Meaning and Computing: Two Approaches to Computable Propositions*

Ivo Pezlar[1][0000−0003−1965−2159]

Czech Academy of Sciences, Institute of Philosophy, Jilska 1, 110 00 Prague, Czechia
pezlar@flu.cas.cz

Abstract. In this paper, we will be interested in the notion of a computable proposition. It allows for feasible computational semantics of empirical sentences, despite the fact that it is in general impossible to get to the truth value of a sentence through a series of effective computational steps. Specifically, we will investigate two approaches to the notion of a computable proposition based on constructive type theory and transparent intensional logic. As we will see, the key difference between them is their accounts of denotations of empirical sentences.

Keywords: computable proposition · sense–denotation distinction · algorithmic theory of meaning · procedural semantics · constructive type theory · transparent intensional logic.

1 Introduction

In this paper, we will be interested in the following issue:

– Can we make sense of empirical sentences in computational terms given that it is generally impossible to get to the truth value of a sentence through a sequence of effective computational steps?

The answer we will put forward is positive, but it requires adoption of the notion of a computable proposition, i.e., a proposition that yields another proposition upon execution, not a truth value. These computable propositions will be understood as meanings of empirical sentences. We examine two possible approaches to this notion, namely, an approach based on constructive type theory (CTT, [14]) and an approach based on transparent intensional logic (TIL, [30]), and we try to clarify the relation between their respective notions of a computable proposition. But first we will look more closely at the general relationship between meaning and computing.

* Work on this paper was supported by Grant Nr. 19-12420S from the Czech Science Foundation, GA ČR.

Let us start by considering the following mathematical object:[1] $2+2$. Using a common sense notion of computation, we can compute it and get the number 4. In what kind of a relationship does the object 4 stand to the object $2+2$? The standard answer is that the relation between $2+2$ and 4 can be understood in terms of an evaluation procedure. The number $2+2$ is a non-canonical form of the canonical number 4 and the latter can be reduced to the former by following appropriate reduction rules associated with the non-canonical operator $+$ used in constructing this non-canonical number. Thus, generally speaking, we can view the non-canonical objects as programs and their corresponding canonical objects as their values. For example, the program $2+2$ terminates at the value 4. Can we then conclude that the *meaning* of the object $2+2$ is its canonical value, i.e., the object 4 in this case? We can, but then we would also have to concede that all other programs that terminate at the same value (e.g., 2×2, $5-1$ or $16 \div 4, \ldots$) have the same meaning, but this would be a hard pill to swallow for many. So what is the meaning of $2+2$, if not its canonical value? Whatever it is, it seems to be connected with the way we are computing its value, i.e., reducing it as a non-canonical object to a canonical one, not with the value itself.

Behind every non-canonical object a there is an unspoken question: 'Can a be reduced to a canonical object'? Could we, perhaps, view this question as roughly equivalent to the question: 'Does a mean anything?' In other words, are the non-canonical objects meaningful only insofar as they are reducible to their corresponding canonical forms?[2] This, however, seems to be too strong of a requirement. Generally, we seem to be understanding non-canonical objects just fine even in cases where we do not know (or cannot know) their corresponding canonical objects. Take, e.g., the following non-canonical object $16547 + 3^4$. It seems clear that we can understand it even if we do not know its value. If we compute it, we learn something new (the canonical form of this object), but the meaning of the original object seems to remain unchanged by this discovery – it does not seem to imbue $16547 + 3^4$ with more meaning than it had before the computation.

Probably the easiest way to make sense of this situation is to accept some form of Fregean meaning-denotation dichotomy: non-canonical objects are meanings 'in themselves' and they do not need to lead to any denotations in order to be intelligible for us. In other words, non-canonical objects do not become meaningful insofar as they are computable to canonical ones, they stand on their own, so to speak.

So far, we have talked about computation and meaning only in regards to mathematical objects, but can we adopt an analogous approach to the empirical

[1] For simplicity, we assume we are working directly with the mathematical objects, thus ignoring the syntactic layer for the sake of the semantic one. For example, consider the difference between a mathematical expression '2+2' and a mathematical object $2+2$: while the former can be reduced to the numeral '4', the latter can be computed to the number 4.

[2] As is the case in, e.g., the Dummett–Prawitz-style proof-theoretic semantics (see [6], [19]).

discourse as well? Consider, e.g., the proposition: *Charles is a bachelor*. In what sense, if any, can we compute it? Clearly, we cannot compute it to its truth value since there is no general method how to compute propositions to their truth values. But maybe there is another way we can go about computing it, and thus carrying over the non-canonical and canonical object distinction?

Let us start with a simple example from intuitionistic logic.[3] A proposition $\neg A$ is usually defined as $A \to \bot$ where $\bot$ denotes absurdity. Note that the meaning of the proposition $\neg A$ is explained indirectly, since we have no proof conditions for it, strictly speaking, only for $A \to \bot$. In other words, if we want to inquire into the meaning of $\neg A$, first we have to transform it into $A \to \bot$. Arguably, this definitional transformation can be regarded as a basic computational step itself. And if so, then we can view $\neg A$ as a non-canonical proposition that can be computed to its canonical form $A \to \bot$.

The same approach, we believe, can be applied to empirical propositions as well. For example, the empirical proposition *Charles is a bachelor* can be viewed as a non-canonical proposition, i.e., a proposition that has no direct truth conditions. And if we want to inquire into its meaning, or rather its truth conditions, we have to transform it to its canonical form. In this case, it might be, for example: *Charles is a man* $\wedge \neg$(*Charles is married*). Thus, the empirical proposition *Charles is a bachelor* can be computed to *Charles is a man* $\wedge$ $\neg$(*Charles is married*). The proposition *Charles is a bachelor* can then be understood as the meaning of the sentence 'Charles is a bachelor', while the proposition *Charles is a man* $\wedge \neg$(*Charles is married*) as its denotation. So, in this approach, canonical propositions are propositions that cannot be computed any further.[4]

From a historical perspective, the idea of entertaining computable propositions can be traced back to two different philosophical and logical sources: a constructive (intuitionistic) tradition concerned with the language of mathematics and logic that starts with [1] and a non-constructive (platonist) tradition concerned with natural language that starts later with [28]. In the rest of the paper, we examine two type-theoretic frameworks that – we believe – best represent these two traditions: Per Martin-Löf's constructive type theory ([14]) and Pavel Tichý's transparent intensional logic ([30]). Specifically, we will look at how these systems approach the notion of a computable proposition, a task which can be split into two further questions: 'What is a proposition?' and 'What is a computation?'[5] In other words, our main aim will be to discuss the notion

[3] This example as well as the whole idea of computing propositions to canonical forms not to truth values was proposed in [15], a paper presented at a workshop on Frege at the University of Leiden in August 25, 2001, transcribed by Bjørn Jespersen.

[4] Since we are mainly interested in the notion of a computable proposition, we intentionally choose very basic examples of propositions such as '$\neg A = A \supset \bot$' or 'Charles is a bachelor' to keep the focus on the computability aspect rather than on the propositional aspect. To learn how to analyze more complex sentences in CTT, consult, e.g., [25], [12], [2]. For TIL, see, e.g., [7], [21], [22].

[5] This investigation can be viewed as a follow up to the paper [18] that explores how these two systems approach mathematical and logical propositions.

of a computable proposition from the perspectives of CTT and TIL and to try to conceptually clarify the relation between their respective approaches to this notion. We will not aim to provide new technical developments for CTT or TIL.

Terminological note. Since we will be discussing two frameworks with different terminological backgrounds, we will try to simplify the vocabulary whenever reasonably possible. For example, we will call constructive sets of constructive type theory as types and hyperpropositions of transparent intensional logic as propositions. Deviations from the standard terminology such as these will always be pointed out throughout the text.

2 Constructive tradition

Although it was probably Aarne Ranta ([25]) who first applied constructive type theory to a systematic study of natural language semantics,[6] the general meaning as a computation approach can be traced much further, at least to the work of L. E. J. Brouwer on constructive mathematics, specifically on constructive logic and the proof-based account of logical constants, also known as the Brouwer-Heyting-Kolmogorov (or simply BHK) interpretation. The general idea is to explain the meaning of logical connectives not in terms of their truth conditions but in terms of their proof conditions. For the constructive propositional logic, we get the following explanations (see Tab. 1).

Table 1. The BHK interpretation of logical constants

a proof of the proposition	*consists of*	*which can be formalized as*
$A \wedge B$	a proof of A and a proof of B	$(a, b) : A \wedge B$
$A \vee B$	a proof of A or a proof of B	$i(a) : A \vee B$ or $j(b) : A \vee B$
$A \to B$	an effective method which takes any proof of A into a proof of B	$\lambda x.b(x) : A \to B$ (where $b(a)$ is a proof of B provided a is a proof of A)
$\bot$	there is no proof (absurdity)	–

This interpretation of logical constants, particularly of implication, served as a basis for another important discovery, specifically the observed analogy between proofs and programs and proposition and types.

2.1 Proofs as programs

The intuition that logic and computing are somehow related has a long tradition – going back at least to Leibniz's concept of the calculus ratiocinator – however,

[6] Following Michael Dummett's exploration of constructive principles outside the scope of mathematics ([5]) and Göran Sundholm's analysis of donkey sentences using constructive type theory ([27]).

it hasn't been made precise until the discovery of the propositions as types principle.[7] Briefly put, it identifies constructive proofs with programs.[8]

As hinted above, the propositions as types principle is closely related to the BHK interpretation of constructive logic and in its most common form it refers to the recognized correspondence between Gerhard Gentzen's intuitionistic natural deduction ([8]) and Alonzo Church's simply typed λ-calculus ([3]),[9] which can be understood as a rudimentary functional programming language (see, e.g., [11]). Specifically, we get the following correspondences (see Tab. 2).

Table 2. The propositions as types principle

natural deduction	λ-*calculus*
assumption	free variable
implication introduction	λ-abstraction
(= 'deduction theorem')	(= function definition)
implication elimination	β-reduction
(= modus ponens)	(= function application)
proposition	type
proof	(functional) program

In practice, this means that, e.g., $A \to B$ can be interpreted as an implicational proposition where A is the antecedent and B the consequent and simultaneously as a type of a function from objects of type A to objects of type B. Proving this proposition then corresponds to constructing an object of the type $A \to B$. Of course, these correspondences can be expanded (e.g., simplification of proofs understood as evaluation of programs, provability as a problem of type inhabitation, etc.) as well as generalized (introduction rules as constructors, elimination rules as destructors, etc.).

2.2 Constructive type theory

The proposition as types principle is one of the fundamental principles of constructive type theory (CTT, [14]) and it will help us to explain what is understood as a proposition in this framework. Specifically, under the propositions as types principle, the question 'What is a proposition?' becomes synonymous

[7] Also known as the Curry-Howard correspondence or isomorphism ([4], [10]), although this name is rather unfortunate as it omits two other key figures of the discovery, namely N. G. de Bruijn and Per Martin-Löf.

[8] For an excellent overview, see, e.g., [31].

[9] For simplicity, we omit the Haskell Curry's side of the discovery – the fact that combinators from combinatory logic correspond to axioms in Hilbert-style calculus. For example, the combinator K (i.e., $\mathsf{K}xy = x$) corresponds to the axiom $A \to B \to A$.

with the question 'What is a type?'[10] In CTT, we specify a type by specifying what constitutes its canonical objects and when two canonical objects are equal. For example, the type N of natural numbers is specified by the following rules introducing canonical objects and equal canonical objects of type N:

$$0 : N \qquad 0 = 0 : N \qquad \frac{n : N}{s(n) : N} \qquad \frac{n = m : N}{s(n) = s(m) : N}$$

where $0 : N$ is a judgment stating that '0 is an object of type N'. Thus, the type N is inhabited by canonical objects of the form $s(n)$ and 0. Furthermore, any object that reduces to a canonical object of a certain type is considered an object of that type as well.

This brings us to the notion of a computation. From the perspective of CTT, a computation is a process of reducing non-canonical objects to their canonical forms via the corresponding computation rules. For example, assuming we have defined 1 as $s(0)$, 2 as $s(1)$, etc., and $+$ as $a{+}0 = a : N$ and $a{+}s(b) = s(a{+}b) : N$, we can form a non-canonical natural number $2 + 2$ that can be computed to $s(2 + 1)$ (lazy evaluation) or fully evaluated to 4.[11]

For the basic semantic scheme of CTT (based on [20]), see the Fig. 1. It implements the Fregean idea that meaning is a mode of presentation for picking out denotation (in modern terms, a program for computing denotation). Note that since we are able to state things like $2 + 2 = 4 : N$, it means that the result of a computation, i.e., 4 in this case, is an object of the same type N as the computation itself, i.e., $2 + 2$ (see Fig. 2). In other words, the levels of meanings and denotations are conflated, or rather, viewed as entities of the same category.[12]

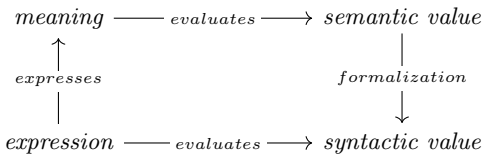

Fig. 1. Semantic scheme of CTT

[10] Or more precisely, 'What is a constructive set?' since in CTT the word 'type' is typically reserved for the higher-order presentation of CTT, which we will not use here. In CTT, one typically starts with the notion of a set and defines a proposition in terms of it. The category of sets is then identified with the category of propositions, which is the way the propositions as types principle is adopted in CTT. For more, see, e.g., [16].

[11] The natural number $2+2$ is considered non-canonical because it does not follow the forms prescribed by the introduction rules for the type N, i.e., it is neither 0 nor does it have the form $s(n)$.

[12] See, e.g., [20], pp. 21-26. This, as we will see, is in contrast with TIL, where meanings and denotations are kept apart and viewed as entities of distinct kinds.

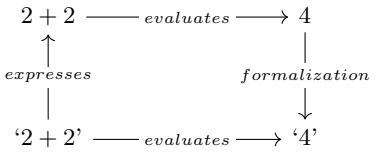

Fig. 2. An example of CTT semantics of non-empirical expressions

Now that we have briefly acquainted ourselves with the notions of a type and a computation, let us return back to propositions. So what exactly is a proposition in CTT? Analogously to what has been said above, to be able to judge that some A is a proposition, we have to know how to form its canonical objects and under what conditions two canonical objects are equal, i.e., provide rules for constructing its canonical proofs and equal canonical proofs.

Furthermore, a proposition A is considered true when we have a proof a of A. And in order to be able to judge that we have a proof a of A, we have to know that A is a proposition and that a is a method that yields upon execution a canonical proof of A as a value.[13] For example, the canonical proofs (objects) and equal canonical proofs of the proposition $A \to B$ are specified by the following pair of rules:

$$\frac{\begin{array}{c}[x : A] \\ b(x) : B\end{array}}{\lambda x.b(x) : A \to B} \qquad \frac{\begin{array}{c}[x : A] \\ b(x) = c(x) : B\end{array}}{\lambda x.b(x) = \lambda x.c(x) : A \to B}$$

What is the canonical proof $\lambda x.b(x)$ of the proposition $A \to B$? It is an object – a lambda coding – that codes a function which takes a proof a of A and transforms it into a proof $b(a)$ of B (compare this with the BHK interpretation for implication in Fig. 1).

Analogously to the case above with non-canonical natural numbers such as, e.g., $2 + 2 : N$, we can have non-canonical proofs of propositions as well. For example, consider the following derivation:

$$\frac{f : (A \to B) \to (C \to D) \qquad g : (A \to B)}{ap(f, g) : C \to D}$$

The proof $ap(f, g)$ of $C \to D$ is non-canonical because it does not have the form prescribed by the introduction rules for implicational propositions, i.e., $\lambda x.b(x)$. However, the non-canonical object $ap(f, g)$ is a method of obtaining a canonical object of $C \to D$. How to execute it? We have $f : (A \to B) \to (C \to D)$ which means that f is a method which yields a canonical object $\lambda x.c(x)$ of $(A \to B) \to (C \to D)$. Now, let us take $g : (A \to B)$ and substitute it for x in $c(x)$. Thus we get $c(g) : C \to D$ and when we execute $c(g)$ it will yield

[13] It is an open question how to best carry over this approach to empirical discourse. See, e.g., [27], [32], [26].

a canonical object of $C \to D$, i.e., something of the form $\lambda y.d(y)$. For a more thorough exposition, see [14].

Note, however, that all the computations discussed so far were concerned exclusively with objects, specifically, with reducing non-canonical objects into canonical ones. For example, $2 + 2 : N$ was computed to $s(2 + 1) : N$ or $ap(f, g) : C \to D$ was computed to $c(g) : C \to D$. But what about computing with propositions themselves? This is, after all, our main interest. To extend computations towards propositions, we utilize the notion of a non-canonical type introduced in [9].

Through the propositions as types principle, this naturally applies to propositions as well. Following [9] (p. 91, def. 6), let us define a non-canonical proposition A as a proposition that has a canonical proposition as the value.[14] The fact that the non-canonical proposition A has the canonical proposition B as value will be denoted as

$$A \Rightarrow B : prop$$

and can be read as 'a proposition A computes to a proposition B'. The meaning of these proposition-computability judgments will be specified inferentially by the corresponding computation rules whose conclusions will take the general form $A \Rightarrow B : prop$ (see, e.g., the computation rule for $\neg comp$ below). Furthermore, again following [9] (p. 97), we adopt explicit definitions of non-canonical propositions of the following form: $D =_{def} A : prop$ where A is a proposition (not necessarily canonical) and D is a new undefined non-canonical proposition. Basically, it tells us that A is the value for the non-canonical proposition D. For example, our motivating case involving the definition of intuitionistic negation can be formalized as $\neg A =_{def} A \to \bot : prop$ which justifies a computational step from $\neg A$ to $A \to \bot$ that can be captured as follows $\neg A \Rightarrow A \to \bot : prop$. The corresponding proposition computation rule (read bottom-up) will be then as follows:

$$\frac{A \to \bot \;\Rightarrow\; A \to \bot : prop}{\neg A \;\Rightarrow\; A \to \bot : prop}\ \neg comp$$

with $\neg A$ being a non-canonical proposition and $A \to \bot$ being a canonical one. In other words, $A \to \bot$ is the value of the computable proposition $\neg A$. From the above considerations, it also follows that $\neg A = A \to \bot : prop$, i.e., that they are equal propositions. [15]

[14] For a proper specification, see [9], especially sections *§ 4. Noncanonical sets and elements* and *§ 5. Nominal definitions*. It is also worth to note that non-canonical propositions/sets are already considered in [13], however, as opposed to [9], no dedicated proposition-computability judgments of the form $A \Rightarrow B : prop$ are used.

[15] Of course, more complex reductions for other logical and/or mathematical propositions can be introduced. For example, [24] (pp. 41-42) shows how we can in CTT define, and thus reduce the propositional function $prime(x)$ (assuming $x : N$) into more basic concepts. Formulating the corresponding computation rule based on the provided definition is then straightforward.

Now, returning to our empirical case, let us assume the following definition: $bachelor(x) =_{def} man(x) \wedge \neg married(x)$ which can be unpacked, analogously as above, into the following three steps. First, we postulate that $bachelor(x) : prop$ assuming $x : A$, i.e., that $bachelor(x)$ is a propositional function taking as arguments objects of some type A (analogously with $married(x)$ and $man(x)$). Then we add the corresponding computation rule (again, assuming $x : A$):

$$\frac{man(x) \wedge \neg married(x) \;\Rightarrow\; man(x) \wedge \neg married(x) : prop}{bachelor(x) \;\Rightarrow\; man(x) \wedge \neg married(x) : prop}$$

Note that here we are taking *man* and *married* as basic, further non-computable concepts.

And finally, we have to show that $bachelor(x)$ and $man(x) \wedge \neg married(x)$ are equal propositional functions, which follows from our computation rule: since both $bachelor(x)$ and $man(x) \wedge \neg married(x)$ have the same canonical forms, we can judge that they are equal propositional functions producing equal propositions. In other words, non-canonical propositions are equal, if their corresponding canonical propositions are equal. To sum it up, the proposition $man(Charles) \wedge \neg married(Charles) : prop$ is a canonical, i.e., further irreducible proposition and $bachelor(Charles)$ is a non-canonical proposition that can be computed to $man(Charles) \wedge \neg married(Charles) : prop$. For an example of the corresponding expanded semantic scheme for computable propositions, see Fig. 3.[16]

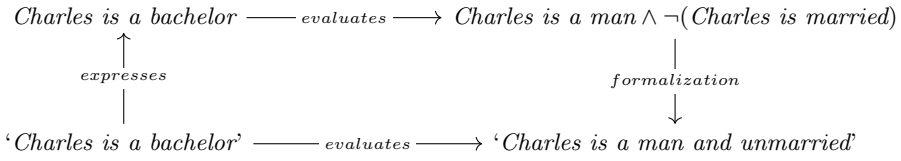

Fig. 3. An example of CTT semantics of empirical expressions

To conclude, in CTT we can approach the notion of a computable proposition via the notion of a non-canonical type introduced by [9], i.e., non-canonical constructive sets in a more standard terminology. This explication rests on two main principles: identification of propositions with types (i.e., the Curry-Howard isomorphism) and capturing computation in terms of the reduction of non-canonical objects to canonical ones.

In the following section, we will examine how TIL approaches the notion of a computable proposition.

[16] Note that our approach has nothing further to say about the meaning of the conjuncts of this canonical proposition. More specifically, the meaning of atomic empirical propositions such as $man(Charles)$ is assumed to be given externally.

3 Non-constructive tradition

As far as we know, it was Pavel Tichý ([28]) who first explicitly suggested to understand meanings of natural language expressions in terms of computations in the late 1960s, specifically in his paper *Intension in terms of Turing machines*:

> ...the fundamental relationship between sentence and procedure is obviously of a semantic nature; namely, the purpose of sentences is to record the outcome of various procedures. Thus e.g. the sentence 'The liquid X is an acid' serves to record that the outcome of a definite chemical testing procedure applied to X is positive. ([28], p. 7)

Tichý saw in Turing machines an opportunity to finally explicate Frege's notion of sense in rigorous terms. Eventually, however, Tichý replaced Turing machines with constructions. Constructions were understood as abstract computations codifying the procedures for computing denotations of the corresponding natural language expressions. Tichý subsequently developed his ideas into a system called transparent intensional logic presented in ([30]), which is still being actively developed (see, e.g., [7], [21], [22]).

3.1 Transparent intensional logic

As we have seen in CTT, the notion of a proposition is ultimately grounded in the notions of a type and a computation. In transparent intensional logic (TIL, [30]), the notion of a proposition can also be explained in these notions, however, the notions of a type and a computation in TIL differ from those of CTT.

First of all, the fundamental notion of computation in TIL, i.e., a construction, is much broader in comparison with the stricter constructive notion of effective computation found in CTT. It encompasses even non-computable and ill-specified procedures. As Tichý puts it:

> [N]ot every construction is an algorithmic computation. An algorithmic computation is a sequence of *effective* steps, steps which consist in subjecting a manageable object (usually a symbol or a finite string of symbols) to a feasible operation. A construction, on the other hand, may involve steps which are not of this sort. [...]. As distinct from an algorithmic computation, a construction is an ideal procedure, not necessarily a mechanical routine for a clerk or a computing machine. ([29], p. 526)

Thus, constructions should be viewed as idealized, abstract, not necessarily effective computations that need not be realizable either by a human or a machine. This more general approach towards computations is reflected in the treatment of types as well. In comparison with CTT, very little is required of them, no canonical objects have to be presented, no criterion of identity is required.[17]

[17] From this perspective, TIL types are much closer to categories in CTT (i.e., types in proper CTT terminology), but even a category is a stricter notion as it has to come with a criterion of application and identity, which is not the case with TIL types.

Specifically, types are understood simply as non-empty pairwise disjoint collections ([30], p. 65). No further stipulations are given of what can and cannot constitute such a collection.

So what is a proposition in TIL?[18] In order to understand what a proposition is we have to better understand Tichý's notion of a construction. First, it is important to make clear what is really meant in TIL when it is said that propositions can be computed to yield their denotations (for the basic Fregean semantic scheme of TIL, see Fig. 4). In most current instances of TIL when one talks about denotations of propositions, they do not mean truth values (understood as references of sentences) but functions from possible worlds to truth values.[19] Actual truth values cannot be, of course, computed and the task of determining them is delegated to empirical investigations. Thus, in TIL, we can compute the denotations of empirical sentences, but we cannot compute their references (i.e., truth values). Hence, the problem of computing truth values of empirical sentences is effectively sidestepped by distinguishing between denotations and references. This, however, does not apply to non-empirical expressions, where denotations and references are typically conflated.

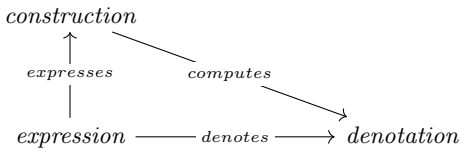

Fig. 4. Semantic scheme of TIL

For example, the non-empirical expression '2 + 2' is understood as expressing the arithmetical construction [**2 + 2**] that computes (or constructs, in standard terminology) the number 4, i.e., an object of type ν, where ν is the type of natural numbers, which can be regarded as both the denotation and the reference of '2+2' (see Fig. 5).[20] In contrast, the empirical sentence 'Charles is a bachelor' expresses the proposition λw[[**Bachelor** w] **Charles**] which computes its denotation, i.e., a function of type $(o\omega)$, where o is the type of truth values and ω is the type of

[18] In standard TIL terminology, the term 'hyperproposition' is used instead and the term 'proposition' is reserved for functions from possible worlds and time moments to truth values. We will diverge from this terminology.

[19] In standard TIL, denotations of propositions are understood as functions from possible world *and* time moments to truth values, however, we omit the time parameter for simplicity here.

[20] The purpose of the bold font is, simply put, to distinguish between the constructional level and non-constructional level. Let us take, e.g., **2** and 2. What is the difference between them? The former is a construction, the latter is a constructed object. In other words, **4** can be understood as a trivial computation that yields the number 4 as a result. Furthermore, we now switch to the standard TIL notation with square brackets to better distinguish its expressions from those of CTT.

possible worlds. And only an empirical investigation can tell us what its reference is, i.e., whether it is true or false (see Fig. 6). Thus, when we are computing propositions in this sense, we are not searching for truth values, but for the corresponding functions from possible worlds to truth values.

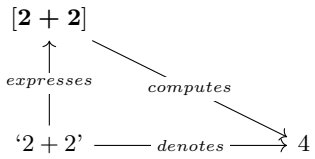

Fig. 5. An example of TIL semantics of non-empirical expressions

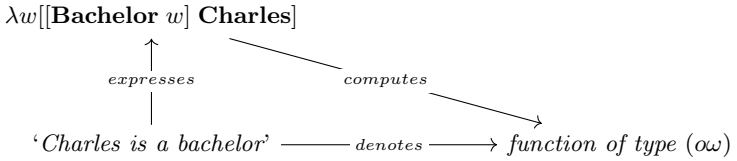

Fig. 6. An example of TIL semantics of empirical expressions

This notion of computation is, however, rather informal as it comes with no general instructions or computation rules telling us how should the evaluation of such computations proceed. Furthermore, it takes us from the level of constructions to a different level, specifically, a level of denotations, i.e., non-constructions (e.g., natural numbers, truth values, individuals).[21] Consequently, this notion of a computation does not seem to be providing a satisfactory ground for explaining the notion of a computable proposition.

Fortunately for us, in TIL, another notion of a computation is identifiable that is much more similar to the notion of a computation in CTT. It comes with explicit computation rules (β-reductions) and the results of these computations do not leave the level of constructions, i.e., the results of such computations have the same type as the computations themselves. Following [17], let us call this new notion of a computation as a *syntactic* ('machine-oriented') notion of a computation and let us refer to the notion of a computation we have considered so far as a *semantic* ('human-oriented') notion of a computation. The main conceptual difference between these two notions is that a semantic computation takes us from constructions to their denotations, while a syntactic computation takes

[21] Strictly speaking, in TIL we can have higher-order constructions that yield lower-order constructions as their denotations, but for simplicity of presentation, we omit these cases here.

us from constructions to other constructions, but never to their denotations. If we add the syntactic computation layer to the original semantic scheme of TIL presented earlier, we get a scheme closer to the one of CTT (see Fig. 7).[22]

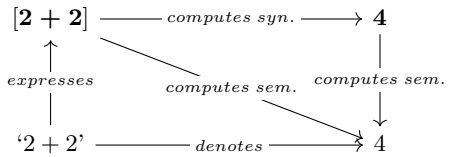

Fig. 7. An example of an expanded TIL semantics of non-empirical expressions

How is this relevant to computable propositions? When we are computing propositions in this syntactic sense, we are not searching for truth values (references) or functions from possible worlds to truth values (denotations) but rather for the simplest possible procedure for evaluating the truth conditions of the corresponding sentence. To better explain this, let us consider the following example. To mirror the initial intuitionistic case of defining $\neg A$ as $A \to \bot$, let us examine a classical case of defining $A \to B$ as $\neg A \vee B$. In TIL, this could be formalized as: $\lambda A\, B[A \to B] =_{def} \lambda A\, B[\neg A \vee B]$. Analogously to the CTT approach, the implicational proposition $\lambda A\, B[A \to B]$ can be understood as a defined proposition computable to its canonical form $\lambda A\, B[\neg A \vee B]$. The form of the corresponding computation rules would then be similar to those of CTT:[23]

$$\frac{\lambda A\, B[\neg A \vee B] \;\Rightarrow\; \lambda A\, B[\neg A \vee B]}{\lambda A\, B[A \to B] \;\Rightarrow\; \lambda A\, B[\neg A \vee B]} \to comp$$

The same approach could also be applied to empirical cases. For example, when we compute the proposition $\lambda w[[\mathbf{Bachelor}\ w]\ \mathbf{Charles}]$ in this syntactic sense, the result we should expect is not a truth value, but rather the canonical form of the procedure for determining the truth value of the corresponding sentence, which might be, e.g., set by the following definition (**B** is an abbreviation of **Bachelor**, **C** of **Charles**, etc.):

$$\lambda w[[\mathbf{B}\ w]\ \mathbf{C}] =_{def} \lambda w[\lambda x[[[\mathbf{M}\ w]\ x] \wedge [\neg[\mathbf{Mar}\ w]\ x]]\ \mathbf{C}]$$

which is, arguably, a more perspicuous test procedure (assuming it is simpler to check whether someone is a man and unmarried than that they are a bachelor).

[22] Note that in comparison with CTT's semantic scheme, here we have three levels of objects: syntactic (the expression '2 + 2'), semantic (the construction [**2 + 2**]), and denotational (the natural number 4). Recall that in CTT, there are only two levels: syntactic and semantic. In other words, in TIL the semantic level is distinguished further into constructional and denotational levels.

[23] This is only a sketch, for proper accounts of definitions/computation rules in TIL, see, e.g., [7], section *2.2.2 Concepts and definitions* or [22], section *3.2 Matches, sequents and rules.*

The computation rule will then be as follows (to save some space, let M represent the proposition $\lambda w[\lambda x[[[\mathbf{M}\ w]\ x] \wedge [\neg[\mathbf{Mar}\ w]\ x]]\ \mathbf{C}]$):

$$\frac{M\ \Rightarrow\ M}{\lambda w[[\mathbf{B}\ w]\ \mathbf{C}]\ \Rightarrow\ M}$$

So, how can we compute with propositions in TIL? Analogously to CTT, we can approach this issue through the idea of defined and primitive propositions. However, in TIL the key difference between these two kinds of propositions will not be the presence or absence of direct proof conditions, but rather whether or not they contain primitive or derived concepts with respect to some conceptual system. In other words, by a canonical proposition we will understand a proposition that contains no derived concepts.

What are conceptual systems? Conceptual systems are, roughly put, applied instances of TIL tailored for a specific purpose (see [7], [23]). Thus, we can have, e.g., conceptual systems for propositional logic, conceptual systems for predicate logic, conceptual systems for reasoning about multiagent systems, etc. Formally, a conceptual system is a tuple $\langle Pr, Type, Var, C, Der\rangle$ where Pr is a finite class of primitive concepts $P_1, \ldots, P_k$, i.e., basic objects of the system, $Type$ is an infinite class of types generated over a finite collection of base types (e.g., o, ι, ν for truth values, individuals, and natural numbers, respectively), Var is an infinite set of variables, countably infinite for each type from $Type$, C is the definition of kinds of basic TIL constructions, and Der is an infinite class of compound concepts derived from Pr and Var utilizing C. From this perspective, assuming a conceptual system CS_1 where $\mathbf{B}$ and $\mathbf{C}$ are simple, further undefined concepts, i.e., $Pr_1 = \{\mathbf{B}, \mathbf{C}\}$, the proposition $\lambda w[[\mathbf{B}\ w]\ \mathbf{C}]$ would be considered as a canonical one. However, assuming some other conceptual system CS_2 where $\mathbf{B}$ is treated as a derived concept with $Pr_2 = \{\mathbf{M}, \mathbf{Mar}, \wedge, \neg\}$, then the proposition $\lambda w[[\mathbf{B}\ w]\ \mathbf{C}]$ will be non-canonical and computable to the canonical proposition $\lambda w[\lambda x[[[\mathbf{M}\ w]\ x] \wedge [\neg[\mathbf{Mar}\ w]\ x]]\ \mathbf{C}]$ (see Fig. 8), which represents the most direct procedure for evaluating the truth conditions of the corresponding sentence (in the given conceptual system). For our logical example, we could use a conceptual system CS_3 with $Pr_3 = \{\neg, \vee\}$, which would render the proposition $\lambda A\, B[A \rightarrow B]$ non-canonical and the proposition $\lambda A\, B[\neg A \vee B]$ canonical. Thus, the status of canonicity of propositions will depend on the choice of the underlying conceptual system.

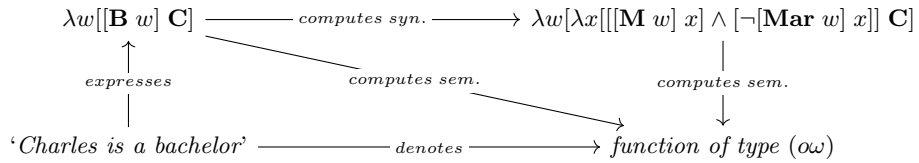

Fig. 8. An example of an expanded TIL semantics of empirical expressions

To conclude, in TIL we can approach the notion of a computable proposition via the notion of a (hyper)proposition, i.e., a construction that yields upon execution a function from possible worlds to truth values. This explication rests on three main principles: distinguishing between denotations and referents of empirical sentences, discerning between syntactic and semantic notions of computations, and capturing the notion of syntactic computation in terms of a reduction of non-canonical proposition to canonical ones within a scope of a given conceptual system.

4 Conclusion

In this paper, we have discussed two computational approaches to the semantics of empirical sentences that are based on the Fregean sense-denotation distinction. However, since the truth values – denotations in Frege's approach – of sentences cannot be in general computed, these approaches had to modify the original scheme by changing what should be regarded as a denotation of an empirical sentence. Both approaches agree that it cannot be truth values and propose that it should be propositions. However, their respective notions of propositions differ. When a CTT-based approach proposes that a denotation of an empirical sentence is a proposition, what is meant is a canonical proposition and a proposition is understood intuitionistically, i.e., as a constructive set of its truth makers. On the other hand, when a TIL-based approach proposes that a denotation of an empirical sentence is a proposition, a proposition is understood as a function from possible worlds to truth values. Thus, in CTT the relation between meaning and denotation holds between objects of different types than in TIL. In the former it holds between propositions, in the latter it holds between propositions and functions.

This is not the only difference. Their respective notions of denoting understood as the relation between sense and denotation also diverge. CTT essentially identifies the notion of denoting with the notion of effective (syntactic) computation. In TIL, however, denoting is rather identified with the notion of constructing, i.e., the notion of not necessarily effective (semantic) computation. However, that does not mean that we cannot make sense of the notion of effectively (syntactically) computable propositions in TIL. It just means that if we compute with propositions in this sense, we can never get to their denotations, just to their canonical forms. This is in contrast to CTT, where canonical propositions and denotations are identified. The relation between propositions and their corresponding canonical forms in TIL is best seen as analogous to the same distinction in CTT, i.e., it should be viewed in terms of effective (syntactic) computability based on the underlying notion of definitional equality. So, from the TIL perspective, CTT conflates the notions of semantic and syntactic computability. Or, from the CTT perspective, TIL muddies the notion of computing by splitting it into two further notions of syntactic and semantic computability. It remains, however, an open question which of these approaches might generally prove to be more productive when dealing with natural language analysis.

To conclude, the key difference between CTT's and TIL's understanding of meaning of empirical sentences, i.e., computable propositions, lies in their respective ideas of what should constitute their corresponding denotations and how we should be able to reach them. CTT identifies denotations with canonical propositions, while TIL keeps them separate. In CTT, denoting has to be an effective procedure, while no such requirement is present in TIL. However, even though in TIL the procedure of getting from a proposition to its denotation is ineffective, the process of getting from a proposition to its canonical form is effective.

References

1. Brouwer, L.E.J.: Over de Grondslagen der Wiskunde. Ph.D. thesis, Universiteit van Amsterdam (1907)
2. Chatzikyriakidis, S., Luo, Z.: Adjectival and Adverbial Modification: The View from Modern Type Theories. Journal of Logic, Language and Information **26**(1), 45–88 (2017). https://doi.org/10.1007/s10849-017-9246-2
3. Church, A.: A formulation of the simple theory of types. The Journal of Symbolic Logic **5**(02), 56–68 (1940). https://doi.org/10.2307/2266170
4. Curry, H.: Functionality in Combinatory Logic. Proceedings of the National Academy of Sciences **20**, 584–590 (1934). https://doi.org/10.1073/pnas.20.11.584.
5. Dummett, M.: The Philosophical Basis of Intuitionistic Logic. Studies in Logic and the Foundations of Mathematics **80**(C), 5–40 (1 1975). https://doi.org/10.1016/S0049-237X(08)71941-4
6. Dummett, M.: The Logical Basis of Metaphysics. Duckworth, London (1991)
7. Duží, M., Jespersen, B., Materna, P.: Procedural Semantics for Hyperintensional Logic: Foundations and Applications of Transparent Intensional Logic. Logic, Epistemology, and the Unity of Science, Springer, Dordrecht (2010). https://doi.org/https://doi.org/10.1007/978-90-481-8812-3
8. Gentzen, G.: Untersuchungen über das logische Schließen. I. Mathematische Zeitschrift **39**(1), 176–210 (1935). https://doi.org/10.1007/BF01201353
9. Granström, J.G.: Treatise on Intuitionistic Type Theory. Logic, Epistemology, and the Unity of Science, Springer, Dordrecht (2011)
10. Howard, W.A.: The formulae-as-types notion of construction. In: Curry, H.B., Hindley, J.R., Seldin, J.P. (eds.) To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism. Academic Press, London (1980)
11. Landin, P.J.: Correspondence Between ALGOL 60 and Church's Lambda-notation: Part I. Communications of the ACM **8**(2), 89–101 (1965). https://doi.org/10.1145/363744.363749
12. Luo, Z.: Formal semantics in modern type theories with coercive subtyping. Linguistics and Philosophy **35**(6), 491–513 (2012). https://doi.org/10.1007/s10988-013-9126-4
13. Martin-Löf, P.: Constructive Mathematics and Computer Programming. In: Cohen, J.L., et al., (eds.) Logic, Methodology and Philosophy of Science VI, 1979. pp. 153–175. North-Holland, Amsterdam (1982)
14. Martin-Löf, P.: Intuitionistic type theory: Notes by Giovanni Sambin of a series of lectures given in Padua, June 1980. Bibliopolis, Napoli (1984)
15. Martin-Löf, P.: The sense/reference distinction in constructive semantics (manuscript) (2001)

16. Nordström, B., Petersson, K., Smith, J.M.: Programming in Martin-Löf's type theory: an introduction. Clarendon Press, Oxford (1990)
17. Pezlar, I.: On Two Notions of Computation in Transparent Intensional Logic. Axiomathes **29**(2) (2019). https://doi.org/10.1007/s10516-018-9401-7
18. Pezlar, I.: Algorithmic Theories of Problems. A Constructive and a Non-Constructive Approach. Logic and Logical Philosophy **26**(4), 473–508 (2017). https://doi.org/https://doi.org/10.12775/LLP.2017.010
19. Prawitz, D.: Meaning Approached Via Proofs. Synthese **148**(3), 507–524 (2006). https://doi.org/10.1007/s11229-004-6295-2
20. Primiero, G.: Information and Knowledge. Springer, Dordrecht (2008)
21. Raclavský, J., Kuchyňka, P., Pezlar, I.: Transparentní intenzionální logika jako characteristica universalis a calculus ratiocinator. Masaryk University Press (Munipress), Brno (2015)
22. Raclavský, J.: Belief Attitudes, Fine-Grained Hyperintensionality and Type-Theoretic Logic. College Publications, London (2020)
23. Raclavský, J., Kuchyňka, P.: Conceptual and derivation systems. Logic and Logical Philosophy **20**(1-2) (2011). https://doi.org/10.12775/LLP.2011.008
24. Rahman, S., McConaughey, Z., Klev, A., Clerbout, N.: A Brief Introduction to Constructive Type Theory. In: Immanent Reasoning or Equality in Action, pp. 17–55. Springer, Cham (2018). https://doi.org/https://doi.org/10.1007/978-3-319-91149-6_2
25. Ranta, A.: Type-theoretical Grammar. Clarendon Press, Oxford (1994)
26. Stovall, P.: Proof-Theoretic Semantics and the Interpretation of Atomic Sentences. In: Sedlár, I., Blicha, M. (eds.) The Logica Yearbook 2019. College Publications, London (2020)
27. Sundholm, G.: Proof Theory and Meaning. In: Gabbay, D., Guenthner, F. (eds.) Handbook of Philosophical Logic, vol. 166, pp. 471–506. Springer, Dordrecht (1986). https://doi.org/10.1007/978-94-009-5203-4_8
28. Tichý, P.: Intension in terms of Turing machines. Studia Logica **24**(1), 7–21 (12 1969). https://doi.org/10.1007/BF02134290
29. Tichý, P.: Constructions. Philosophy of Science **53**(4), 514–534 (1986). https://doi.org/https://doi.org/10.1086/289338
30. Tichý, P.: The Foundations of Frege's Logic. de Gruyter, Berlin (1988)
31. Wadler, P.: Propositions as Types. Communications of the ACM **58**(12), 75–84 (2015). https://doi.org/https://doi.org/10.1145/2699407
32. Więckowski, B.: A Constructive Type-Theoretical Formalism for the Interpretation of Subatomically Sensitive Natural Language Constructions. Studia Logica **100**(4), 815–853 (2012). https://doi.org/10.1007/s11225-012-9431-x