Organon F 24 (1) 2017: 10-36

A Valid Rule of β-conversion for the Logic of Partial Functions[1]

MARIE DUŽÍ – MILOŠ KOSTEREC

ABSTRACT: The goal of this paper is to examine the conditions of validity for the rule of β-conversion in TIL, which is a hyperintensional, typed λ-calculus of partial functions. The rule of β-reduction is a fundamental computational rule of the λ-calculi and functional programming languages. However, it is a well-known fact that the specification of this rule is ambiguous (see, e.g., Plotkin 1975 or Chang & Felleisen 2012). There are two procedurally non-equivalent ways of executing the rule, namely β-conversion '*by name*' and β-conversion '*by value*'. In the λ-calculi conversion by name is usually applied, though it is known that such a conversion is not unconditionally valid when partial functions are involved. If a procedure that is typed to produce an argument value is improper by failing to produce one, conversion by name cannot be validly applied. On the other hand, conversion by value *is* valid even in the case of improperness. Moreover, we show that in a typed λ-calculus the specification of λ-closure is also not unambiguous. There is an interpretation of this specification under which β-reduction by name is not valid even when the argument procedure does *not* fail to produce a value. As a result, we present a universally valid rule of β-reduction *by value*.

[1] Received: 3 October 2016 / Accepted: 19 December 2016

✉ Marie Duží
VSB-Technical University of Ostrava, Department of Computer Science FEI
17. listopadu 15, 708 33 Ostrava, Czech Republic
e-mail: marie.duzi@vsb.cz

✉ Miloš Kosterec
Department of Logic and Methodology of Sciences, Comenius University in Bratislava
Gondova 2, 814 99 Bratislava, Slovak Republic
e-mail: milos.kosterec@gmail.com

KEYWORDS: β-reduction by name – β-reduction by value – λ-Closure – λ_τ-Closure – substitution – validity.

0. Introduction

The goal of this paper is to sort out the conditions of validity of β-conversion in a hyperintensional, partial, typed λ-calculus. Since Transparent Intensional Logic (TIL) is such a system, we will examine these conditions in TIL as a sample theory.[2] The terms of TIL are interpreted procedurally, which is to say that they denote *procedures* (roughly, Church's functions-in-intension) producing set-theoretical functions/mappings (Church's functions-in-extension) rather than the mappings themselves. This is in good harmony with the original interpretation of the terms of the lambda calculus, which was indeed procedural. For instance, Barendregt says:

> [I]n this interpretation the notion of a function is taken to be intensional, i.e., as an algorithm. (Barendregt 1997, 184)

We would rather say, "… is taken to be *hyperintensional*, i.e., as a procedure", because the term 'intensional' is currently reserved for mappings from possible worlds (if not among proof-theoretic semanticists, then at least among model-theoretic semanticists).

Thus λ-Closure, $[\lambda x_1 \ldots x_n\, X]$, transforms into the very procedure of producing a function by abstracting over the values of the variables $x_1, \ldots, x_n$. Similarly, Composition, $[X\, X_1 \ldots X_n]$, transforms into the very procedure of applying a function produced by the procedure X to the tuple-argument (if any) produced by the procedures $X_1, \ldots, X_n$. The procedural semantics of TIL makes it possible to explicitly deal with those features that are otherwise hidden if dealing only with the products of the procedures, i.e. functions-in-extension. These features concern in particular the operations in a hyperintensional context where the very procedure denoted by a term is being operated on, and such features show up also when dealing with β-conversion.

[2] For details on TIL see, in particular, Tichý (1988) and Duží et al. (2010).

The rule of β-reduction is a fundamental computational rule of the λ-calculi and functional programming languages. In the λ-calculi the rule is usually specified thus:

$(\lambda x\, M)\, N \vdash M\, [x := N]$

where M is a procedure with a formal parameter x, and M calls another procedure N to supply the actual argument value. Hence by '$M\, [x := N]$' is meant the collision-less substitution of N for all the occurrences of the variable x in the calling procedure M. However, Plotkin in (1975) pointed out that this specification is ambiguous. There are two procedurally or operationally non-equivalent ways of executing the rule, namely β-reduction '*by name*' and β-reduction '*by value*'. From the operational point of view, these two ways differ in the way the argument value is being passed for the formal parameter x. If by name, then the procedure denoted by the term N is executed *after* its substitution for all the occurrences of the variable x in the calling-procedure body M (after appropriate renaming of λ-bound variables to prevent collision). If by value, then the procedure N is executed *first*, and only if N does not fail to produce an argument *value* is this value substituted for all the occurrences of x in the body M. Plotkin (1975) put forward a programming language and a formal calculus for each calling mechanism and then showed how each determines the other. As a result, he proved that the two mechanisms are *not* operationally equivalent. Moreover, in Duží (2013 and 2014) it has been *logically* proved that these two ways of executing the conversion are not only operationally but also *denotationally non-equivalent* whenever *partial functions* are involved.

By *validity* of the β-conversion rule we mean the following. The rule is valid if and only if both the terms on the right-hand and the left-hand side of the rule denote procedures that are *strictly equivalent* in the sense that under any valuation v the two procedures produce the same function/mapping or are both v-improper, that is, fail to produce anything.[3]

In Duží & Jespersen (2015) it has been proved that β-reduction by name is not valid if partial functions are involved and the procedure denoted by

[3] As an extreme case, the produced function/mapping might be nullary, i.e. an atomic object. The produced object can be also a lower-order procedure.

the term *N* fails to produce an argument value.[4] However, there is an interpretation of λ-Closure, namely λ_τ-Closure, under which β-reduction by name is not valid even if the procedure *N* does produce an argument value.

The novel contributions of this paper are as follows. We define a variant of β-conversion by value and prove its validity regardless of whether particular constituents are improper and regardless of whether we deal with λ- or λ_τ-Closure. However, we also prove that in a special case of λ_τ-Closure this rule is not applicable. Moreover, this paper provides a systematic study of the applicability of β-conversion in a *hyperintensional* lambda calculus of partial functions, which to the best of our knowledge has not been presented until now, though similar work has been undertaken since the early 1970s, but merely for simple-typed or untyped λ-calculi. Moreover, the call-by-name strategy *cannot* be applied in a hyperintensional context, i.e., in hyperintensional λ-calculi such as TIL. The reason is that in such a context the formal parameter *x* is contained within a displayed (as opposed to executed) procedure that figures here only as an object to operate on, which makes the substitution *logically unfeasible.* Our *substitution method* based around the functions *Sub* and *Tr* is similar to Chang & Felleisen (2012)'s call-by-need reduction by value. However, their work is couched in an *un*typed λ-calculus.

The rest of the paper is organized as follows. Section 1 presents the fundamentals of TIL, especially the technical apparatus needed to deal with the rules of β-conversion. In Section 2 we introduce three variants of β-conversion and examine their validity; they are *β_n-conversion by name*, *β_v-conversion by value* and *restricted β_r-conversion by name*. In Section 3 we examine Tichý's λ_τ-Closure and show that there is an interpretation of his definition under which neither of the conversions by name is valid and β_v-conversion by value is not applicable. Thus, we recommend using β-reduction by value and λ-Closure only. Section 4 contains some concluding remarks.

[4] There are two other defects connected with this way of executing the rule, i.e. with calling by name, that are also demonstrated by Duží in (2013 and 2014), to wit, a loss of analytic information and non-effectiveness.

1. TIL in brief

In this section, we briefly recapitulate the technical fundamentals of TIL necessary for dealing with β-conversions. The terms of the TIL language denote abstract procedures that produce set-theoretical mappings (functions-in-extension) or lower-order procedures. These procedures are rigorously defined as TIL *constructions.* Being procedural objects, constructions can be executed in order to operate on input objects (of a lower-order type) and produce the object (if any) they are typed to produce, while non-procedural objects, i.e. non-constructions, cannot be executed. There are two atomic constructions that present input objects to be operated on. They are *Trivialization* and *Variables.* The operational sense of Trivialization is similar to that of constants in formal languages. A Trivialization presents an object *X* without the mediation of any other procedures. Using the terminology of programming languages, the Trivialization of *X*, '^0X' in symbols, is just a *pointer* to *X*. Variables produce objects dependently on valuations; they *v*-construct. We adopt an objectual variant of the Tarskian conception of variables. To each type (see Definition 2) are assigned countably many variables that range over this particular type. Objects of each type can be arranged into infinitely many sequences. The valuation *v* selects one such sequence of objects of the respective type, and the first variable *v*-constructs the first object of the sequence, the second variable *v*-constructs the second object of the sequence, and so on. Thus the execution of a Trivialization or a variable never fails to produce an object. However, the execution of some of the molecular constructions can fail to present an object of the type they are typed to produce. When this happens, we say that the constructions are *v-improper*. There are two kinds of improperness. Either a construction is compounded in a type-theoretically incoherent ('nonsensical') way, or it is an application of a function to an argument at which the function is not defined.

Thus, we define:

Definition 1 (*construction*)

(i) *Variables* *x*, *y*, … are *constructions* that construct objects (elements if their respective ranges) dependently on a valuation *v*; they *v*-construct.

(ii) Where X is an object whatsoever (even a construction), ${}^0X$ is the *construction Trivialization* that constructs X without any change.

(iii) Let X, $Y_1,\ldots,Y_n$ be arbitrary constructions. Then the *Composition* $[X\ Y_1\ldots Y_n]$ is the following *construction*. For any v, the Composition $[X\ Y_1\ldots Y_n]$ is *v-improper* if one or more of the constructions X, $Y_1,\ldots,Y_n$ are v-improper, or if X does not v-construct a function that is defined at the n-tuple of objects v-constructed by $Y_1,\ldots,Y_n$. If X does v-construct a v-proper function, then $[X\ Y_1\ldots Y_n]$ v-constructs the value of this function at the n-tuple.

(iv) (λ-) *Closure* $[\lambda x_1\ldots x_m\ Y]$ is the following *construction*. Let $x_1, x_2, \ldots, x_m$ be pair-wise distinct variables and Y a construction. Then $[\lambda x_1\ldots x_m\ Y]$ *v-constructs* the function f that takes any members $B_1, \ldots, B_m$ of the respective ranges of the variables $x_1, \ldots, x_m$ into the object (if any) that is $v(B_1/x_1,\ldots,B_m/x_m)$-constructed by Y, where $v(B_1/x_1,\ldots,B_m/x_m)$ is like v except for assigning B_1 to $x_1, \ldots, B_m$ to x_m.

(v) Where X is an object whatsoever, ${}^1X$ is the *construction Single Execution* that v-constructs what X v-constructs. Thus if X is a v-improper construction or not a construction as all, ${}^1X$ is v-improper.

(vi) Where X is an object whatsoever, ${}^2X$ is the *construction Double Execution*. If X is not itself a construction, or if X does not v-construct a construction, or if X v-constructs a v-improper construction, then ${}^2X$ is v-improper. Otherwise ${}^2X$ v-constructs what is v-constructed by the construction v-constructed by X.

(vii) Nothing is a *construction*, unless it so follows from (i) through (vi). □

Note that the (λ-) Closure $[\lambda x_1\ldots x_m\ Y]$ is not v-improper for any valuation v, as it always v-constructs a function. Even if the constituent Y is v-improper for every valuation v, the Closure is not v-improper. Yet in such a case the resulting function is a bizarre object; it is a degenerate function that is undefined at all arguments.

With constructions of constructions, constructions of functions, functions, and functional values in our stratified ontology, we need to keep track of the traffic between multiple logical strata. The *ramified type hierarchy* does just that. The type of first-order objects includes all objects that are not constructions. Therefore, it includes not only the standard objects of individuals, truth-values, sets, etc., but also functions defined on possible worlds (i.e., the intensions germane to possible-world semantics). The type of second-order objects includes constructions of first-order objects and functions that have such constructions in their domain or range. The type of third-order objects includes constructions of first- and second-order objects and functions that have such constructions in their domain or range. And so on, ad infinitum.

Definition 2 (*ramified hierarchy of types*)
Let B be a *base*, where a base is a collection of pair-wise disjoint, non-empty sets. Then:

T_1 (*types of order 1*).

(i) Every member of B is an elementary *type of order 1 over B.*

(ii) Let α, β_1, …, β_m ($m > 0$) be types of order 1 over B. Then the collection ($\alpha\ \beta_1 \dots \beta_m$) of all m-ary partial mappings from $\beta_1 \times \dots \times \beta_m$ into α is a functional *type of order 1 over B.*

(iii) Nothing is a *type of order 1 over B* unless it so follows from (i) and (ii).

C_n (*constructions of order n*)

(i) Let x be a variable ranging over a type of order n. Then x is a *construction of order n over B.*

(ii) Let X be a member of a type of order n. Then ${}^0X$, ${}^1X$, ${}^2X$ are *constructions of order n over B.*

(iii) Let $X, X_1, \dots, X_m$ ($m > 0$) be constructions of order n over B. Then $[X\ X_1 \dots X_m]$ is a *construction of order n over B.*

(iv) Let $x_1, \dots, x_m, X$ ($m > 0$) be constructions of order n over B. Then $[\lambda x_1 \dots x_m\ X]$ is a *construction of order n over B.*

(v) Nothing is a *construction of order n over B* unless it so follows from C_n (i)-(iv).

T_{n+1} (*types of order n + 1*)
Let $*_n$ be the collection of all constructions of order n over B. Then

(i) $*_n$ and every type of order n are *types of order n + 1.*

(ii) If $m > 0$ and α, β_1, …, β_m are types of order $n + 1$ over B, then ($\alpha\ \beta_1 \dots \beta_m$) (see T_1 ii)) is a *type of order n + 1 over B.*

(iii) Nothing is a *type of order n + 1 over B* unless it so follows from (i) and (ii). □

For the purposes of natural-language analysis, we are usually assuming the following base of ground types:

o: the set of truth-values {**T**, **F**};
ι: the set of individuals (the universe of discourse);
τ: the set of real numbers (doubling as discrete times);
ω: the set of logically possible worlds (the logical space).

We model sets and relations by their characteristic functions. Thus, for instance, $(o\iota)$ is the type of a set of individuals, while $(o\iota\iota)$ is the type of a relation-in-extension between individuals. Empirical expressions denote *empirical conditions* that may or may not be satisfied at the world/time pair selected as points of evaluation. We model these empirical conditions as possible-world-semantic *intensions.* Intensions are entities of type $(\beta\omega)$: mappings from possible worlds to an arbitrary type β. The type β is frequently the type of the *chronology* of α-objects, i.e., a mapping of type $(\alpha\tau)$. Thus α-intensions are frequently functions of type $((\alpha\tau)\omega)$, abbreviated as '$\alpha_{\tau\omega}$'. *Extensional entities* are entities of a type α where $\alpha \neq (\beta\omega)$ for any type β. Where w ranges over ω and t over τ, the following logical form essentially characterizes the logical syntax of empirical language:

$\lambda w \lambda t\, [\ldots w \ldots . t \ldots]$.

Examples of frequently used intensions are: *propositions* of type $o_{\tau\omega}$, *properties of individuals* of type $(o\iota)_{\tau\omega}$, *binary relations-in-intension between individuals* of type $(o\iota\iota)_{\tau\omega}$, *individual offices* (or *roles*) of type $\iota_{\tau\omega}$.

Logical objects like *truth-functions* and *quantifiers* are extensional: $\wedge$ (conjunction), $\vee$ (disjunction) and $\supset$ (implication) are of type (ooo), and $\neg$ (negation) of type (oo). The *quantifiers* $\forall^\alpha$, $\exists^\alpha$ are type-theoretically polymorphic total functions of type $(o(o\alpha))$, for an arbitrary type α, defined as follows. The *universal quantifier* $\forall^\alpha$ is a function that associates a class A of α-elements with **T** if A contains all elements of the type α, otherwise with **F**. The *existential quantifier* $\exists^\alpha$ is a function that associates a class A of α-elements with **T** if A is a non-empty class, otherwise with **F**. Below all type indications will be provided outside the formulae in order not to clutter the notation. Moreover, the outermost brackets of Closures will be omitted whenever no confusion can arise. Furthermore,

'X/α' means that an object X is (a member) of type α. '$X \to_v \alpha$' means that X is typed to v-construct an object of type α, regardless of whether X in fact constructs anything. We write '$X \to \alpha$' if what is v-constructed does not depend on a valuation v. Throughout, it holds that the variables $w \to_v \omega$ and $t \to_v \tau$. If $C \to_v \alpha_{\tau\omega}$ then the frequently used Composition $[[C\ w]\ t]$, which is the intensional descent (a.k.a. extensionalization) of the α-intension v-constructed by C, will be encoded as 'C_{wt}'.

In order to work with a hyperintensional context, in which a *construction* is operated on, we need two special functions, *Sub* and *Tr*. The polymorphic function *Sub* of type $(*_n*_n*_n*_n)$ operates on constructions as follows. When applied to constructions C_1, C_2, C_3, *Sub* returns as its value the construction D that is the result of the correct (i.e. collision-less) substitution of C_1 for C_2 in C_3. For instance, the result of the Composition $[{}^0Sub\ {}^{00}John\ {}^0him\ {}^0[{}^0Wife\_of_{wt}\ him]]$ is the Composition $[{}^0Wife\_of_{wt}\ {}^0John]$. The logical operation of substitution is treated as a theoretical primitive.

The likewise polymorphic function *Tr* returns as its value the Trivialization of its argument. Thus the result of $[{}^0Tr\ {}^0John]$ is ${}^0John$. If what is wanted as output is the Trivialization of the Trivialization of John, the corresponding Composition is $[{}^0Tr\ {}^{00}John]$. When x ranges over ι, the Composition $[{}^0Tr\ x]$ $v(John/x)$-constructs ${}^0John$. Note one essential difference between the function *Tr* and the construction Trivialization. Whereas the variable x is *free* in $[{}^0Tr\ x]$, the Trivialization ${}^0x$ *binds* the variable x by constructing just x independently of valuation.

2. β-conversion

In the lambda calculi, the rule of β-conversion is usually specified in this form:

$$(\lambda x.M)\ N =_\beta M\ [x{:=}N]$$

The right-hand side contractum is the result of substituting the *term N* for all free occurrences of the variable x within the term M. The rule of β-reduction is the left-to-right part of the above equality:

$$(\beta) \qquad (\lambda x.M)\ N \to_\beta M\ [\mathrm{x}{:=}N]$$

The rule (β) is used to model the application of the function referred to by the term λ*x*.*M* to the argument denoted by *N*. Using programming-language technical jargon, we can explicate the rule as follows. The term λ*x*.*M* denotes, or *declares*, a procedure with a formal parameter *x* and the procedural body *M*. Thus the redex on the left-hand side denotes a procedure that consists in calling the procedure λ*x*.*M* which is to be executed with the actual argument value replacing the formal parameter *x*, and this value is to be provided by the sub-procedure *N*. The contractum term on the right-hand side is schematic. In principle, it can be read as the instruction to execute the procedural body *M* in which the formal parameter *x* has been replaced by the actual argument value provided by the procedure *N*. In case *N* fails to produce an argument value, the procedure body *M* has nothing to operate on, and thus the rule (β) cannot be applied. While in the λ-calculi of total functions this fact is irrelevant, in the λ-calculi of partial functions this eventuality has to be taken into account.[5]

Partiality, as we only know too well, brings about technical complications. However, we do need to work with partial functions, because otherwise we face the problem of a non-recursive explosion of domains that is computationally non-tractable (for details see Duží 2003). Yet just a few results have been obtained in this area. Moggi (1988) would appear to have been the first to put forward a definition of a partial λ-calculus, and Feferman (1995) presents a set of axioms for the *Partial Lambda Calculus* (for details see, e.g., Duží et al. 2010, § 2.7, 261-262). However, they both specify the predicate '↓' which in '*N*↓' means that the term *N* is 'defined' or 'referring'. Consequently, the rule is valid in the sense of weak congruency; if both sides are defined then they denote the same value. However, such a restriction to non-recursively defined cases of *v*-properness would be a serious shortcoming of TIL or indeed any other formal semantics based on the λ-calculus. Hence, we do need a *universally valid* rule regulating β-transformation. TIL is a λ-calculus of partial functions, and in virtue of its procedural semantics we have the technical machinery required to specify a universally valid rule of β-conversion.

[5] A partial function is a function with *at most* one value at each argument. Every total function is, therefore, a partial function, but not vice versa.

2.1. *Three kinds of β-conversion*

Now we are going to examine three kinds of β-conversion using the technical apparatus of TIL. The three kinds are β-*conversion by name* (β_n-conversion), β-*conversion by value* (β_v-conversion), and *restricted* β-*conversion by name* (β_r-conversion). We will use simple examples to illustrate them. Let the calling procedure *M* and the called procedure *N* be [λ*x* λ*y* [^{0}> *y x*]] and [^{0}*Div* ^{0}3 ^{0}0], respectively. Then we have the Composition

(1) [[λ*x* λ*y* [^{0}> *y x*]] [^{0}*Div* ^{0}3 ^{0}0]]

Types. *x*, *y* $\rightarrow_v$ τ; >/(οττ); *Div*/(τττ): the division function; 3,0/τ.

The Closure [λ*x* λ*y* [^{0}> *y x*]] produces a mapping of type ((οτ)τ), i.e. a function *f* that associates a number *x* with the class of numbers *y* that are larger than the number *x*.

However, as mentioned above, partiality is a complicating factor. Some molecular constructions can be *v-improper* in the sense of failing to produce the sort of object they are typed to construct. There are two kinds of improperness, as we said above. Either a construction is compounded in a type-theoretically incoherent way, or it is the procedure of applying a function to an argument at which the function is not defined. We will now address the latter kind of improperness. Improperness rooted in wrong typing will be examined in Section 3 below.

The Composition [^{0}*Div* ^{0}3 ^{0}0] is the procedure of applying the division function to arguments 3 and 0. Since dividing any number by 0 is not defined, this Composition does not *v*-construct anything for any valuation *v*; it is *v-improper* for any valuation *v*, or *improper* for short.

The Composition (1) is the procedure of applying the function *f* constructed by [λ*x* λ*y* [^{0}> *y x*]] to the argument that is to be produced by the Composition [^{0}*Div* ^{0}3 ^{0}0]. Yet since this Composition does not produce anything, there is no argument to apply *f* to. Hence, the Composition (1) is by Def. 1 also *improper*.

2.1.1. *β-conversion by name*

The result of applying β_n-conversion to (1) is that the *x* in the 'body' of *M* is replaced by [^{0}*Div* ^{0}3 ^{0}0]. This yields:

(2) $[[\lambda x\, \lambda y\, [^0{>}\, y\, x]]\, [^0Div\, ^03\, ^00]] \rightarrow_{\beta n} [\lambda y\, [^0{>}\, y\, [^0Div\, ^03\, ^00]]]$

This result demonstrates the problem of β_n-reduction in the logic of partial functions. While the left-hand side Composition of (2) is improper, the right-hand side contractum is *not improper.* It produces a *degenerate function* undefined at all its arguments. In other words, we obtain an empty class of numbers, the characteristic function of which is undefined at any number. Bizarre as it is, it is still something rather than nothing and therefore an object. Hence the left-hand and the right-hand side constructions of (2) are not strictly equivalent, hence the β_n-rule is not valid.

In this simple case, the absence of strict equivalence might seem harmless. After all, if that bizarre function is applied to a number, the result is an improper construction; hence also a gap comparable to a truth-value gap, and the final result would be the same. Yet our operational semantics reveals that it is not quite as harmless as it might seem. The execution of the left-hand side construction is improper, which is something we already know. It makes no sense to execute this construction, because it fails to produce something. However, in the right-hand side construction this fact is hidden. We end up with a procedure producing a function, and only after calling this procedure a second time is this failure revealed.

This deficiency is best demonstrated by an analysis of an empirical attitude *de re*. Consider:

(3) Tom believes *of* the Pope that he is wise.

On the *de re* reading of (3) the property of being believed by Tom to be wise is ascribed to the individual (if any) that holds the papal office. Thus, the analysis amounts to this construction:

(3*) $\lambda w \lambda t\, [\lambda he\, [^0Believe_{wt}\, ^0Tom\, \lambda w^* \lambda t^*\, [^0Wise_{w^*t^*}\, he]]\, ^0Pope_{wt}]$

Types. Variable $he \rightarrow_v \iota$; $Believe/(o\iota o_{\tau\omega})_{\tau\omega}$; Tom/ι; $Wise/(o\iota)_{\tau\omega}$; $Pope/\iota_{\tau\omega}$.[6]

[6] For the sake of simplicity, we analyse the attitude of believing intensionally, that is, as a relation-in-intension to a *proposition*, which makes for an *implicit* attitude. The believer is related to the proposition regardless of the particular way the proposition is conceptualized or constructed. This approach yields notorious problems with logical-mathematical omniscience. Thus, a more appropriate analysis would be hy-

The construction of the papal office, ^{0}*Pope*, occurs in (3*) extensionally, i.e. with *de re* supposition. Thus if the Pope does not exist (that is, if the papal office is not occupied in world w and time t of evaluation) then its extensionalization ${}^0Pope_{wt}$ is v-improper, and (3*) constructs a proposition that lacks a truth-value at the relevant $\langle w, t\rangle$-pair. This is as it should be, though, because there is an *existential presupposition de re.*[7] Now executing β-reduction by name consists in replacing the 'formal parameter' *he* (that is, the variable *he*) by the Composition ${}^0Pope_{wt}$, which in turn yields this construction:

(4) $\lambda w\lambda t\ [{}^0Believe_{wt}\ {}^0Tom\ \lambda w^*\lambda t^*\ [{}^0Wise_{w^*t^*}\ {}^0Pope_{wt}]]$

However, in (4) ^{0}*Pope* does *not* occur with supposition *de re*. This is because ${}^0Pope_{wt}$ has been drawn into the λ-generic intensional context of Tom's perspective ($\lambda w^*\lambda t^*$), and an intensional context is dominant over the lower extensional one, which in turn means that the improperness of ${}^0Pope_{wt}$ is suppressed or irrelevant. If ${}^0Pope_{wt}$ is v-improper, then Tom believes that the degenerate proposition v-constructed by the Closure $\lambda w^*\lambda t^*\ [{}^0Wise_{w^*t^*}\ {}^0Pope_{wt}]$ is true, which is a logical possibility. In other words, there is no logical reason for the proposition constructed by (4) to be undefined. Thus β_n-reduction has turned a *de re* occurrence into a *de dicto* occurrence, which is wrong.

For these reasons a necessary condition for the validity of β-reduction by name is usually specified, namely that the procedure that is typed to produce an argument value be proper. For instance, Raclavský (2009) presents the following definition of the validity of β-reduction by name:

> Let C be a closure of the form $\lambda x\ [\ldots x\ldots]$ that can contain also other variable than x (λ-bound or not). Let C be composed with the construction D in the Composition $[C\ D]$. Let D be a v-proper construction. If D contains free occurrences of variables and these variables are λ-bound

perintensional believing relating the believer to a hyperproposition, that is, a *construction* of a proposition, which makes for an *explicit* attitude: $Believe^*/(o\iota*_n)_{\tau\omega}$. Yet as a toy example, demonstrating the invalidity of β-reduction by name the implicit *Believe* suffices.

[7] For details on the analysis of propositional attitudes *de dicto* and *de re* see, for instance, Duží et al. (2010, § 5.1) or Duží & Jespersen (2012).

in *C*, then let *C* be α-expanded into a construction *C′* that does not contain the variables free in *D* as λ-bound. Then the construction *C″* that is obtained from *C′* by substituting the construction *D* for all free occurrences of the λ-bound variable corresponding to *x* in *C″* is the *β-reduced form of the construction C*. (Raclavský 2009, 285)[8]

Hence, the conditions for the validity of β-reduction by name can be summarized as follows:

i) the construction *D* of an argument value must be *v*-proper;
ii) no collision of variables must arise; if *D* contains free occurrences of variables that occur λ-bound in *C*, we must apply α-conversion to avoid collision.

This is a standard way of specifying β-conversion by name. However, in Duží & Jespersen (2013) another shortcoming of β-reduction by name has been identified. Even if β-reduction by name is a valid transformation satisfying conditions (i) and (ii), it can yield *a loss of analytic information* about *which function* has been applied to *which argument*. The authors illustrate this problem by an analysis of the well-known sentence, "John loves his wife, and so does Peter". There are two non-equivalent readings of this sentence. On the so-called sloppy reading, both John and Peter love their own wives, making them exemplary husbands. On the so-called strict reading, John and Peter share the property of loving John's wife, with trouble looming on the horizon. The problem is that β-reduction by name reduces the sloppy reading to the strict one, squeezing out the former. As a result, the anaphor resolution of 'so does Peter' invalidates the natural reading on which Peter loves his own wife whom he is presupposed not to share with John, as would be a possibility in a bigamist culture. As a solution to this problem, the authors define the rule of β-reduction by value, which we are going to examine below.

2.1.2. Restricted β-conversion by name

Above we specified the shortcomings evinced by β-conversion by name. There is, however, a restricted variant of this conversion that

8 Translated from the Czech original by the authors.

suffers none of them. This variant is *restricted* β-conversion by name. β_r-conversion consists in collision-less substitution of free variables for λ-bound variables ranging over the same types. It is a strictly equivalent, and thus valid, conversion. For instance, $[\lambda x\ [{}^0{+}\ x\ {}^0 1]\ y]$ can be simplified to $[{}^0{+}\ y\ {}^0 1]$. This transformation is nothing but a manipulation with λ-bound variables that has much in common with η-reduction and much less with β-reduction. The latter is the operation of applying a function *f* to its argument *a* in order to obtain the value of *f* at *a* (leaving it open whether a value emerges). No such features can be found in β_r-reduction. It is just a formal simplification of the original construction.

For instance, above we analysed the *de re* attitudinal sentence “Tom believes of the Pope that he is wise” as ascribing the property of being believed by Tom to be wise to the holder of the papal office:

$$\lambda w \lambda t\ [\lambda he\ [{}^0Believe_{wt}\ {}^0Tom\ \lambda w^*\lambda t^*\ [{}^0Wise_{w^*t^*}\ he]]\ {}^0Pope_{wt}]$$

This is the β_r-restricted form of the literal analysis of the sentence “The Pope has the *property* of being believed by Tom to be wise”, which amounts to

$$\lambda w \lambda t\ [\lambda w'\lambda t'[\lambda he\ [{}^0Believe_{w't'}\ {}^0Tom\ \lambda w^*\lambda t^*\ [{}^0Wise_{w^*t^*}\ he]]]_{wt}\ {}^0Pope_{wt}]$$

Yet we see little reason to differentiate semantically or logically between “The Pope is believed by Tom to be wise” and “The Pope has the property of being believed by Tom to be wise”.[9] Hence, this kind of reduction is frequently applied in logical analysis of natural-language expressions.

[9] This is not to say we see no reason at all not to differentiate. For instance, if the believer is a self-assured nominalist then they may protest that while they do believe that the Pope is wise they do not believe that the Pope has any properties. Or it could be argued that one thing is to believe that the Pope is wise and another is to believe that the Pope has the property of being wise, because the latter at least appears to presuppose that the believer have the additional conceptual resources to master the notion of *property*.

2.1.3. β-conversion by value

Above we examined unrestricted β-conversion by name and warned against its undesirable side-effects. The difference between conversion by name and by value has consequences also from the point of view of *computational complexity*. When conversion by name is executed, the called procedure *N* is to be executed as many times as the variable *x* occurs in the calling procedure *M*. Here is a simple example for illustration. Consider the application of the identity function $\lambda x\ [x=x]$ to the argument computed by[10] $((1+1)/2)^2$:

$[\lambda x\ [x=x]\ ((1+1)/2)^2]$

Reduction by name results in the equality

(A) $(1+1)/2)^2 = (1+1)/2)^2$

On the other hand, if we pass the argument by value, then we first obtain the argument value by executing the procedure $(1+1)/2)^2$. This produces the number 1, the Trivialization of which is afterwards substituted for *x*. As a result, we obtain the equality

(B) ${}^0 1 = {}^0 1$

It is readily seen that procedure (A) is much more complicated than (B). Passing the argument to the function by name and by value makes, therefore, a difference to the computational complexity of the resulting procedure.

Hence, we need a universal rule of β-conversion that would not exhibit the above defects. Fortunately, it turns out to be feasible to formulate such a *generally valid logical rule*. The invalid rule by name is moulded on the programming technique of calling a sub-procedure *N* by name: the sub-procedure itself is substituted for the 'local variable' *x* in the 'procedure body' *M*. Programmers are well aware of the fact that this technique can

[10] Now we use the usual mathematical notation to make the constructions easier to read. In TIL notation the construction $((1+1)/2)^2$ would be written as '$[{}^0 Power\ [{}^0 Div\ [{}^0 +\ {}^0 1\ {}^0 1]\ {}^0 2]\ {}^0 2]$', where *Power*, *Div*/(υυυ), υ the type of natural numbers.

have undesirable side-effects, unlike the technique of calling a sub-procedure by value.

The rule of β-reduction *by value* was originally specified logically for TIL in Duží et al. (2010, § 2.7). Unfortunately, there is a typo in Claim 2.6, (cf. Duží et al. 2010, 270) that proves its validity. The correct definition can be found in Duží (2014). Here we recapitulate the correct definition and provide the proof of validity.

Definition 3 *(β-conversion by value)*
Let $Y \rightarrow_v \alpha$; $x_1, D_1 \rightarrow_v \beta_1, \ldots, x_n, D_n \rightarrow_v \beta_n$, $[\lambda x_1 \ldots x_n\ Y] \rightarrow_v (\alpha\beta_1 \ldots \beta_n)$. Then the conversion

$$[[\lambda x_1 \ldots x_n\ Y]\ D_1 \ldots D_n] \Rightarrow_\beta {}^2[{}^0Sub\ [{}^0Tr\ D_1]\ {}^0x_1 \ldots [{}^0Sub\ [{}^0Tr\ D_n]\ {}^0x_n\ {}^0Y]]$$

is *β-reduction by value*. The reverse conversion is *β-expansion by value*. □

Claim 1
β-reduction and β-expansion by value are valid conversions. In other words, the constructions

$$[[\lambda x_1 \ldots x_n\ Y]\ D_1 \ldots D_n]$$

and

$${}^2[{}^0Sub\ [{}^0Tr\ D_1]\ {}^0x_1 \ldots [{}^0Sub\ [{}^0Tr\ D_n]\ {}^0x_n\ {}^0Y]]$$

are strictly equivalent.

Proof
Let C be identical to $[[\lambda x_1 \ldots x_n\ Y]\ D_1 \ldots D_n]$ and D to ${}^2[{}^0Sub\ [{}^0Tr\ D_1]$ ${}^0x_1 \ldots [{}^0Sub\ [{}^0Tr\ D_n]\ {}^0x_n\ {}^0Y]]$. We are to prove that for any valuation v either both C and D are v-improper, or C and D v-construct the same object.

(a) If for some i, $1 \leq i \leq n$, construction D_i is v-improper then so is the Composition C, according to Def. 1, iii). Then also the Compositions $[{}^0Tr\ D_i]$ and $[{}^0Sub\ [{}^0Tr\ D_1]\ {}^0x_1 \ldots [{}^0Sub\ [{}^0Tr\ D_n]\ {}^0x_n\ {}^0Y]]$ are v-improper according to Def. 1, iii), and thus also the construction D is v-improper according to Def. 1, vi).

(b) Otherwise, let $D_1, \ldots, D_n$ all be v-proper, v-constructing the objects $d_1, \ldots, d_n$, respectively. Then by Def. 1, iv) the Closure $[\lambda x_1 \ldots x_n\ Y]$ v-constructs the function $f/(\alpha\beta_1 \ldots \beta_n)$.

(b1) If Y is $v(d_1/x_1,\ldots,d_n/x_n)$-improper, then f is undefined on $\langle d_1,\ldots,d_n\rangle$ and thus Composition C is v-improper according to Def. 1, iii). We are to show that D is also v-improper. The Composition $[{}^0Sub\ [{}^0Tr\ D_1]\ {}^0x_1 \ldots [{}^0Sub\ [{}^0Tr\ D_n]\ {}^0x_n\ {}^0Y]]$ v-constructs $Y(x_1/{}^0d_1,\ldots, x_n/{}^0d_n)$, i.e. the construction Y where all the occurrences of the variables $x_1,\ldots,x_n$ have been replaced by ${}^0d_1,\ldots,{}^0d_n$, respectively. Since Y is $v(d_1/x_1,\ldots,d_n/x_n)$-improper, the execution of $Y(x_1/{}^0d_1,\ldots, x_n/{}^0d_n)$, hence D, is v-improper as well according to Def. 1, vi).

(b2) Otherwise, if Y is not $v(d_1/x_1,\ldots,d_n/x_n)$-improper, then the value of f on $\langle d_1,\ldots,d_n\rangle$ is the α-entity $v(d_1/x_1,\ldots,d_n/x_n)$-constructed by Y. Let this α-entity be a. Then by Def. 1, iii), construction C v-constructs a. We are to show that construction D also v-constructs a. The first Execution of D v-constructs $Y(x_1/{}^0d_1,\ldots, x_n/{}^0d_n)$. Since the Trivializations ${}^0d_1,\ldots,{}^0d_n$ construct the entities $d_1,\ldots,d_n$, respectively, the second Execution v-constructs the entity a.

Hence, C and D come out strictly equivalent.

In Section 2.1.1 we demonstrated the invalidity of the β_n-conversion of the Composition

$$[\lambda x\ [\lambda y\ [{}^0\!>y\ x]]\ [{}^0Div\ {}^03\ {}^00]]$$

Using the rule of β_v-conversion defined above, here is a valid conversion of this Composition:

$$[\lambda x\ [\lambda y\ [{}^0\!>y\ x]]\ [{}^0Div\ {}^03\ {}^00]] \Rightarrow_\beta {}^2[{}^0Sub\ [{}^0Tr\ [{}^0Div\ {}^03\ {}^00]]\ {}^0x\ {}^0[\lambda y\ [{}^0\!>y\ x]]]$$

It is readily seen that both the left-hand and the right-hand side constructions are improper. Indeed, since $[{}^0Div\ {}^03\ {}^00]$ is improper, by Def. 1, iii) the Composition $[\lambda x\ [\lambda y\ [{}^0\!> y\ x]]\ [{}^0Div\ {}^03\ {}^00]]$ is improper. For the same reason, the Composition $[{}^0Tr\ [{}^0Div\ {}^03\ {}^00]]$ is improper and thus also the whole Composition $[{}^0Sub\ [{}^0Tr\ [{}^0Div\ {}^03\ {}^00]]\ {}^0x\ {}^0[\lambda y\ [{}^0\!> y\ x]]]$ as well as its Double Execution are improper. Partiality is strictly propagated up, as it should be.

Similarly, the analysis of the *de re* attitude (3*) can be validly reduced in this way:

$\lambda w \lambda t\ [\lambda he\ [{}^0Believe_{wt}\ {}^0Tom\ \lambda w^*\lambda t^*\ [{}^0Wise_{w^*t^*}\ he]]\ {}^0Pope_{wt}] \Rightarrow_\beta$
$\lambda w \lambda t\ {}^2[{}^0Sub\ [{}^0Tr\ {}^0Pope_{wt}]\ {}^0he\ {}^0[{}^0Believe_{wt}\ {}^0Tom\ \lambda w^*\lambda t^*\ [{}^0Wise_{w^*t^*}\ he]]]$

Remark: The reduced construction is actually the *literal* analysis of the sentence "Tom believes of the Pope that he is wise". The anaphoric reference 'he' referring to the holder of the papal office is resolved by the substitution of this holder (if any) for the variable *he*, that is, by the constituents $[{}^0Sub\ [{}^0Tr\ {}^0Pope_{wt}]\ {}^0he\ \ldots$.

Proof

We are to prove that for any world *w* and time *t* of evaluation, the constructions

$[\lambda he\ [{}^0Believe_{wt}\ {}^0Tom\ \lambda w^*\lambda t^*\ [{}^0Wise_{w^*t^*}\ he]]\ {}^0Pope_{wt}]$

and

${}^2[{}^0Sub\ [{}^0Tr\ {}^0Pope_{wt}]\ {}^0he\ {}^0[{}^0Believe_{wt}\ {}^0Tom\ \lambda w^*\lambda t^*\ [{}^0Wise_{w^*t^*}\ he]]]$

v-construct the same truth-value or are both *v*-improper.

1) Let ${}^0Pope_{wt}$ *v*-construct an individual *a*. Then we will show that both constructions *v*-construct the same truth-value as does the Composition $[{}^0Believe_{wt}\ {}^0Tom\ \lambda w^*\lambda t^*\ [{}^0Wise_{w^*t^*}\ {}^0a]]$. In any world *w* and time *t* of evaluation the following steps are truth-preserving:

$\Rightarrow$

a) $[\lambda he\ [{}^0Believe_{wt}\ {}^0Tom\ \lambda w^*\lambda t^*\ [{}^0Wise_{w^*t^*}\ he]]\ {}^0Pope_{wt}]$ $\varnothing$
b) ${}^0Pope_{wt} = {}^0a$ $\varnothing$
c) $[\lambda he\ [{}^0Believe_{wt}\ {}^0Tom\ \lambda w^*\lambda t^*\ [{}^0Wise_{w^*t^*}\ he]]\ {}^0a]$
a), b), SI (Leibniz)
d) $[{}^0Proper\ {}^0a]$ by Def. 1
e) $[{}^0Believe_{wt}\ {}^0Tom\ \lambda w^*\lambda t^*\ [{}^0Wise_{w^*t^*}\ {}^0a]]$ β-reduction by name

$\Leftarrow$

f) ${}^2[{}^0Sub\ [{}^0Tr\ {}^0Pope_{wt}]\ {}^0he\ {}^0[{}^0Believe_{wt}\ {}^0Tom\ \lambda w^*\lambda t^*\ [{}^0Wise_{w^*t^*}\ he]]]$ $\varnothing$
g) ${}^0Pope_{wt} = {}^0a$ $\varnothing$
h) $[{}^0Tr\ {}^0Pope_{wt}] = [{}^0Tr\ {}^0a]$ g), SI, Def. of *Tr*
i) ${}^2[{}^0Sub\ [{}^0Tr\ {}^0a]\ {}^0he\ {}^0[{}^0Believe_{wt}\ {}^0Tom\ \lambda w^*\lambda t^*\ [{}^0Wise_{w^*t^*}\ he]]]$
h), SI
j) $[{}^0Believe_{wt}\ {}^0Tom\ \lambda w^*\lambda t^*\ [{}^0Wise_{w^*t^*}\ {}^0a]]$
i), Def. of *Sub*, Def.1, vi)

2) Let ${}^0Pope_{wt}$ be v-improper. Then by Def. 1, iii), vi) all the Compositions

$[{}^0Tr\ {}^0Pope_{wt}]$,
$[\lambda he\ [{}^0Believe_{wt}\ {}^0Tom\ \lambda w^*\lambda t^*\ [{}^0Wise_{w^*t^*}\ he]]\ {}^0Pope_{wt}]$
$[{}^0Sub\ [{}^0Tr\ {}^0Pope_{wt}]\ {}^0he\ {}^0[{}^0Believe_{wt}\ {}^0Tom\ \lambda w^*\lambda t^*\ [{}^0Wise_{w^*t^*}\ he]]]$

and thus also the Double Execution

${}^2[{}^0Sub\ [{}^0Tr\ {}^0Pope_{wt}]\ {}^0he\ {}^0[{}^0Believe_{wt}\ {}^0Tom\ \lambda w^*\lambda t^*\ [{}^0Wise_{w^*t^*}\ he]]]$

are v-improper. □

Remarks: At steps (c), (h) and (i) the rule of substitution of identicals for extensional contexts is applied. More precisely, since ${}^0Pope$ occurs extensionally (*de re*), the v-congruent constructions ${}^0Pope_{wt}$ and ${}^0a$ are substitutable salva veritate here. For details, see Duží et al. (2010, §2.7.1) and Duží (2013). Step (e) is justified by step (d). Since the Trivialization of an entity is never v-improper, β-reduction by name can be validly applied here.

3. λ_α-Closure and β-conversion

We have so far tacitly applied the definition of λ-Closure as per Definition 1, iv):

(λ-)*Closure* $[\lambda x_1 \ldots x_m\ Y]$ is the following *construction*. Let $x_1, \ldots, x_m$ be pair-wise distinct variables and Y a construction. Then $[\lambda x_1 \ldots x_m\ Y]$ *v-constructs* a function f that takes any members $B_1, \ldots, B_m$ of the respective ranges of the variables $x_1, \ldots, x_m$ into the object (if any) that is $v(B_1/x_1, \ldots, B_m/x_m)$-constructed by Y, where $v(B_1/x_1, \ldots, B_m/x_m)$ is like v except for assigning B_1 to x_1, …, B_1 to x_m.

According to this definition, λ-Closure is not v-improper for any valuation v; it always v-constructs a function f of the following type. Let $x_1 \rightarrow \beta_1, \ldots, x_m \rightarrow \beta_m$, and let Y be typed to v-construct objects of type α. Then the function f is of type $(\alpha\ \beta_1 \ldots \beta_m)$. However, the function f can be a *degenerate function*, which takes no argument to a value. This is the case when Y is v-improper for any valuation v.

Tichý (1988) applies the definition of λ_α-Closure that leaves room for a slightly different procedure than the above. Recapitulating Tichý's definition, we have:[11]

> To generalize, let α be a type, $x_1, \ldots, x_m$ distinct variables ranging over the respective types $\beta_1, \ldots, \beta_m$, and v a valuation. *Any construction Y* can be used in constructing a mapping from $\beta_1, \ldots, \beta_m$ into α; we shall call this latter construction the *λ_α-closure of Y on $x_1, \ldots, x_m$*, or briefly $[\lambda_\alpha x_1 \ldots x_m Y]$. For any v, $[\lambda_\alpha x_1 \ldots x_m Y]$ v-constructs the mapping which takes any $X_1, \ldots, X_m$ of the respective types $\beta_1, \ldots, \beta_m$ into that member (if any) of α which is $v(X_1/\mathrm{x}_1, \ldots, X_m/x_m)$-constructed by Y, where $v(X_1/\mathrm{x}_1, \ldots, X_m/x_m)$ is like v except for assigning X_1 to x_1, …, and X_m to x_m. (Tichý 1988, 65; emphasis ours)

Claim 2
Every λ-Closure is a λ_α-Closure, but not vice versa.

Proof
Let $Y \rightarrow_v \gamma$ and $\alpha = \gamma$. Then λ-Closure and λ_α-Closure are identical procedures producing the same (possibly degenerate) function *f*. However, if $\alpha \neq \gamma$ then λ-Closure is a procedure identical to λ_γ-Closure that produces a function $f/(\gamma\ \beta_1 \ldots \beta_m)$ while λ_α-Closure is another procedure producing a (degenerate) function $g/(\alpha\ \beta_1 \ldots \beta_m)$. □

Claim 3
The λ_α-Closure $[\lambda_\alpha x_1 \ldots x_m Y]$, $Y \rightarrow_v \gamma$ and $\alpha \neq \gamma$, v-constructs a degenerate function.

Proof is obvious. If Y is typed to v-construct objects of type γ then the resulting mapping does not take any $X_1, \ldots, X_m$ of the respective types $\beta_1, \ldots, \beta_m$ into that member (if any) of type α which is $v(X_1/\mathrm{x}_1, \ldots, X_m/x_m)$-constructed by Y, because there is no such member.

Hence, if $\alpha \neq \gamma$ then λ_α-Closure is not identical to any λ-Closure. According to Tichý's definition, λ_α-Closure is also never v-improper for any

[11] Tichý uses here the term 'collection', because his definition of types follows only later in the text. For the sake of simplicity and in the interest of a smooth reading, we use 'type' instead of 'collection'.

valuation v. It always v-constructs a function f; but again, this function can be degenerate. As with λ-Closure, the function f is degenerate in case Y is v-improper for any valuation v. But, importantly, this definition leaves room for another way of v-constructing a degenerate function. Suppose that Y is typed to v-construct objects of type γ, where $\gamma \neq \alpha$. Then, since for any valuation v no member of the type α is v-constructed by Y, the function $f/(\alpha\ \beta_1 \ldots \beta_m)$ is degenerate even if Y itself is not v-improper.[12]

A simple example to illustrate the situation. Let C be the λ_τ-*Closure*

$[\lambda_\tau x\ [{}^0{=}\ {}^00\ {}^01]]$

where τ is the type of real numbers, $x \rightarrow_v \tau$. Then the function f produced by $[\lambda_\tau x\ [{}^0{=}\ {}^00\ {}^01]]$ is a degenerate function. The reason is this. According to the strict reading of Tichý's definition the λ_τ-Closure C produces a mapping of type $(\tau\tau)$. Yet the Composition $[{}^0{=}\ {}^00\ {}^01]$ produces the truth-value **F**, which is an object of type ο. Hence no object of type τ is v-constructed by $[{}^0{=}\ {}^00\ {}^01]$, and thus f does not return any value at any number. Note the difference between $[\lambda_\tau x\ [{}^0{=}\ {}^00\ {}^01]]$ and $[\lambda x\ [{}^0{=}\ {}^00\ {}^01]]$. While the former constructs a degenerate function f of type $(\tau\tau)$, the latter constructs an empty class of numbers, that is, an object of type $(o\tau)$.

The difference between λ-Closure and λ_α-Closure affects also the validity of β-conversion. When dealing with the validity of rules of β-conversion, we have considered so far only one problematic issue, namely the case when the procedure N that is to produce an argument value is v-improper by failing to do so. We have shown that in such a case the unrestricted rule of β-reduction by name is not a valid rule, while the restricted version of β-reduction by name and β-reduction by value are valid rules.

Nonetheless, there is another problematic issue, namely the procedure of applying a degenerate function f to an argument value. Trivially, such a procedure fails to produce anything, because a degenerate function returns no value at any argument. We have seen that the λ-Closure $[\lambda\ x_1 \ldots x_m\ Y]$ v-constructs a degenerate function f in case Y is v-improper for any valuation

[12] True, it is dubious whether Tichý indeed intended the interpretation that we present here, because he does not adduce any example of such a procedure. Thus it seems that he tacitly presupposed that the type α is identical with the type γ, and that the subscript α at 'λ_α' was intended only to indicate objects of which type Y is typed to v-construct. Yet if we take his definition literally, such an interpretation is possible.

v. Yet even in this case the rule of β-conversion by value is valid, as we have proved in *Claim* 1.

However, the λ_α-Closure $[\lambda_\alpha\, x_1 \ldots x_m\, Y]$ such that $Y \rightarrow_v \gamma$ and $\alpha \neq \gamma$ v-constructs a degenerate function f even in case Y is not v-improper, which is a complicating factor.

To illustrate the situation, consider the Composition

(5) $[[\lambda_\tau x\, [{}^0{=}\ {}^0 0\ x]]\ {}^0 2]$

It satisfies both of the conditions (i) and (ii) for the validity of β-reduction by name as specified above; (i) Trivialization ${}^0 2$ is never v-improper, it constructs an argument value, namely the number 2 to which the function $f/(\tau\tau)$ constructed by $[\lambda_\tau\, x\, [{}^0{=}\ {}^0 0\ x]]$ is applied. The second condition (ii) is trivially satisfied, there being no collision of variables. Hence β-reduction by name would appear to be valid; but alas, it *is not*:

$$[[\lambda_\tau x\, [{}^0{=}\ {}^0 0\ x]]\ {}^0 2] \rightarrow_\beta [{}^0{=}\ {}^0 0\ {}^0 2]$$

The reason is obvious. Since f returns no value at any argument, its application to any number is improper. Thus the left-hand side redex, i.e. the Composition (5), is improper. However, the contracted right-hand side Composition $[{}^0{=}\ {}^0 0\ {}^0 2]$ is a proper Composition producing the truth-value **F**.

The main reason for the insufficiency of Raclavský's proposal of the validity conditions for β-reduction (by name) is that he does not take into account the strict literal reading of Tichý's definition of λ_τ-Closure $[\lambda_\tau\, x_1 \ldots x_m\, Y]$, that is, the possibility that Y is typed to v-construct objects of type α where $\alpha \neq \tau$. In other words, he works with λ-Closure rather than λ_τ-Closure.

Actually, to the best of our knowledge, the possibility that λ_α-Closure $[\lambda_\alpha\, x_1 \ldots x_m\, Y]$ can v-construct a degenerate function though the 'body' procedure Y is v-proper has not been taken into account up to now. Thus, we formu-late:

Claim 4

Let a construction $Y \rightarrow_v \gamma$ be v-proper for any valuation v, and let $\alpha \neq \gamma$. Further, let $D_1,\ldots,D_m$ be v-proper constructions of objects of the respective types $\beta_1,\ldots,\beta_m$. Then the λ_α-Closure $[\lambda_\alpha\, x_1 \ldots x_m\, Y]$ v-constructs a degenerate function $f/(\alpha\ \beta_1\ldots\beta_m)$ due to $Y \rightarrow_v \gamma$ and $\alpha \neq \gamma$, and β-reduction by name, symbolized thus:

$$[[\lambda_\alpha x_1 \ldots x_m\ Y]\ D_1,\ldots,D_m] \rightarrow_\beta Y(x_1{:}D_1,\ldots,x_m{:}D_m)$$

is not a valid conversion.

Proof
By assumption, the left-hand side is the procedure of applying a degenerate function to a tuple-argument provided by $D_1,\ldots,D_m$. Since a degenerate function does not have any value at any argument, this procedure is v-improper by failing to produce any value. Yet the right-hand side procedure is proper by assumption. □

Fortunately, β-conversion by value is unaffected by this kind of invalidity. According to Def. 3, β-conversion by value is applicable only if type α is identical to type γ. If they are not, β-conversion by value would not be valid, either. Recall the Composition (5). Reducing this Composition by value, and at the same time ignoring the necessary condition $\alpha = \gamma$, would result in:

$$[[\lambda_\tau x\ [{}^0{=}\ {}^0 0\ x]]\ {}^0 2] \Rightarrow_\beta {}^2[{}^0 Sub\ [{}^0 Tr\ {}^0 2]\ {}^0 x\ {}^0[{}^0{=}\ {}^0 0\ x]]$$

While the left-hand side is improper, because the λ_τ-Closure constructs a degenerate function of type $(\tau\tau)$, there is no logical reason for the right-hand side to be improper. The right-hand side construction constructs the truth-value **F**. This is because the result of the substitution is the Composition $[{}^0{=}\ {}^0 0\ {}^0 2]$, the execution of which yields **F**. In other words, the following constructions are equivalent by constructing the same truth-value **F**:

$${}^2[{}^0 Sub\ [{}^0 Tr\ {}^0 2]\ {}^0 x\ {}^0[{}^0{=}\ {}^0 0\ x]]$$
$${}^{20}[{}^0{=}\ {}^0 0\ {}^0 2]$$
$$[{}^0{=}\ {}^0 0\ {}^0 2]$$

Claim 5
Let λ_α-Closure $[\lambda_\alpha\ x_1 \ldots x_m\ Y]$ v-construct a degenerate function $f/(\alpha\ \beta_1\ldots\beta_m)$ due to $Y \rightarrow_v \gamma$ and $\alpha \neq \gamma$. Then β-reduction by value:

$$[[\lambda_\alpha x_1 \ldots x_m\ Y]\ D_1,\ldots,D_m] \Rightarrow_\beta$$
$${}^2[{}^0 Sub\ [{}^0 Tr\ D_1]\ {}^0 x_1\ \ldots\ [{}^0 Sub\ [{}^0 Tr\ D_n]\ {}^0 x_n\ {}^0 Y]]$$

is not applicable.

Proof
According to Def. 3, in order that the rule of β-reduction by value be applicable, the types α and γ must be identical, which they fail to be here. □

Let us now examine the validity of β_r-reduction when the λ_α-Closure $[\lambda_\alpha\, x_1 \ldots x_m\, Y]$ *v*-constructs a function that is degenerate because $Y \rightarrow_v \gamma$, $\alpha \neq \gamma$. Consider again the λ_τ-Closure $[\lambda_\tau\, x\, [{}^0{=}\ {}^0 0\ x]]$. Composing this Closure with variable $y \rightarrow_v \tau$, we obtain:

$[[\lambda_\tau\, x\, [{}^0{=}\ {}^0 0\ x]]\ y] \Rightarrow_{\beta r} [{}^0{=}\ {}^0 0\ y]$

Unfortunately, the Composition $[{}^0{=}\ {}^0 0\ y]$ *is not* *v*-improper for any *v*, while the redex $[[\lambda_\tau\, x\, [{}^0{=}\ {}^0 0\ x]]\ y]$ *is* *v*-improper for any *v*.

Claim 6
Let λ_α-Closure $[\lambda_\alpha\ x_1 \ldots x_m\ Y]$ *v*-construct a degenerate function $f/(\alpha\ \beta_1 \ldots \beta_m)$ due to $Y \rightarrow_v \gamma$ where $\gamma \neq \alpha$. Further, let *Y* be *v*-proper for any valuation *v*, and let $y_1, \ldots, y_m$ be variables ranging over the respective types $\beta_1, \ldots, \beta_m$. Then the restricted β_r-reduction by name:

$[[\lambda_\alpha\, x_1 \ldots x_m\, Y]\ y_1, \ldots, y_m] \rightarrow_\beta Y(x_1{:}y_1, \ldots, x_m{:}y_m)$

is not a valid conversion.

Proof is obvious.

4. Concluding remarks

Above we have examined the conditions for the validity of the rule of β-reduction in the hyperintensional, typed λ-calculus of partial functions. While unconditional β-reduction by name is not a strictly equivalent transformation in the logic of partial functions, β-reduction by value and restricted β-reduction by name are strictly equivalent, hence valid conversions. If reduction by name is to be validly applied, then none of the constituents of the application procedure must be *v*-improper. This is the case of restricted β_r-reduction, which merely substitutes variables for λ-bound variables of the same respective types. Such a reduction is often applied in the analysis of empirical natural-language expressions.

In current TIL as expounded in Duží et al. (2010) and later, only λ-Closure has been considered while Tichý (1988) defined λ_α-Closure, $[\lambda_\alpha\, x_1 \ldots x_m\ Y]$. We showed that Tichý's definition leaves room for a procedure that produces a degenerate function even if Y is not v-improper for any valuation v. This interpretation is, however, fatal for β-reduction. The rule of β-reduction by value is not applicable, and the rule of restricted β_r-reduction is not valid even in case Y is not v-improper for any v, but Y is typed to v-construct objects of a type different from α. For this reason we recommend working only with λ-Closure, that is, with a λ_α-Closure $[\lambda_\alpha\, x_1 \ldots x_m\ Y]$ such that Y is typed to v-construct objects of type α.

For background, in programming languages the difference between β-reduction by name and by value revolves around the choice of *evaluation strategy*. Historically, call-by-value and call-by-name date back to *Algol 60*, a language designed in the late 1950s. The difference between call-by-name and call-by-value is often called *passing by reference* vs. *passing by value*, respectively. Strangely enough, purely functional programming languages such as *Clean* and *Haskell* use call-by-name. In our opinion, call by value would be a better evaluation strategy. For instance, *Java* manipulates objects by reference. However, *Java* does not pass arguments by reference, but by value. Call-by-value is not a single evaluation strategy, but rather a cluster of evaluation strategies in which a function's argument is evaluated before being passed to the function. In *call-by-reference* evaluation (also referred to as *call-by name* or *pass-by-reference*), a calling procedure receives an implicit reference to the argument sub-procedure. This typically means that the calling procedure can modify the argument sub-procedure. A call-by-reference language makes it more difficult for a programmer to track the effects of a procedure call, and may introduce subtle bugs.

Our proposal amounts to a *logical specification of an evaluation strategy by-value as adapted to TIL*. We have also developed a computational variant of TIL, the so-called *TIL-Script* language. For the reasons set out above, the grammar of the TIL-Script language does not make it possible to define λ_α-Closure, hence only λ-Closure is used. Finally, only the call-by-value reduction strategy is applied, which is thus universally applicable and valid.

Acknowledgments

This research has been supported by the Grant Agency of the Czech Republic, project No. GA15-13277S, *Hyperintensional logic for natural language analysis*, and also by the internal grant agency of VSB_TU Ostrava, project SGS No. SP2016/100.

References

BARENDREGT, H. P. (1997): The Impact of the Lambda Calculus. *Bulletin of Symbolic Logic* 3, 181-215.

CHANG, S. & FELLEISEN, M. (2012): The Call-by-Need Lambda Calculus, Revisited. *Programming Languages and Systems*. *Lecture Notes in Computer Science*. Vol. 7211, 128-147.

DUŽÍ, M. (2003): Do We Have to Deal with Partiality? In: Bendová, K. & Jirků, P. (eds.): *Miscellanea Logica* V. Prague: Karolinum, 45-76.

DUŽÍ, M., JESPERSEN, B., & MATERNA, P. (2010): *Procedural Semantics for Hyperintensional Logic. Foundations and Applications of Transparent Intensional Logic*. Berlin: Springer.

DUŽÍ, M. (2013): Deduction in TIL: From Simple to Ramified Hierarchy of Types. *Organon F* 20, Supplementary issue 2, 5-36.

DUŽÍ, M., & JESPERSEN, B. (2012): Transparent Quantification into Hyperintensional Contexts *de re*. *Logique et Analyse* 55, No. 220, 513–554.

DUŽÍ, M., & JESPERSEN, B. (2013): Procedural Isomorphism, Analytic Information and β-conversion by Value. *Logic Journal of the IGPL* 21, No. 2, 291-308.

DUŽÍ, M. (2014): Structural Isomorphism of Meaning and Synonymy. *Computación y Sistemas* 18, No. 3, 439-453.

DUŽÍ M., & JESPERSEN, B. (2015): Transparent Quantification into Hyperintensional Objectual Attitudes. *Synthese* 192, No. 3, 635-677.

FEFERMAN, S. (1995): Definedness. *Erkenntnis* 43, No. 3, 295-320.

MOGGI, E. (1988): *The Partial Lambda-Calculus*. *PhD thesis*. University of Edinburg, available as *LFCS report* at http://www.lfcs.inf.ed.ac.uk/reports/88/ECS-LFCS-88-63/.

PLOTKIN, G. D. (1975): Call-by-Name, Call-by-Value, and the Lambda Calculus. *Theoretical Computer Science* 1, No. 2, 125-159.

PLOTKIN, G. D. (1977): LCF Considered as a Programming Language. *Theoretical Computer Science* 5, No. 3, 223-255.

RACLAVSKÝ, J. (2009): *Jména a deskripce: logicko-sémantická zkoumání* [*Names and Descriptions: Logico-Semantical Investigations*]. Olomouc: Nakladatelství Olomouc.

TICHÝ, P. (1988): *The Foundations of Frege's Logic*. Berlin: de Gruyter.