

Noname manuscript No.
(will be inserted by the editor)

Type Polymorphism, Natural Language Semantics, and TIL

Ivo Pezlar

Received: date / Accepted: date

Abstract Transparent intensional logic (TIL) is a well-explored type-theoretical framework for semantics of natural language. However, its treatment of polymorphic functions, which are essential for the analysis of various natural language phenomena, is still underdeveloped. In this paper, we address this issue and propose an extension of TIL that introduces polymorphism via type variables ranging over types and generalized variables ranging over constructions and types. Furthermore, we offer an analysis of sentences involving non-specific notional attitudes of the general form ‘A considers (believes, desires, wants, seeks, . . .) something’.

Keywords notional attitudes · polymorphism · type theory · transparent intensional logic

1 Introduction

Analysis of notional attitudes exemplified by the following cases:¹

- I am thinking of Pegasus.
- Ponce de Leon searched for the fountain of youth.
- Schliemann sought the site of Troy.
- Ctesias is hunting unicorns.

has a long tradition in the semantic analysis of natural language. In this paper, we will be interested in a more general class of these sentences that can be obtained

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

Ivo Pezlar
Czech Academy of Sciences, Institute of Philosophy
Jilská 1, 110 00 Prague 1
Tel.: +420 221 183 348
ORCID: 0000-0003-1965-2159
E-mail: pezlar@flu.cas.cz

¹ Examples taken from Church (1951), p. 111, Church (1956), p. 8, and Quine (1956), p. 177.

by replacing the concrete objects² of the respective attitudes (thinking, searching, hunting, etc.) with non-specific ones:

- I am thinking of *something*.
- Ponce de Leon searched for *something*.
- Schliemann sought the site of *something*.
- Ctesias is hunting *something*.

This represents a specific challenge to type-theoretic approaches to natural language semantics for obvious reasons: since we cannot know in advance what type of object is being referred to by ‘something’ (e.g., I can be thinking of people, trees, houses, places, . . . , numbers, sets, types, functions, etc.) we have to adopt polymorphic properties that can be applied to an unspecified range of objects. In other words, we have to introduce polymorphism into the system. There are two general approaches to how to treat polymorphism which we might call syntactic polymorphism and semantic polymorphism.³

The first approach treats polymorphism just as a meta-language tool for referring to a wide range of similar functions. Usually a meta-language symbol is introduced, e.g., ‘ α ’, that ranges over all the possible variations. For example, we might encounter a function *isEqual* of the type $(\alpha \rightarrow \alpha \rightarrow Bool)$ which takes two objects of type α as arguments and returns *True* if they are equal and *False* otherwise. This approach entails, among other things, that we have similar yet different functions specific for each type. For example, if we are dealing with the equality of individuals, we get the function *isEqual_Ind* of the type $(Ind \rightarrow Ind \rightarrow Bool)$, if we are dealing with natural numbers, we get the function *isEqual_Nat* of the type $(Nat \rightarrow Nat \rightarrow Bool)$, etc. But, generally speaking, there is no ‘the’ equality function, only its multiple instances tailored for each type. Hence, $(\alpha \rightarrow \alpha \rightarrow Bool)$ is not a type, just a notation shorthand for various types.

The second approach treats polymorphism as a semantic feature of the system itself. In practice, this means extending the definition of types and allowing the type variables to appear on the object level of the framework. For example, using this approach the type $(\alpha \rightarrow \alpha \rightarrow Bool)$ is a proper type of the system (or more specifically, type constructor/type-valued function) and the general properly polymorphic function *isEqual* can thus be entertained and applied to objects of any type. For instance, if we apply the type constructor $(\alpha \rightarrow \alpha \rightarrow Bool)$ to the type *Ind*, we get the type $(Ind \rightarrow Ind \rightarrow Bool)$, etc.

In this paper, we investigate the issue of non-specific notional attitudes from the perspective of transparent intensional logic (TIL; Tichý (1988)). TIL is a well-explored type-theoretic framework for the semantics of natural language, however, it

² Similarly to Church (1951), we consider the objects of notional attitudes to be abstract entities rather than mental ones. For an alternative approach, compare, e.g., with Moltmann (2008) and recently Moltmann (2017) utilizing truthmaker semantics.

³ Fox and Lappin (2005) relies on a similar distinction between schematic and genuine polymorphism, Duží (1993) uses the terms weak and strong polymorphism, and others can no doubt be found as well. It is worth noting that from the perspective of Cardelli and Wegner (1985) the classification (based on Strachey (2000)) of both these kinds of polymorphism would fall under their category of universal/parametric polymorphism. In comparison to Cardelli and Wegner (1985), we rely on a rather strict notion of polymorphism, since they also include in the kinds of polymorphism, e.g., subtyping and overloading.

relies on the syntactic approach to polymorphism where the type variables are treated just as placeholders. This puts them outside of the semantic framework of TIL and thus provides much less interesting analyses in return. In this paper, we propose an extension of TIL that handles polymorphism semantically via type variables ranging over types and generalized variables ranging over constructions and types. Furthermore, we offer the analysis of sentences involving non-specific notional attitudes of the general form ‘A considers (believes, desires, wants, seeks, ...) something’.

This paper is structured as follows: in Section 2 we present the basics of TIL, in Section 3 we examine closely the treatment of polymorphism and variables in TIL, in Section 4 we introduce type variables and in the final section, Section 5, we introduce generalized variables, offer an extension to the ramified type theory of TIL, and propose a semantic analysis of non-specific notional attitudes.

Remark 1 The problem of polymorphism is closely related to the problem of ‘nominalization’, i.e., the process of the transformation of non-noun phrases (verb phrases, common nouns, ...) into noun phrases and the problem of flexible predicates such as ‘good’, ‘fun’, ‘interesting’, ‘boring’, etc. that are predicable over a variety of types of objects (‘Sleep is good’, ‘Sleeping is good’, ‘To sleep is good’, ‘That they sleep is good’, etc.). Also, as was already observed by Chierchia (1982), type theory in general enforces constraints on the structure of natural language that is not directly observable. In natural language it seems to be the case that we are using ‘one size fits all’ predicates, e.g., ‘is interesting’ can be applied to cakes just as to mathematical results. As Chierchia put it:

A theory [of types] imposes various limitations on the categorial structure of English syntax. As we will see, in type theory properties (like, say, to be fun) have to be ranked differently in the type hierarchy according to whether they are attributed to individuals (as in ‘John is fun’) or to properties (as in, say, ‘to dance is fun’). Such ranking doesn’t seem to have any overt correlate in the syntactic behavior of predicate nominals like fun in natural languages. So, those limitations that type theory imposes on English syntax are likely to turn out to be artificial. (Chierchia (1982), p. 305)

On the other hand, it seems certainly true that in both cases we are encountering different kinds of ‘interestingness’ with various applicability criteria (an interesting cake is probably interesting in a different sense than a mathematical theorem) and many-sorted type-theoretical approaches are well-equipped to handle these differences. Briefly put, although type-theoretical approaches to natural language semantics without a doubt come with their own set of challenges, the benefits of utilizing them (e.g., blocking paradoxical behavior due to violations of the vicious circle principle, avoiding categorial mistakes, etc.) seem to generally outweigh their potential drawbacks.

2 Brief introduction to TIL

Transparent intensional logic (TIL, Tichý (1988), Duží et al. (2010), Raclavský et al. (2015)) is a theory of abstract constructions that utilizes the extended language of

typed lambda calculus with partial functions and focuses on the semantics of natural language.⁴ In spirit, it shares many similarities with Montague semantics but the meanings of natural language expressions are understood rather in terms of algorithms (hyperintensions) than in terms of functions from possible worlds (intensions), which takes it closer to the systems with an interpreted formal syntax such as Martin-Löf's constructive type theory (see, e.g., Pezlar (2017)).

Syntactically, TIL's theory of constructions can be captured using the following four construction terms, namely variables, compositions, closures, and n -executions, respectively:

$$\text{constructions} := x \mid [CC_1 \dots C_m] \mid [\lambda x_1 \dots x_m C] \mid {}^n X$$

where C_i is any construction, X is either a construction or a non-construction (e.g., a truth value, an individual, a number, etc.).⁵

The first three constructions roughly correspond to variables, function applications, and function abstractions as known from λ -calculus. Construction n -execution allows us to either execute constructions to determine what object they construct, if any (if $n > 0$),⁶ or not execute them, i.e., leave them as they are (if $n = 0$).⁷ What constructions construct might depend on a *valuation* v , i.e., an assignment of values to free variables. In that case, we say that they v -construct. If a construction C v -constructs nothing at all, we will say that it is a v -improper construction. Otherwise, we say that C is a v -proper construction. If we have two constructions C_1 and C_2 that v -construct the same object X , or they are both v -improper, we will say that C_1 and C_2 are v -congruent constructions, denoted as $C_1 \cong C_2$. If they are v -congruent for all valuations v , we will say that C_1 and C_2 are *equivalent* constructions, denoted as $C_1 = C_2$.

In standard TIL, there are four atomic types forming the type base B : truth values, individuals, real numbers/time moments, and possible worlds, denoted as o, ι, τ, ω , respectively.⁸ These basic types are then expanded with types of n -th order constructions, denoted as $*_n$. If α and β_1, \dots, β_m are types, then we can form a function type $(\alpha\beta_1 \dots \beta_m)$. Specifically, it is a type of function from the elements of type β_1, \dots, β_m to the elements of type α .⁹

⁴ It is worth noting that, strictly speaking, TIL itself is just an applied instance of Tichý's type theory (which is a modification of Church's type theory) intended for the purposes of logical analysis of natural language (similarly to, e.g., transparent hyperintensional logic (THL) recently employed in Raclavský (2020)).

⁵ For a proper specification, see Appendix 7. For the definition of n -execution, see Pezlar (2018).

⁶ It is important to note that if we allow n -executions with $n \geq 2$, the Church-Rosser theorem is no longer valid in TIL, as was recently demonstrated by Kostelec (2019).

⁷ In a standard TIL presentation, the cases of $n = 0$ (i.e., 0-execution also known as trivialization) and $n > 0$ are strictly kept apart to emphasize their different logical roles, most importantly, 0-execution supplies objects (of any type) for compound constructions, while ($n > 0$)-execution is used for executing constructions. Furthermore, 0-execution can raise a context of a construction up to the hyperintensional level, while, e.g., 2-execution can decrease the context down (see, e.g., Duží and Horák (2019)). For more about TIL and the three kinds of contexts, see Duží et al. (2010), section 2.6.

⁸ TIL is an open-ended framework and other atomic types can be added, e.g., we can add v as the type of natural numbers.

⁹ Note that TIL relies on the Church notation $(\alpha\beta)$ for function types. In a more standard notation, this would be written as $\beta \rightarrow \alpha$. Furthermore, due to the presence of partial functions, we cannot generally

Since constructions can ν -construct other objects or be ν -constructed by other constructions they receive two-dimensional typing: the first dimension is the type of the construction itself, denoted as $C/type$, the second dimension is the type of the object the construction is supposed to construct, denoted as $C : type$. We can also chain these notations to get $C/type_1 : type_2$. For example, $[^0+^05^07]/*_1 : \nu$ which can be read as ‘ $[^0+^05^07]$ is a first-order construction typed to construct natural numbers’.¹⁰

To simplify notation, we denote 0-execution by **boldface** font, with the exception of standard logical and mathematical symbols such as ‘+’, ‘=’, ‘ \forall ’, ‘ \supset ’, etc. which we keep in normal font with 0-execution implicitly assumed. Also, we will use infix notation whenever expected. For example, instead of ‘ $[^0+^05^07]$ ’ we will write ‘**[5 + 7]**’ and instead of ‘ $[^0\supset A B]$ ’ we will write ‘**[A \supset B]**’. Furthermore, we extend the standard notation of closure construction by including explicit typing of variables and omit the outer most brackets whenever possible. Thus, we will write $\lambda x_1 : \alpha_1 \dots x_m : \alpha_m Y$ instead of $[\lambda x_1 \dots x_m Y]$.¹¹

Sample analysis. For example, the sentence:

(s) Alice believes there is a natural number greater than four but smaller than five.

can be analysed as follows (see Fig. 1 for the corresponding type-checking tree):

$$\lambda w : \omega \lambda t : \tau [\mathbf{Believe}_{wt}^* \mathbf{Alice}^0 [\exists x : \nu [x > \mathbf{4}] \wedge [x < \mathbf{5}]]]]$$

Reading this construction from left to right (with some minor simplifications):

- ‘ $\lambda w : \omega \lambda t : \tau$ ’ binds the world and time parameters w and t – which receives the function $\mathbf{Believe}^*$ as two of its four arguments (see below) – and displays their type annotations, i.e., the variable w ranges over objects of type ω (possible worlds) and the variable t ranges over objects of type τ (time moments captured as real numbers),
- **$\mathbf{Believe}^*$** is a construction (specifically a trivialization also known as 0-execution) that constructs the function $\mathbf{Believe}^*$ of type $((\omega \tau) \omega)$, usually shortened as $(\omega \tau) \omega$. The superscript $*$ indicates that this is a constructional/hyperintensional belief (i.e., a belief that a given construction ν -constructs a proposition that is true in a given world and time of evaluation, or that a given construction ν -constructs a truth value *true*, in the case of mathematics and logic), which is different from other types of belief (e.g., a sentential belief, i.e., a belief that a given proposition is true in a given world and time, which would have type $(\omega \tau) \omega$). Informally, $\mathbf{Believe}^*$ is a function that takes a possible world, a specific time moment in that world, an individual, and a construction of a proposition/truth value, and

assume that all multiargument functions can be reduced to a series of functions taking a single argument. In other words, Schönfinkel’s reduction does not hold. For a proof, see Tichý (1982).

¹⁰ In TIL literature the symbol ‘ \rightarrow ’ is used instead of ‘ \supset ’, however, we choose the latter because it leads to a clearer notation once explicit typing is adopted. Also note that the symbol ‘/’ is used for typing annotations of non-constructions as well. For example, if we want to declare that addition function $+$ on natural numbers ν has type $(\nu \nu \nu)$, we can write it as $+ / (\nu \nu \nu)$.

¹¹ It is worth mentioning that Tichý (1988) used explicit subscripts with ‘ λ ’ to indicate the type of the output of the constructed function. For example, $[\lambda_{\alpha x} \mathbf{Odd} x]$.

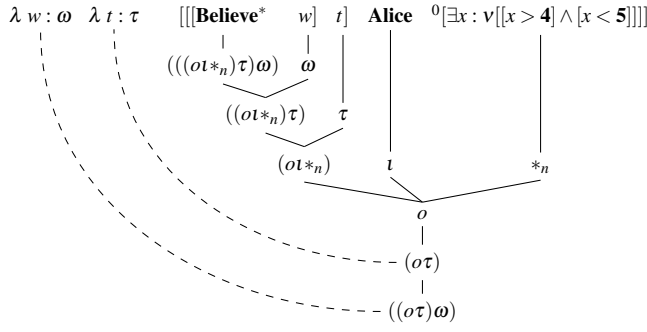


Fig. 1 Type-checking tree

returns *true* if that individual believes that the construction produces (a proposition that takes) the truth value *true*, otherwise *false*.

- ‘ w_t ’ is used as a shorthand for consecutive applications of world and time parameters to the function $Believe^*$. In full, it should be written as: $[Believe^* w] t$,
- **Alice** is a construction (a trivialization, see above) that constructs the individual *Alice* of type ι .
- $\exists x : v[x > 4] \wedge [x < 5]]$ is a higher-order construction (due to trivialization) that constructs a first-order construction, namely the composition construction $\exists x : v[x > 4] \wedge [x < 5]]$. ‘ $\exists x : v \dots$ ’ is an abbreviation for ‘ $\exists \lambda x : v[\dots]$ ’ where \exists is the existential quantifier of type $(o(o\nu))$ applied to the class of numbers $\lambda x : v[\dots]$ and returns *true* if the class is non-empty, otherwise *false*.

3 Polymorphism in TIL

3.1 The Problem

Let us try to analyse our initial examples involving notional attitudes. Since all of the sentences share an analogous form ‘[subject] + [notional attitude] + [object]’, we will focus only on the first sentence ‘I am thinking of something’. Furthermore, to slightly simplify it, we replace the personal pronoun ‘I’ with a proper noun ‘Alice’ and obtain:

(1’) Alice is thinking of something.

In TIL, we can analyze it, e.g., in the following manner:

$$\lambda w : \omega \lambda t : \tau \exists x : ? [Think_{wt} Alice x]$$

which constructs the proposition (a function from possible worlds w and time moments t to truth values) that is true if there is something Alice is thinking of, otherwise it is false. However, we encounter problems once we start checking types of the involved objects. Namely, it is unclear over what type of objects should the variable x range and, consequently, what the type of the function $Think$ should be.

For specific instances this is straightforward. For example, assuming Alice is thinking of some individual (e.g., ‘Bob’), we can simply restrict the variable x to the type t , i.e., $x : t$, and the type of the function *Think* would become $(otl)_{\tau\omega}$, i.e., $Think/(otl)_{\tau\omega}$. If she is thinking of some property of natural numbers (e.g., ‘being prime’), we would have $x : (oV)$ and $Think/(ot(oV))_{\tau\omega}$, etc. But in more general cases, such as ‘Alice is thinking of something’, where no specific type of object of Alice’s contemplation is known or can be known, difficulties arise. In particular, we have no appropriate type to assign to the variable x , which we symbolized by ‘?’ in the corresponding analysis above. Thus, we need variables that range over objects of *any* type, not just over objects of some specific type.¹²

3.2 TIL: Current State

function/symbol	type	description
\forall, \exists	$(o(o\alpha))$	universal and existential quantifiers
<i>Sing</i>	$(\alpha(o\alpha))$	singulariser (a function taking a singleton set and returning its only member)
<i>Card</i>	$(\tau(o\alpha))$	cardinality function
<i>All, Some, No</i>	$((o(o\alpha))(o\alpha))$	restricted quantifiers
$=$	$(o\alpha\alpha)$	identity function
<i>Tr</i>	$(*_n\alpha)$	trivialization function
<i>Exist</i>	$(o\alpha_{\tau\omega})_{\tau\omega}$	existence as a property of an intension
<i>Seek</i>	$(ot\alpha_{\tau\omega})_{\tau\omega}$	notional/intentional attitude of seeking

Table 1 Examples of polymorphic functions of TIL

In TIL, we can encounter many type-theoretically polymorphic functions (see Table 1), but what exactly are they? In Duží et al. (2010) we are given the following informal explanation in a footnote:

By ‘type-theoretically polymorphous functions’ we mean a set of functions that are defined and thus behave in the same way, independently of their type. For instance, any member of the set of functions *Cardinality* associates a finite class with the number of its elements. Hence this definition is polymorphous; there are actually infinitely many cardinality functions, one for each type: *Card1*/ $(\tau(o\alpha))$ – the number of a set of individuals, *Card2*/ $(\tau(o\tau))$ – the number of a set of numbers, etc., which we indicate by using a type variable α in the type of *Cardinality*/ $(\tau(o\alpha))$. (Duží et al. (2010), p. 86)

From the above description it is clear that the symbol ‘ α ’ plays the role of type variable. Thus, we can view type-theoretically polymorphic functions as functions whose types involve at least one occurrence of a type variable. This seems straightforward, unfortunately, it is not obvious what exactly a type variable from the TIL perspective is. The only available variables in standard TIL are variables understood as construction and these do not range over types themselves, only over objects of specific types.

So how should we understand type variables? As already discussed above, there are essentially only two options how to approach them:

¹² The need for type variables was also discussed in Raclavský et al. (2015).

1. *Type variables as metavariables*: This is the easier, but also the less interesting answer. Type variables become just syntactic placeholders, which means that they are beyond the semantic framework of TIL.
2. *Type variables as variables*: This is the more attractive answer, yet also the more demanding. Since ‘standard’ variables are considered as constructions, it seems reasonable to consider type variables as some kind of constructions as well. This way, type variables would enrich the semantic framework of TIL. However, a new kind of construction ν -constructing types would have to be introduced.

Regarding type variables as metavariables seems to be the implicit approach taken in [Duží et al. \(2010\)](#).¹³ Type variable ‘ α ’ is not considered as a part of TIL’s ramified type theory, rather as a metalanguage device. See, e.g.,:

Remark. α -sets of elements of type α are modelled by their characteristic functions. Thus they are $(o\alpha)$ -objects. For instance, a set of individuals is an object of type (oi) , a set of real numbers is an object of type $(o\tau)$, a set of couples of real numbers (i.e., a binary relation over reals) is an object of type $(o\tau\tau)$. ([Duží et al. \(2010\)](#), p. 44).

It gets the job done, however, it is somewhat undesirable given the fact that one of the basic credos of TIL is that it has no use for metalanguage: “TIL does not need a metalanguage, since [it has] a ramified type hierarchy instead” (see [Duží et al. \(2010\)](#), p. 55).

Understanding type variables as constructions seems to be the approach most in line with TIL semantic doctrine. After all, type variables are *variables*, and variables are treated as constructions in TIL, thus type variables should be treated as constructions as well.¹⁴ Furthermore, this would allow us to regard type variables as proper entities of the semantic framework of TIL (with appropriately extended ramified type theory). First, however, we explore TIL’s standard notion of *variables as constructions* developed by [Tichý \(1988\)](#) (pp. 60–62) and then we propose how to extend it to cover even type variables and generalized variables.

3.3 Variables

As discussed above, variables are basic constructions of TIL and they construct objects dependently on a valuation function ν . This means, among other things, that variables are taken as extra-linguistic entities, whose procedural content consists in retrieving values, i.e., variables are not just symbolic placeholders. To better understand this approach to variables, we will examine Tichý’s original definition from [Tichý \(1988\)](#).¹⁵

Definition 1 (Variable).

Let R be an arbitrary non-empty collection. By an *R-sequence* we shall understand any infinite sequence:

¹³ TIL-Script, the software variant of TIL, also treats type variables simply as syntactic placeholders and whenever they appear type checking is simply skipped. See e.g., [Duží and Fait \(2019\)](#), p. 223.

¹⁴ An alternative syntactic rule-based approach is, however, also possible. For a brief sketch, see Appendix 7.3.

¹⁵ Parts of the exposition of the variable construction follow my PhD thesis, see [Pezlar \(2016\)](#), pp. 6–8.

(s) $X_1, X_2, X_3, X_4, \dots$

(with or without repetitions) of members of R .

For any natural number n let $|R|_n$ be the (incomplete) [i.e., depending on external sequence] construction which consists in retrieving the n -th member of an R -sequence. Constructions of this form will be called *variables*. (Tichý (1988), p. 60)

Definition 2 (Valuation).

In [a] more general case [where various logical types are needed] we shall need whole arrays of sequences containing an R^i -sequence for each type R^i . We shall call such arrays *valuations*. Thus, where $R^1, R^2, R^3, R^4, \dots$ is an enumeration (without repetitions) of all the types, a valuation is an array of the form

$$(v) \begin{array}{l} X_1^1, X_2^1, X_3^1, X_4^1, \dots \\ X_1^2, X_2^2, X_3^2, X_4^2, \dots \\ X_1^3, X_2^3, X_3^3, X_4^3, \dots \\ X_1^4, X_2^4, X_3^4, X_4^4, \dots \\ \vdots \end{array}$$

where $X_1^i, X_2^i, X_3^i, X_4^i, \dots$ is an R^i -sequence. Let v be this valuation. Relative to v , the variable $|R^i|_n$ constructs X_n^i , i.e., the n -th term of the R^i -sequence occurring in v . (Tichý (1988), p. 61)

Thus, a valuation array is, simply put, a sequence of sequences – to make this more apparent, we will utilize the following list notation:

$$v = \left[\begin{array}{l} [X_1^1, X_2^1, X_3^1, X_4^1, \dots], \\ [X_1^2, X_2^2, X_3^2, X_4^2, \dots], \\ [X_1^3, X_2^3, X_3^3, X_4^3, \dots], \\ [X_1^4, X_2^4, X_3^4, X_4^4, \dots], \\ \vdots \end{array} \right]$$

For example, let us have the following toy valuation array (or valuation for short) v_1 :

$$v_1 = \left[\begin{array}{l} [true_1^1, false_2^1], \\ [1_1^2, 2_2^2, 3_2^2, \dots], \\ [Alice_1^3, Bob_2^3, Cecil_3^3, Dana_4^3, \dots], \\ \vdots \end{array} \right]$$

In this case, the variable $|R^1|_2$ receives through v_1 the value *false*, $|R^3|_1$ retrieves *Alice*, etc. So what is a variable from Tichý's point of view? It is essentially a search and retrieve mechanism that takes the coordinates $\langle i, n \rangle$ and some valuation array v_m as input and returns the object located at that position as output.

From a type-theoretical perspective, the valuation array v_1 looks as follows:

$$v_{1t} = \left[\begin{array}{l} [o, o], \\ [v, v, v, \dots], \\ [t, t, t, t, \dots], \\ \vdots \end{array} \right]$$

where o , v , t represent types of truth values, natural numbers, and individuals, respectively.¹⁶ But what about the variables $|R^i|_n$ themselves? What type do they belong to? Tichý gives the following answer:

(c_n i) Let τ be any *type of order n over B* . Every variable ranging over τ is a *construction of order n over B* . If X is of (i.e., belongs to) type τ then 0X , 1X , and 2X are *constructions of order n over B* .

Let $*_n$ be the collection of *constructions of order n over B* . (Tichý (1988), p. 61)

Let types o , v , t from our example above constitute the type base B_1 . Then, by definition 16.1.–1.(t_1 i) (see Tichý (1988), p. 66) o , v , t are types of order 1. Further, let us have variables $|R^1|_1$, $|R^2|_1$, $|R^3|_1$ that range over types o , v , t , respectively. By definition 16.1.–2.(c_n i) above, they are constructions of order 1 and hence they all belong to the same type $*_1$.

Due to the dependency of variables on valuation arrays, Tichý describes them as *incomplete* (or heteronomous) constructions (Tichý (1988), p. 60). In other words, variables need some ‘external’ building material, in this case valuation arrays, to be able to construct anything. This might sound odd at first, but remember that in TIL, variables are not placeholders for values but general procedures for fetching them, so it makes sense that they need some independent database to get their values from.

Also note that if we want to learn what type of objects variables are v -constructing, we have to check not the type of the variables themselves, but the type of objects they are *typed* to construct within the ramified type hierarchy (see Appendix 7). For example, from the information given above, we can infer that variable $|R^2|_1$ is typed to v -construct objects of type v (the superscript ‘2’ points our attention to the second row (R -sequence) of v_1 , which is the type of natural numbers), i.e., $|R^2|_1/*_1 : v$, which can be read as “ $|R^2|_1$ is a first-order construction (a variable) that is typed to v -construct an object of type v ”. In other words, it is the parameter i in conjunction with some specific valuation v that sets the range of the variable $|R^i|_n$ by pointing it to a specific row of v , which represents some type.

In the rest of the paper, we will write variables $|R^i|_n$, $|R^i|_{n+1}$, ... simply as x, y, \dots to use a less cluttered notation.¹⁷

4 Type Variables

Recall that standard variables are restricted to objects of a certain type. For example, a variable $x : o$ is restricted to the type of truth values. In other words, standard variables are locked to a single row in our valuation arrays and they v -construct only the objects of this specific row. However, we want them to range not over objects but over types.

¹⁶ We obtain v_{1i} from v_1 by replacing objects in the array by their corresponding types, e.g., *true* is replaced by o , etc.

¹⁷ Note that when we type a variable to a certain type (i.e., specify its range), we are essentially just assigning a concrete number to i . If $i = 1$, then the variable at hand is typed to construct truth values o , if $i = 2$, then it constructs natural numbers, etc.

Thus, type variables should, in a way, range over columns of v_t as opposed to standard variables that range over rows of v .

The definitions of variables and valuations presented above lead naturally to the following definition of type variables:

Definition 3 (Type variable).

Let T be a collection of base types together with types of constructions and function types generated inductively from them (see Definition 16.1. in Tichý (1988)). By a T -sequence we shall understand any infinite sequence

(s') $t_1, t_2, t_3, t_4, \dots$

(with or without repetitions) of members of T .

For any natural number n let $|T|_n$ be the (incomplete) construction which consists in retrieving the n -th member of an T -sequence. Constructions of this form will be called *type variables*.

Each valuation array v determines a T -sequence. It consists of the collection of types to which the objects of v belong and these types are linearly ordered in the same way as the rows of v to which they correspond are. For example, in the case of v_{1t} the T -sequence would be $[o, v, t, \dots]$, since the first three rows of v correspond to types of truth values, natural numbers, and individuals, respectively.¹⁸

Thus, the range of a type variable is the collection of all types of ramified type theory of TIL (see Appendix 7) ordered in a T -sequence. For example, the type variable $|T^1|$ will v -construct the type o (i.e., the type represented by the first row of the valuation-array v_{1t}), $|T^2|$ will v -construct v , etc.

It is worth emphasizing that we can keep the original structure of valuation arrays from **Definition 2**, we just need to adjust the ‘crawl’ mechanism accordingly to the above explanation. We can do this as follows: first, we will search v_{1t} that can be obtained from the corresponding valuation array v_1 , then: type variable $|T|_n$ v -constructs t_n , i.e., the n -th type of the T -sequence occurring in v_t .¹⁹ Hereinafter, we will write type variables $|T|_n, |T|_{n+1}, \dots$ simply as α, β, \dots .

Once we start treating types as objects, we introduce a new type denoted Type which is essentially the collection of all types. Thus, e.g., o/Type , v/Type , \dots , $*_1/\text{Type}$, etc. Furthermore, note that this type of types is not part of our valuation arrays, hence, we cannot have type variables ranging over it (it is not the case that Type/Type).²⁰

Now, let us return to our initial example, specifically to the analysis of the sentence ‘Alice is thinking of something’, which we already attempted unsuccessfully earlier. With type variables adopted, we could analyze it as follows:

$$\lambda w : \omega \lambda t : \tau \exists \alpha : \text{Type}[\mathbf{Thinks}_{wt} \text{ Alice } \alpha]$$

Note, however, that this analysis is not general enough. It can be interpreted as ‘Alice is thinking of some type’, since α is a *type variable* only. But the original sentence

¹⁸ Thus, this T -sequence is essentially obtained by transposing v_{1t} : from $[[o, o], [v, v, v, \dots], [t, t, t, \dots], \dots]$ we get $[o, v, t, \dots]$.

¹⁹ Strictly speaking, we should be distinguishing between valuation and type valuation, but we conflate them to simplify the presentation.

²⁰ Of course, we could expand valuation arrays to accommodate even them.

was ‘Alice is thinking of something’, which means she might be considering objects of various types, not just various types. Hence, a more general analysis will be necessary. Specifically, note that so far we have treated variables (ranging over constructions and non-constructions) and type variables (ranging over types) separately. Considering two different sorts of variables (valuations, etc.) can be, however, restricting sometimes, as we have just seen above. Thus, in the next section, we introduce a generalized notion of a variable that can range over objects belonging to any type (including the construction types $*_n$) and types as well.

5 Generalized Variables

So far, we have introduced i) standard variables ranging over constructions (and non-constructions) and operating with valuation arrays v and ii) type variables ranging over types and operating with type valuation arrays v_t . It follows that if we want to introduce variables that range over both constructions (and non-constructions) and types, the associated valuation arrays will have to contain the information from both arrays v and v_t . This can be simply done by taking v and enriching it with the corresponding type information contained in v_t . Note that since the types are constant throughout the whole rows of v_t (recall, e.g., v_{1t} with the second row $[v, v, v, \dots]$), we do not need to incorporate the whole rows of v_t into v , we just need to incorporate their first elements, since they already carry all the type information we need. Thus, valuation arrays for generalized variables will still be two-dimensional, but their first element will now be the type of all the subsequent elements.

Definition 4 (Generalized Variable).

Let W be an arbitrary non-empty collection. By a W -array we shall understand any infinite sequence:

$$[X_0, X_1, X_2, X_3, X_4, \dots]$$

such that the first member X_0 of the sequence is a type, while the other members X_n , $n > 0$, are the entities belonging to the type X_0 .

Then, for any natural number n , let $|W|_n$ be the (incomplete) construction that consists in retrieving the member X_n of the given W^i -array, i.e., the element on the coordinates $\langle i, n \rangle$. The construction $|W|_n$ is a *generalised variable*.²¹

Definition 5 (Generalized Valuation).

In a more general case we will need a two-dimensional array containing a W^i -array $[X_0^i, X_1^i, X_2^i, X_3^i, X_4^i, \dots]$ for each type. We shall call such arrays *generalized valuations*. Thus, where $W^1, W^2, W^3, W^4, \dots$ is an enumeration of all

²¹ An earlier version of this paper contained a more complicated definition of a generalized variable and I would like to thank an anonymous reviewer for suggesting a simplification.

the types, a generalized valuation is a two-dimensional array of the form:

$$V = \begin{bmatrix} [X_0^1, X_1^1, X_2^1, X_3^1, \dots], \\ [X_0^2, X_1^2, X_2^2, X_3^2, \dots], \\ [X_0^3, X_1^3, X_2^3, X_3^3, \dots], \\ [X_0^4, X_1^4, X_2^4, X_3^4, \dots], \\ \vdots \end{bmatrix}$$

Let V be this valuation. Relative to V , variable $|W^i|_n$ constructs X_n^i , i.e., the n -th member of the W^i -array occurring in V .

For example, let us have the following generalized toy valuation array V_1 :

$$V_1 = \begin{bmatrix} [o_0^1, true_1^1, false_2^1], \\ [v_0^2, 1_1^2, 2_2^2, 3_3^2, \dots], \\ [t_0^3, Alice_1^3, Bob_2^3, Cecil_3^3, Dana_4^3, \dots], \\ \vdots \end{bmatrix}$$

Then, e.g., the variable $|W^1|_2$ constructs through V_1 as value *false*, but through $|W^1|_0$ it constructs o , $|W^3|_1$ constructs *Alice*, $|W^3|_0$ constructs t , etc.

To keep the notation simple, we will use the letters ‘ x ’, ‘ y ’, ... for generalized variables as well and it should always be clear from the context whether a standard or generalized variable is used.

5.1 Extending Ramified Type Theory

We have already described the mechanism of generalized variables that v -construct an object or a type to which the object belongs and their relation to expanded valuations, which closely mirrors the rationale behind standard variables. Next, we extend the definition of ramified type theory (see Appendix 7) to accommodate them accordingly.

The extension of ramified type theory will involve an introduction of an additional dimension to the ramification that will subsume the standard ramified type theory.²² This will allow us to have (generalized) variables that range over constructions (and non-constructions) as well as over all the types of the ‘lower’ dimensions.

First, we briefly introduce the ramified type theory of TIL. The core of the ramified type theory of TIL is its step-wise stratification of types by orders (for a proper specification, see Appendix 7). We start with first-order types (types inhabited by objects involving no constructions), then we define second-order types which are inhabited by constructions, including variables ranging over first-order types. Naturally, we can go further and introduce third-order types with constructions including variables that v -construct objects belonging to second or first-order types, and so on.

For example, assume our first-order types are natural numbers v and truth values o . With these types, we can, e.g., form a new function type $(o v)$ which is the type of properties of natural numbers. As an example of a function of this type, consider the

²² A similar extension of ramified type theory was already briefly discussed in Duží (1993).

function *Prime* that takes a natural number and returns *true* if it is a prime number and *false* otherwise. Thus, e.g., the construction **[Prime 3]** constructs *true* and **[Prime 4]** constructs *false*. Note that *Prime* is still an object of the first-order type.

Next, assume we introduce a variable x ranging over natural numbers, i.e., $x : \nu$. Now, we can form a construction **[Prime x]** which constructs *true* if the variable x gets assigned a prime number, otherwise it constructs *false*. Since this object contains a variable ranging over objects of the first-order type, it is an object of the second-order type.²³ Furthermore, we can introduce a variable y ranging over objects of the second-order type. For example, assume a higher-order predicate *HasVariable* that takes a construction of the second-order type and constructs *true* if it contains at least one occurrence of a variable (free or bound) and *false* otherwise. Thus, **[HasVariable y]** constructs *false*, if the variable y gets assigned the construction **[Prime 3]** and it returns *true* if it gets assigned the construction **[Prime x]**. Note that in this case, the variable y ranges over objects of the second-order type, hence it itself is an object of the third-order types. Analogously, we could introduce even fourth-, fifth-, sixth-order variables and so on.

Now, the limitation of ramified type theory in its current form is that it does not regard types as proper objects, and, consequently, does not allow variables ranging over types. However, this is exactly what we need to adequately analyze sentences containing semantic polymorphism. As mentioned above, we extend the current ramified type theory with a new dimension that will allow us to introduce generalized variables that can range over both constructions (and non-constructions) and types. We start with basic ramified type theory, whose objects (including types) will be assigned a new common type called *kind*, then we introduce variables ranging over all objects of this supertype. Note that this modification mirrors the definition of ramified type theory where we begin with simple type theory and then expand it. Now, we start with ramified type theory and extend it.

Definition 6 (Extended Ramified Type Theory of TIL (eRTT)).

Let *Type* be a base, i.e., a set of types defined by ramified type theory (RTT).

- (k₁i) Every member of *Type* or an entity belonging to that member is a *kind of order 1 over Type*, denoted as \square_1 .
- (C_ki) Let A be any *kind of order k over Type*. Every generalized variable ranging over A is a *construction of kind of order k over Type*. If X is of (i.e., belongs to) kind A but is not itself a member of *Type*, i.e., it is an entity belonging to some member of *Type*, then 0X , 1X , and 2X are *constructions of kind of order k over Type*.
- (C_kii) If $0 < m$ and X_0, X_1, \dots, X_m are *constructions of kind of order k* , then $[X_0 X_1 \dots X_m]$ is a *construction of kind of order k over Type*. If $0 < m$, A is a *kind of order k over Type*, and Y as well as the distinct

²³ Of course, as an anonymous reviewer remarked, the presence of variables is not necessary for forming higher-order objects. For example, the other constructions **[Prime 3]** or **[Prime 4]** also belong to the second-order type even though they do not contain variables ranging over objects of the first-order type. They belong to the second-order type because they are constructions of order 1 that ν -construct objects of a type of order 1 (see Appendix 7, Def. 2). The same holds for constructions belonging to the third-order type, etc. as well.

- variables x_1, \dots, x_m are constructions of kind of order k over Type, then $[\lambda_A x_1 \dots x_m Y]$ is a construction of kind of order k over Type.
- (C_kiii) Nothing is a construction of kind of order k over Type unless it follows from (C_ki) and (C_kii).

Let \square_k ($k > 1$) be the collection of *generalized constructions of kind order k over Type*. The collection of *kinds of order $k + 1$ over Type* is defined as follows:

- (K_{k+1}i) Every *kind of order k* , denoted \square_k ($k \geq 1$), is a *kind of order $k + 1$* .
- (K_{k+1}ii) If $0 < m$ and A, B_1, \dots, B_m are *kinds of order $k + 1$ over Type*, then the collection $(AB_1 \dots B_m)$ of all m -ary (total and partial) mappings from B_1, \dots, B_m to A is also a *kind of order $k + 1$ over Type*.
- (K_{k+1}iii) Nothing is a *kind of order $k + 1$ over Type* unless it follows from (K_{k+1}i) and (K_{k+1}ii).

So, simply put, first-order kinds are types of objects involving no generalized variables or constructions containing them. In other words, a kind of first-order is the type of all members of Type or entities belonging to those members.²⁴ For example, *Alice*/ ι , **Alice**/ \ast_1 , ⁰**Alice**/ \ast_2 , ι /Type are all first-order kinds, i.e., *Alice*, **Alice**, ⁰**Alice**, ι / \square_1 . But, e.g., a generalized variable x ranging over objects of first-order kind has type \square_2 , i.e., $x/\square_2 : \square_1$.

Second-order kinds are types of objects containing generalized variables ranging over first-order kinds and constructions containing such variables. Analogously, we can have third-order kinds with variables ranging over objects of second-order kind and so on. Furthermore, analogously to RTT's cumulativity of types, we have cumulativity of kinds, i.e., objects of kind \square_n are also objects of kind \square_{n+1} .

Extended ramified type theory gives us tools to properly analyze our motivating examples containing instances of semantic polymorphism. As an example, let us try to offer a more appropriate analysis of the simplified variant of (1):

(1') Alice is thinking of something.

All we need to do to properly analyze (1') is to use a generalized variable and adjust the type of the involved function accordingly. The analysis we obtain is as follows:

$$\lambda w : \omega \lambda t : \tau \exists x : \square_1 [\mathbf{Think}_{wt} \mathbf{Alice} x]$$

where **Think** constructs a function of type $(\iota \square_1)_{\tau \omega}$, **Alice** constructs an individual of type ι and variable x ranges over first-order kinds. Within eRTT, the whole construction will receive the type \square_2 (i.e., second-order kind) because it contains a generalized variable. For the corresponding type checking tree, see Fig. 2.²⁵ Analyses of (1), (2), (3), and (4) would proceed analogously.²⁶

²⁴ Note, however, that Type is not a proper object of eRTT, hence, e.g., a generalized variable x can construct ω , but not its type Type.

²⁵ Recall that ' $\exists x : \alpha \dots$ ' is a notation shortcut for ' $[\exists \lambda x : \alpha [\dots]]$ '.

²⁶ For more about a standard TIL-based analysis of notional attitudes such as seeking, finding, etc., see section 5.2 *Notional attitudes* in Duží et al. (2010).

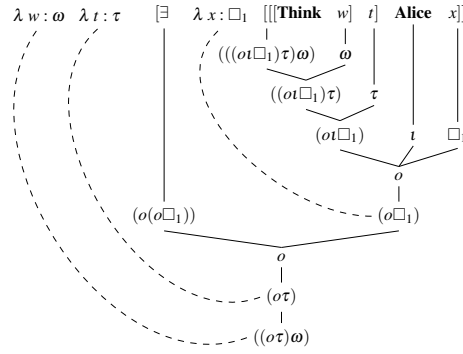


Fig. 2 Type-checking tree

Remark 2 Note that we can now properly type even the inadequate analysis of (1') from earlier which corresponded rather to the sentence 'Alice is thinking of some type'. If we consider type variables only (no generalized variables), then we will have the following types: $Think^T : (ot \text{Type})_{\tau\omega}$ and $\alpha/\square_2 : \text{Type}$. The analysis would then be: $\lambda w : \omega \lambda t : \tau \exists \alpha : \text{Type}[\mathbf{Think}_{w,t} \mathbf{Alice} \alpha]$.

Of course, even this analysis is potentially insufficient. Since Alice can be thinking of anything, she might be thinking, e.g., of the type of all types, kinds, or eRTT itself, etc., which our current analysis cannot cover since we have no variables that could range over these types of objects, so we would have to introduce yet another expansion of eRTT. In other words, it might always turn out that we need some larger type or kind than we currently have. However, for the analysis of everyday natural language phenomena the level of generality provided by eRTT seems to be more than sufficient.

Remark 3 Some might argue that the role of generalized variables is too overloaded and that we are trying to do too much with them.²⁷ Wouldn't it be better to keep standard variables and type variables separate as is, e.g., common in polymorphic lambda calculus? Of course, we could do that. We do not need to go all the way towards generalized variables. We can just stop with the introduction of type variables from the previous section and be satisfied with that (assuming we appropriately extend the definition of ramified type theory). However, recall that our goal was to adequately analyze the sentence (1'). And for that purpose, considering separately standard variables and type variables seem insufficient. In practice, it might turn out that an implementation of generalized variables is indeed an overkill and thus their adoption should be carefully considered, however, semantically speaking, they allow us analyses otherwise unattainable.

²⁷ This issue was raised by an anonymous reviewer.

6 Conclusion

In this paper, we have investigated the treatment of polymorphic functions in TIL, which relies on type variables understood as syntactic placeholders. This approach, however, carries certain disadvantages, most importantly it puts type variables outside of semantic theory of TIL. In practice, this leads, e.g., to our inability to properly analyse sentences involving non-specific notional attitudes such as ‘Alice is thinking of something’, etc.

To alleviate these issues, we have proposed an alternative approach that treats type variables as proper variables in the sense of TIL, i.e., as semantic objects that can v -construct other objects. To address the issue of analysis of non-specific notional attitudes we furthermore introduced generalized variables that act as both standard variables as well as type variables by ranging over both constructions (and non-constructions) and types. This led to the introduction of generalized valuations arrays and to the extended definition of ramified type theory which introduces new ‘large’ types called kinds.

7 Appendix

7.1 Constructions

The original definition of TIL constructions was given by Tichý (1988), here we follow Duží et al. (2010) (an alternative formulation can be found in Raclavský et al. (2015)):

Definition 7 (Constructions).

1. The *variable* x is a *construction* that constructs an object O of the respective type dependently on a valuation. We say that it v -constructs O .
2. Where X is any object, 0X is the *construction trivialization*. It constructs X without any change.
3. The *composition* $[X_0 X_1 \dots X_m]$ is the following *construction*. If X v -constructs a function f of type $(\alpha \beta_1 \dots \beta_m)$ and $X_1 \dots X_m$ v -construct objects b_1, \dots, b_m of types β_1, \dots, β_m , respectively, then the *composition* $[X_0 X_1 \dots X_m]$ v -constructs the value (an object of type α , if any) of f on the tuple-argument $\langle b_1, \dots, b_m \rangle$. Otherwise, it is a *v -improper construction*, i.e., *construction* that does not construct anything.
4. The *closure* $[\lambda x_1 \dots x_m Y]$ is the following *construction*. Let x_1, \dots, x_m be pairwise distinct variables v -constructing objects of types β_1, \dots, β_m and Y a construction v -constructing an object of type α . Then $[\lambda x_1 \dots x_m Y]$ is the *construction closure*. It v -constructs the following function f of type $(\alpha \beta_1 \dots \beta_m)$: let $\langle b_1, \dots, b_m \rangle$ be a tuple of objects of types $\beta_1 \dots \beta_m$, respectively, and v' be a valuation that associates x_i with b_i and is identical to v otherwise. Then the value of function f on argument tuple $\langle b_1, \dots, b_m \rangle$ is the object of type α v' -constructed by Y . If Y is v' -improper, then f is undefined on $\langle b_1, \dots, b_m \rangle$.

5. The *single execution* 1X is the *construction* that either v -constructs the object v -constructed by X or, if X v -constructs nothing, is v -improper.
6. The *double execution* 2X is the following *construction*: let X be any object, the *double execution* 2X is *v -improper* if X is a non-construction or if X does not v -construct a construction or if X v -constructs a v -improper construction. Otherwise, let X v -construct a construction X' and let X' v -construct and object X'' , then 2X v -constructs X'' .
7. Nothing other is a *construction*, unless it follows from 1–6.

7.2 Ramified Type Theory

We follow the specification from Tichý (1988):

Definition 8 (Ramified Type Theory of TIL).

Let B be a base, i.e., a set of atomic types.

1. (t₁i) Every member of B is a *type of order 1 over B*.
 - (t₁ii) If $0 < m$ and $\alpha, \beta_1, \dots, \beta_m$ are *types of order 1 over B*, then the collection $(\alpha\beta_1 \dots \beta_m)$ of all m -ary (total and partial) mappings from β_1, \dots, β_m to α is also a *type of order 1 over B*.
 - (t₁iii) Nothing is a *type of order 1 over B* unless it follows from (t₁i) and (t₁ii).
2. (c_ki) Let α be any *type of order k over B*. Every variable ranging over α is a *construction of order k over B*. If X is of (i.e., belongs to) type α , then 0X , 1X , and 2X are *constructions of order k over B*.
 - (c_kii) If $0 < m$ and X_0, X_1, \dots, X_m are *constructions of order k*, then $[X_0 X_1 \dots X_m]$ is a *construction of order k over B*. If $0 < m$, α is a *type of order k over B*, and Y as well as the distinct variables x_1, \dots, x_m are *constructions of order k over B*, then $[\lambda_\alpha x_1 \dots x_m Y]$ is a *construction of order k over B*.
 - (c_kiii) Nothing is a *construction of order k over B* unless it follows from (c_ki) and (c_kii).

Let $*_k$ be the collection of *constructions of order k over B*. The collection of *types of order k + 1 over B* is defined as follows:

- (t_{k+1}i) $*_k$ and every *type of order k* is a *type of order k + 1*.
- (t_{k+1}ii) If $0 < m$ and $\alpha, \beta_1, \dots, \beta_m$ are *types of order k + 1 over B*, then the collection $(\alpha\beta_1 \dots \beta_m)$ of all m -ary (total and partial) mappings from β_1, \dots, β_m to α is also a *type of order k + 1 over B*.
- (t_{k+1}iii) Nothing is a *type of order k + 1 over B* unless it follows from (t_{k+1}i) and (t_{k+1}ii).

7.3 Types and Rules

In this paper, we have explored a semantic treatment of type variables in TIL. Alternatively, we could attempt a syntactic rule-based analysis as well. However, since

this topic is outside the scope of the present paper, we sketch only the basics of this approach. The key observation is that type variables generally appear in two kinds of judgments in TIL literature:

$$C : *_n \rightarrow \sigma(\alpha)$$

$$X : \sigma(\alpha)$$

which can be read as ‘a construction C belonging to a type $*_n$ is typed to v -construct an object of type σ ’ and ‘a non-construction X belongs to a type α ’, respectively, where σ may contain α as a free type variable. For example:²⁸

$$\begin{aligned} \text{Cardinality}_\tau &: (\tau(o\tau)) \\ \text{Cardinality}_\alpha &: (\tau(o\alpha)) \\ \text{Tr}_\alpha &: (*_n\alpha) \\ X &: *_n \rightarrow ((\alpha\tau)\omega) \\ z &: *_1 \rightarrow \alpha \end{aligned}$$

Furthermore, note that these judgments are composed of three parts: a typed object (i.e., either a construction C or a non-construction X), a typing relation ‘:’ and a type term (i.e., either $*_n \rightarrow \sigma(\alpha)$ in case of constructions or $\sigma(\alpha)$ in case of non-constructions).

Now, as we mentioned above, type terms might depend on some variable α of type Type (the type of all types). We can explicitly capture this dependency by abstracting type terms from α via λ abstractor and get $\lambda\alpha : \text{Type}.*_n \rightarrow \sigma(\alpha)$ and $\lambda\alpha : \text{Type}.\sigma(\alpha)$. Consequently, every free occurrence of α in σ becomes bound in $\lambda\alpha : \text{Type}.*_n \rightarrow \sigma(\alpha)$ and $\lambda\alpha : \text{Type}.\sigma(\alpha)$. The obvious complement to abstraction is, of course, application. Thus, the rules we obtain are as follows:²⁹

$$\frac{\alpha : \text{Type} \vdash C : *_n \rightarrow \sigma(\alpha)}{C : *_n \rightarrow \lambda\alpha : \text{Type}.\sigma(\alpha)} \text{ abs-C} \quad \frac{C : *_n \rightarrow \lambda\alpha : \text{Type}.\sigma(\alpha) \quad \kappa : \text{Type}}{C : *_n \rightarrow \sigma[\kappa/\alpha]} \text{ app-C}$$

$$\frac{\alpha : \text{Type} \vdash X : \sigma(\alpha)}{X : \lambda\alpha : \text{Type}.\sigma(\alpha)} \text{ abs-X} \quad \frac{X : \lambda\alpha : \text{Type}.\sigma(\alpha) \quad \kappa : \text{Type}}{X : \sigma[\kappa/\alpha]} \text{ app-X}$$

where $\sigma[\kappa/\alpha]$ is the result of substituting κ for all occurrences of α in σ .

Note that the premises of abs-C and abs-X rules are hypothetical judgments, i.e., judgments made in a certain context. Thus, we can read $\alpha : \text{Type} \vdash C : *_n \rightarrow \sigma(\alpha)$ as ‘a judgment $C : *_n \rightarrow \sigma(\alpha)$ is assertable given that we have some type term α of type Type ’. Analogously for the hypothetical judgment $\alpha : \text{Type} \vdash X : \sigma(\alpha)$.

These rules can help us to explain syntactically the general process of instantiation of type terms containing type variables to some specific type, a process which was investigated semantically in this paper via the use of expanded valuation arrays.

Acknowledgements I would like to thank Prof. Marie Duží and Prof. Jiří Raclavský for their valuable comments that helped to significantly improve this paper.

²⁸ Examples taken from Duží et al. (2010).

²⁹ Note that in contrast to, e.g., second-order lambda calculus (Girard (1972), Reynolds (1974)), the logic of construction-terms and type-terms remains separated.

References

- Cardelli L, Wegner P (1985) On understanding types, data abstraction, and polymorphism. *ACM Computing Surveys* 17(4):471–523, DOI 10.1145/6041.6042
- Chierchia G (1982) Nominalization and Montague Grammar: A semantics without types for natural languages. *Linguistics and Philosophy* 5(3):303–354, DOI 10.1007/BF00351458
- Church A (1951) The Need for Abstract Entities in Semantic Analysis. *Proceedings of the American Academy of Arts and Sciences* 80(1):100–112
- Church A (1956) *Introduction to Mathematical Logic*. Princeton University Press, Princeton
- Duží M (1993) Frege, notional attitudes, and the problem of polymorphism. In: Stelzner M, Stelzner W (eds) *Logik und Mathematik. Frege-Kolloquium Jena 1993*, de Gruyter, Berlin, pp 314–323
- Duží M, Fait M (2019) Type Checking Algorithm for the TIL-Script Language. In: Endrjukaite T, Dudko A, Jaakkola H, Thalheim B, Kiyoki Y, Yoshida N (eds) *Information Modelling and Knowledge Bases XXX*, frontiers edn, IOS Press, Amsterdam, DOI 10.3233/978-1-61499-933-1-219
- Duží M, Horák A (2019) Hyperintensional Reasoning based on Natural Language Knowledge Base. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* URL <http://arxiv.org/abs/1906.07562>
- Duží M, Jespersen B, Materna P (2010) Procedural Semantics for Hyperintensional Logic: Foundations and Applications of Transparent Intensional Logic. *Logic, Epistemology, and the Unity of Science*, Springer, Dordrecht, DOI <https://doi.org/10.1007/978-90-481-8812-3>
- Fox C, Lappin S (2005) *Foundations of Intensional Semantics*. Blackwell
- Girard JY (1972) *Interprétation fonctionnelle et Élimination des coupure de l'arithmétique d'ordre supérieur*. PhD thesis, Université Paris VII
- Kosterec M (2019) Substitution contradiction, its resolution and the Church-Rosser Theorem in TIL. *Journal of Philosophical Logic* pp 1–13, DOI 10.1007/s10992-019-09514-y
- Moltmann F (2008) Intensional verbs and their intentional objects. *Natural Language Semantics* 16(3):239–270, DOI 10.1007/s11050-008-9031-5
- Moltmann F (2017) Cognitive Products and the Semantics of Attitude Verbs and Deontic Modals. In: Moltmann F, Textor M (eds) *Act-Based Conceptions of Propositional Content*, Oxford University Press, p 408
- Pezlar I (2016) *Investigations into Transparent Intensional Logic: A Rule-based Approach*. PhD thesis, Masaryk University, URL https://is.muni.cz/th/hhga/pezlar_phd_thesis.pdf
- Pezlar I (2017) Algorithmic Theories of Problems. A Constructive and a Non-Constructive Approach. *Logic and Logical Philosophy* 26(4):473–508, DOI <https://doi.org/10.12775/LLP.2017.010>
- Pezlar I (2018) On Two Notions of Computation in Transparent Intensional Logic. *Axiomathes* pp 1–17, DOI <https://doi.org/10.1007/s10516-018-9401-7>
- Quine WVO (1956) Quantifiers and Propositional Attitudes. *Journal of Philosophy* 53(5):177–187, DOI 10.2307/2022451

- Raclavský J (2020) *Belief Attitudes, Fine-Grained Hyperintensionality and Type-Theoretic Logic*. College Publications, London
- Raclavský J, Kuchyňka P, Pezlar I (2015) *Transparentní intenzionální logika jako charakteristica universalis a calculus ratiocinator*. Brno: Masaryk University Press (Munipress)
- Reynolds JC (1974) *Towards a Theory of Type Structure*. In: *Colloquium on Programming*, Paris, 9-11 April 1974, pp 1–18
- Strachey C (2000) *Fundamental Concepts in Programming Languages*. *Higher-Order and Symbolic Computation* 13(1/2):11–49, DOI 10.1023/A:1010000313106
- Tichý P (1982) *Foundations of partial type theory*. *Reports on Mathematical Logic* (14):59–72, DOI 10.1007/BF00370346
- Tichý P (1988) *The Foundations of Frege's Logic*. *Foundations of Communication*, Berlin: de Gruyter