a direct consequence of such fact. The following derivation shows that $x : \overline{A \to B} \vdash_{\mathsf{PTT}_1} (\mathsf{p}(x), \mathsf{q}(x)) : A \times \overline{B}$:

$$\dfrac{\dfrac{x : \overline{A \to B}}{\mathsf{p}(x) : A} \qquad \dfrac{x : \overline{A \to B}}{\mathsf{q}(x) : \overline{B}}}{(\mathsf{p}(x), \mathsf{q}(x)) : A \times \overline{B}}$$

However, it is necessary to include a conversion rule into PTT_1 in order to make $(\mathsf{p}(x), \mathsf{q}(x)) = x$ and prove that $x : \overline{A \to B} \vdash_{\mathsf{PTT}_1} x : A \times \overline{B}$. Similarly, for types $\overline{A \times B}$ and $\overline{A} + \overline{B}$ apply the same introduction and elimination rules, and the following derivation shows that $z : \overline{A \times B} \vdash_{\mathsf{PTT}_1} \mathsf{D}(z, (x)\mathsf{i}(x), (y)\mathsf{j}(y)) : \overline{A} + \overline{B}$:

$$\dfrac{z : \overline{A \times B} \qquad \dfrac{(x : \overline{A})}{\mathsf{i}(x) : \overline{A} + \overline{B}} \qquad \dfrac{(y : \overline{B})}{\mathsf{j}(x) : \overline{A} + \overline{B}}}{\mathsf{D}(z, (x)\mathsf{i}(x), (y)\mathsf{j}(y)) : \overline{A} + \overline{B}}$$

However, it is necessary to include a conversion rule into PTT_1 in order to make $\mathsf{D}(z, (x)\mathsf{i}(x), (y)\mathsf{j}(y)) = z$ and prove that $z : \overline{A \times B} \vdash_{\mathsf{PTT}_1} z : \overline{A} + \overline{B}$. Analogous situations occur when trying to proving some other apparently evident equivalences between types, which justifies the inclusion of the conversion rules presented in Table 3 into PTT_1, where $W \in \{A \to B, (\Pi x : A)B(x), \overline{B \prec A}, \overline{(\Sigma x : A)B(x)}\}$, $X \in \{A \times B, B \prec A, (\Sigma x : A)B(x), \overline{A + B}, \overline{A \to B}, \overline{(\Pi x : A)B(x)}\}$, $Y \in \{A + B, \overline{A \times B}\}$ and $Z \in \{(\Sigma x : A)B(x), \overline{(\Pi x : A)B(x)}\}$. The rules in the first row of the table are called *eta rules* and the ones in the second row are called *co-eta rules*.[7]

$\dfrac{t : W}{(\lambda x)\mathsf{Ap}(t, x) = t : W}$	$\dfrac{t : X}{(\mathsf{p}(t), \mathsf{q}(t)) = t : X}$
$\dfrac{t : Y}{\mathsf{D}(t, (x)\mathsf{i}(x), (y)\mathsf{j}(y)) = t : Y}$	$\dfrac{t : Z}{\mathsf{E}(t, (x, y)(x, y)) = t : Z}$

Table 3: Eta and co-eta conversion rules.

With the addition of the eta and co-eta conversion rules, the following strong equivalences between types of PTT_1 can be proven.

[7] For an in-depth discussion of eta and co-eta rules in Martin-Löf type theory see [7].

Theorem 3.5. *In* PTT_1*, for every two small types A and B, we have the following strong equivalences:*

$$\overline{A \to B} \equiv^{\mathsf{s}}_{\mathrm{U}_0} \overline{B} \prec \overline{A}, \qquad \overline{A \times B} \equiv^{\mathsf{s}}_{\mathrm{U}_0} \overline{A} + \overline{B}, \qquad \overline{(\Pi x : A)B} \equiv^{\mathsf{s}}_{\mathrm{U}_0} (\Sigma x : A)\overline{B},$$
$$\overline{B \prec A} \equiv^{\mathsf{s}}_{\mathrm{U}_0} \overline{A} \to \overline{B}, \qquad \overline{A + B} \equiv^{\mathsf{s}}_{\mathrm{U}_0} \overline{A} \times \overline{B}, \qquad \overline{(\Sigma x : A)B} \equiv^{\mathsf{s}}_{\mathrm{U}_0} (\Pi x : A)\overline{B}, \qquad \overline{\overline{A}} \equiv^{\mathsf{s}}_{\mathrm{U}_0} A.$$

Under the propositions-as-types interpretation in [1], equivalence and strong equivalence in $\mathsf{PL}^{\mathsf{S}}_0$ do not correspond, respectively, to equivalence and strong equivalence in PTT_0. For instance, we have that $\vdash_{\mathsf{PL}^{\mathsf{S}}_0} (A \wedge A) \Leftrightarrow A$ (and also $\vdash_{\mathsf{PL}^{\mathsf{S}}_0} (A \wedge A) \Leftrightarrow_{\mathsf{s}} A$), but $(A \times A) \not\equiv_{\mathrm{U}_0} A$ (and consequently $(A \times A) \not\equiv^{\mathsf{s}}_{\mathrm{U}_0} A$) in PTT_0. Moreover, while $\Leftrightarrow$ and $\Leftrightarrow_{\mathsf{s}}$ are logical constants defined in the object language of $\mathsf{PL}^{\mathsf{S}}_0$, the relations $\equiv_{\mathrm{U}_0}$ and $\equiv^{\mathsf{s}}_{\mathrm{U}_0}$ are defined on the meta-language. The same differences occur if the propositions-as-types interpretation is extended to $\mathsf{PL}^{\mathsf{S}}_1$ and PTT_1. In certain way, the equivalence relation defined for types is more exigent that the equivalence defined for formulae, demanding not only equivalence with respect to deductibility (or inhabitation) but also demanding that their proof terms (or inhabitants) are just the same.

In intuitionistic type theory (and other type theories) a conversion relation between types is defined in order to make equivalent types be equal under such conversion relation, thus ensuring uniqueness in type assignment. This is not possible in PTT_1 (and neither in PTT_0), because in these systems there are equivalent types that are not strongly equivalent (for instance, $\overline{A \to B} \equiv_{\mathrm{U}_0} A \times \overline{B}$ but $\overline{\overline{A \to B}} \not\equiv_{\mathrm{U}_0} \overline{A \times \overline{B}}$). When two types of PTT_1 are strongly equivalent they work as being the same type and 'substitution by strongly equivalent types is possible', what does not happen if they are only equivalent (for instance, we have that $\overline{A \to B} \equiv_{\mathrm{U}_0} A \times \overline{B}$, but $\overline{\overline{A \to B}} \not\equiv_{\mathrm{U}_0} \overline{A \times \overline{B}}$, consequently $\overline{A \to B}$ and $A \times \overline{B}$ are not inter-substitutable in PTT_1).

Theorem 3.6. *Let A and B be small types of* PTT_1.

(i) If $A \equiv^{\mathsf{s}}_{\mathrm{U}_0} B$*, then* $A \triangleq_{\mathrm{U}_0} B$.

(ii) If $A \triangleq_{\mathrm{U}_0} B$*, then for every context* Γ *and small type C, when* C' *is the result of substituting some occurrences of A by B in C, we have that* $\Gamma \vdash_{\mathsf{PTT}_1} t : C$ *iff* $\Gamma \vdash_{\mathsf{PTT}_1} t : C'$.

Theorem 3.7. *In the universe* U_0 *of* PTT_1*:*

(i) $\overline{}$ *is an involution.*

(ii) $\times$ *and* $+$ *are dual type constructors under* $\overline{}$.

(iii) $\to$ *and* $\prec$ *are dual type constructors under* $\overline{}$.

(iv) Π *and* Σ *are dual type constructors under* $\overline{}$.

Based on Theorem 3.7, we shall add to PTT_1 the equality rules in Table 4.

As in PTT_1 we do not have basic types,[8] we suppose that types of PTT_1 are generated by a denumerable set of type variables $V = \{\alpha, \alpha_1, \ldots, \beta, \beta_1, \ldots\}$ which represent arbitrary (possibly dependent) basic types. In types $(\Pi x : A)B$ and $(\Sigma x : A)B$ we shall call A the *generating type*, considering that $B(x)$ is a family of types generated on A.

Definition 3.8. Let A be a type of PTT_0. The *dual* of A, which will be denoted by $A^\star$, is the result of exchange $\to$ and $\prec$ and swap the type on the left-hand side with the type on the right-hand side of such constructors, exchange $\times$ and $+$, exchange Π and Σ, and exchange each type variable α by $\overline{\alpha}$, letting the generating types unchanged.

[8] By *basic types* we mean types without type constructors.

$\dfrac{A : \mathrm{U}_0 \qquad B : \mathrm{U}_0}{\overline{A \to B} = \overline{B} \prec \overline{A} : \mathrm{U}_0}$	$\dfrac{A : \mathrm{U}_0 \qquad B : \mathrm{U}_0}{\overline{B \prec A} = \overline{A} \to \overline{B} : \mathrm{U}_0}$
$\dfrac{A : \mathrm{U}_0 \qquad B : \mathrm{U}_0}{\overline{A \times B} = \overline{A} + \overline{B} : \mathrm{U}_0}$	$\dfrac{A : \mathrm{U}_0 \qquad B : \mathrm{U}_0}{\overline{A + B} = \overline{A} \times \overline{B} : \mathrm{U}_0}$
$\dfrac{A : \mathrm{U}_0 \qquad B : \mathrm{U}_0}{\overline{(\Pi x : A)B} = (\Sigma x : A)\overline{B} : \mathrm{U}_0}$	$\dfrac{A : \mathrm{U}_0 \qquad B : \mathrm{U}_0}{\overline{(\Sigma x : A)B} = (\Pi x : A)\overline{B} : \mathrm{U}_0}$
$\dfrac{A : \mathrm{U}_0}{\overline{\overline{A}} = A : \mathrm{U}_0}$	

Table 4: Equality rules for the opposite type constructor.

Theorem 3.9 (Principle of duality in PTT_1)**.** *Let A be a small type of PTT_1, then $\vdash_{\mathsf{PTT}_1} A = \overline{A^\star} : \mathrm{U}_0$.*

As it was pointed out in [1], in PTT_0 the type constructor $\to$ can be defined by $A \to B \stackrel{\text{def}}{=} \Pi x : A.B$, when x is not free in B, as in ITT (because $A \to B \equiv_s \Pi x : A.B$, when x is not free in B). However, while in ITT the type constructor $\times$ can be defined by $A \times B \stackrel{\text{def}}{=} (\Sigma x : A)B$, when x is not free in B, in PTT_0 this definition is not possible (because $\overline{A \times B} \not\equiv_{\mathrm{U}_0} \overline{(\Sigma x : A)B}$). The same happens in PTT_1. However, in PTT_1 we have that $B \prec A \equiv_s (\Sigma x : \overline{A})B$, when x is not free in B; which allows us to define the type constructor $\prec$ by $B \prec A \stackrel{\text{def}}{=} (\Sigma x : \overline{A})B$, when x is not free in B. This shows that co-function types in PTT_1 can be viewed as a kind of product, different of the Cartesian product $\times$.[9] While the opposite of a co-function type is a function type, the opposite of a Cartesian product is a disjoint union.

Moreover, as it was also pointed out in [1], while in ITT the type constructors $+$ and Σ cannot be defined by means of the other type constructors, in PTT_0 the type constructor $+$ can be defined by $A + B \stackrel{\text{def}}{=} \overline{\overline{A} \times \overline{B}}$ (because $A + B \equiv^{\mathsf{s}}_{\mathrm{U}_0} \overline{\overline{A} \times \overline{B}}$) and Σ can be defined by $(\Sigma x : A)B \stackrel{\text{def}}{=} \overline{(\Pi x : A)\overline{B}}$ (because $(\Sigma x : A)B \equiv^{\mathsf{s}}_{\mathrm{U}_0} \overline{(\Pi x : A)\overline{B}}$). These definitions are also valid in PTT_1.

Taking into account the possible definitions of type constructors described in the previous paragraphs, the sets of constructors $\{\Pi, \times, \overline{}\}$, $\{\Pi, +, \overline{}\}$, $\{\Sigma, \times, \overline{}\}$ and $\{\Sigma, +, \overline{}\}$ are complete for PTT_1.

Duality in PTT_1, and the equality rules in Table 4, also allows us to carry the opposite type constructors to basic types as stated below.

Definition 3.10. A type A of PTT_1 is in *opposite normal form* if the opposite constructor is only applied to type variables in A.

Theorem 3.11. *Every small type of PTT_1 is equal to a type in opposite normal form.*

4 Intuitive interpretations of opposite and co-function types

Martin-Löf [9, p. 5] provides four different intuitive interpretations of judgements in intuitionistic type theory. In one of such interpretations, types are understood as ‘intentions’. However, as the notion of intention is too vague, we shall not consider this interpretation here. The other three interpretations are shown in Table 5.

The first interpretation corresponds to the so-called *propositions-as-types interpretation*. With respect to the second interpretation, Martin-Löf explains that:

[9]The Cartesian product in ITT is defined as $A \times B \equiv_s (\Sigma x : A)B$, when x is not free in B. This definition corresponds to $B \prec \overline{A}$ in PTT_1, because $B \prec \overline{A} \stackrel{\text{def}}{=} (\Sigma x : \overline{\overline{A}})B \equiv_s (\Sigma x : A)B$.

A : **Set**	**a** : **A**	**Inhabitation of A**
A is a proposition	a is a proof (construction) of proposition A	A is true
A is a problem (task)	a is a method of solving the problem (doing the task) A	A is solvable
A is a set	a is an element of the set A	A is non-empty

Table 5: Interpretations of Martin-Löf's judgements forms.

> [This intepretation] is very close to programming, 'a is a method ...' can be read as 'a is a program ...'. Since programming languages have a formal notation for the program a, but not for A, we complete the sentence with '...which meets the specification A'. In Kolmogorov's interpretation, the word problem refers to something to be done and the word program to how to do it.

In the third interpretation types are interpreted as sets.

Under each one of the interpretations of types, constructors $\rightarrow, \times, +, \Pi, \Sigma$ have their respective interpretations in ITT, which are shown in Table 6. In such table, symbols $\supset, \wedge, \vee, \forall, \exists$ are the intuitionistic logical constants for implication, conjunction, disjunction, universal quantification and existential quantification, respectively. Moreover, a many-sorted version of first-order intuitionistic logic must be considered [1], and the sort of variables are indicated by superscripts. In the third column, by $\{B_x\}_{x\in A}$ we denote a *family of problems (or problem specifications)* that is parametrised by elements in A; that is, for each $x \in A$, we have that B_x is a specification of a problem. By a *general method for solving a family of problems* $\{B_x\}_{x\in A}$ we mean a single method that for each $x \in A$ gives a solution for B_x, and by a *particular method for solving a problem in a family* $\{B_x\}_{x\in A}$ we mean a method that for some $x \in A$ gives a solution for B_x.

	Types interpreted as propositions	**Types interpreted as problems**	**Types interpreted as sets**
$A \rightarrow B$	$A \supset B$	Methods that transforms any solution of A into a solution of B	Set of functions from A to B
$A \times B$	$A \wedge B$	Methods that solve A and B	Cartesian product of A and B
$A + B$	$A \vee B$	Methods that solve A or B	Disjoint union of A and B
$(\Pi x : A)B$	$(\forall x^A)B$	General methods for solving the family of problems $\{B_x\}_{x\in A}$	Generalised Cartesian product of the family of sets $\{B_x\}_{x\in A}$
$(\Sigma x : A)B$	$(\exists x^A)B$	Particular methods for solving a problem in the family $\{B_x\}_{x\in A}$	Generalised disjoint union of the family of sets $\{B_x\}_{x\in A}$

Table 6: Interpretation of constructors in ITT.

In PTT_1, the type constructors $\rightarrow, \times, +, \Pi, \Sigma$ can be interpreted in the same way that in ITT, under every interpretation of types. In the logical interpretation, logical constants correspond to those of $\mathsf{PL}_1^{\mathsf{S}}$. Now, we shall explain how opposite types and co-function types can be interpreted in every interpretation of types.

(i) For the interpretation of types as propositions, as we mentioned in Section 2, the propositions-as-types correspondence between PL_0^S and PTT_0 can be extended to obtain a propositions-as-types correspondence between PL_1^S and PTT_1. Under such extended correspondence:

- The opposite type constructor corresponds to the negation of PL_1^S. Thus, following the same interpretation of negation in PL_0^S, the negation in PL_1^S represents falsity and consequently the inhabitation of $\overline{A}$ can be understood as '$\neg A$ is true' or as 'A is false', and $a : \overline{A}$ can be understood as 'a is a proof of the negation of A' or as 'a is a refutation of A'. Under this interpretations, the dualities in Section 3 make sense.
- The co-function type constructor corresponds to co-implication in PL_1^S.

(ii) Under the interpretation of types as problems:

- A type $\overline{A}$ can be understood as a specification of a problem whose solution excludes a solution of A. Consequently, inhabitation of $\overline{A}$ can be understood as '$\overline{A}$ is solvable' or as 'A is unsolvable', and $a : \overline{A}$ can be understood as 'a is a method that solves $\overline{A}$' or as 'a is a method that shows the unsolvability of A'. Under this interpretations, the dualities in Section 3 make sense.
- As $B \prec A \equiv_{\mathrm{U}_0} \overline{A} \times B$, thus $c : B \prec A$ can be understood as 'c is a method that solves $\overline{A}$ (or shows the unsolvability of A) and that solves B'.

(iii) Under the interpretation of types as sets:

- It is harder to glimpse an intuitive interpretation for opposite types. Taking into consideration that, if we interpret types as sets, the inhabitation of type A means that A is non-empty, we might initially think that the duality in this case is between non-emptiness and emptiness. Accordingly, the inhabitation of $\overline{A}$ should be interpreted as '$\overline{A}$ is non-empty' or as 'A is empty'. But it does not make sense that the non-emptiness of a set (i.e. $\overline{A}$) leads to the emptiness of another set (i.e. A). However, considering an intentional conception of sets, under which it can be roughly said that a set consists in a collection of individuals that fall under an associated concept,[10], which we shall denote by C_A, we can understood $\overline{A}$ as the collection of individuals that fall under the 'dual concept' to C_A. Consequently, the inhabitation of $\overline{A}$ must be understood as 'there are individuals that fall under the concept $\mathrm{C}_{\overline{A}}$' or as 'there are individuals that fall under the dual concept to C_A'. Under this (somewhat blurred) intentional interpretation of sets, the dualities in Section 3 make sense. However, taking into account Theorem 3.11, for a full interpretation of types as sets we just need to have the interpretation of opposite basic types, but in this endeavour it is necessary to define what exactly means 'dual concept' (at least for concepts associated with basic types), which is not a simple task and is left for future work.
- As $B \prec A \equiv_{\mathrm{U}_0} \overline{A} \times B$, thus $c : B \prec A$ can be understood as 'c is a pair of individuals, the first one falling under the concept $\mathrm{C}_{\overline{A}}$ (or under the concept dual to C_A) and the second one falling under the concept C_B'.

Acknowledgements

The authors are deeply grateful to the anonymous reviewers for their valuable comments and suggestions which led to important improvements to the article.

[10]For an historical and philosophical analysis of the concept of set, the differentiation between intensional and extensional, and also between essential and accidental conceptions, see [6, Chapter III].

References

[1] Juan C. Agudelo-Agudelo & Andrés Sicard-Ramírez (2021): *Type Theory with Opposite Types: A Paraconsistent Type Theory*. *Logic Journal of the IGPL*, doi:10.1093/jigpal/jzab022.

[2] Ahmad Almukdad & David Nelson (1984): *Constructible Falsity and Inexact Predicates*. *The Journal of Symbolic Logic* 49(1), pp. 231–233, doi:10.2307/2274105.

[3] Wikipedia contributors (2021): *Duality (mathematics) — Wikipedia, The Free Encyclopedia*. Available at `https://en.wikipedia.org/w/index.php?title=Duality_(mathematics)&oldid=1018141284`. [Online; accessed 22-April-2021].

[4] Roberto Di Cosmo & Dale Miller (2019): *Linear Logic*. In Edward N. Zalta, editor: *The Stanford Encyclopedia of Philosophy*, Summer 2019 edition, Metaphysics Research Lab, Stanford University.

[5] Timothy Gowers, June Barrow-Green & Imre Leader, editors (2008): *The Princeton Companion to Mathematics*. Princeton University Press, Princeton.

[6] Johan Georg Granström (2011): *Treatise on Intuitionistic Type Theory*. *Logic, Epistemology and the Unity of Science* 22, Springer, doi:10.1007/978-94-007-1736-7.

[7] Ansten Klev (2019): *Eta-rules in Martin-Löf Type Theory*. *The Bulletin of Symbolic Logic* 25(3), pp. 333–359, doi:10.1017/bsl.2019.21.

[8] E. G. K. López-Escobar (1972): *Refutability and Elementary Number Theory*. *Indagationes Mathematicae (Proceedings)* 75(4), pp. 362–374, doi:10.1016/1385-7258(72)90053-4.

[9] Per Martin-Löf (1984): *Intuitionistic Type Theory*. Bibliopolis, Napoli. Notes by Giovanni Sambin of a series of lectures given in Padua 1980.

[10] Sara Negri (2002): *A Normalizing System of Natural Deduction for Intuitionistic Linear Logic*. *Archive for Mathematical Logic* 41(8), pp. 789–810, doi:10.1007/s001530100136.

[11] Sara Negri & Jan von Plato (2001): *Structural Proof Theory*. Cambridge University Press, doi:10.1017/CBO9780511527340.

[12] David Nelson (1949): *Constructible Falsity*. *The Journal of Symbolic Logic* 14(1), pp. 16–26, doi:10.2307/2268973.

[13] Heinrich Wansing (2008): *Constructive negation, implication and co-implication*. *Journal of Applied Non-Classical Logics* 18(2–3), pp. 341–364, doi:10.3166/jancl.18.341-364.

[14] Heinrich Wansing (2016): *Falsification, Natural Deduction and Bi-Intuitionistic Logic*. *Journal of Logic and Computation* 26(1), pp. 425–450, doi:10.1093/logcom/ext035.