Computación y Sistemas
ISSN: 1405-5546
computacion-y-sistemas@cic.ipn.mx
Instituto Politécnico Nacional
México

Duží, Marie
Structural Isomorphism of Meaning and Synonymy
Computación y Sistemas, vol. 18, núm. 3, julio-septiembre, 2014, pp. 439-453
Instituto Politécnico Nacional
Distrito Federal, México

Structural Isomorphism of Meaning and Synonymy

Marie Duží

VSB-Technical University Ostrava,
Czech Republic

marie.duzi@vsb.cz

Abstract. In this paper I am going to deal with the phenomenon of synonymy from the logical point of view. In Transparent Intensional Logic (TIL), which is my background theory, the sense of an expression is an algorithmically *structured procedure* detailing what operations to apply to what procedural constituents to arrive at the object (if any) denoted by the expression. Such procedures are rigorously defined as TIL *constructions*. In this new orthodoxy of structured meanings and *procedural semantics* we encounter the problem of the granularity of procedure individuation. Though the identity of TIL constructions is rigorously defined, they are a bit too fine-grained from the procedural point of view. In an effort to solve the problem we introduced the notion of *procedural isomorphism.* Any two terms or expressions whose respective meanings are procedurally isomorphic are deemed semantically indistinguishable, hence synonymous and thus substitutable in any context, whether extensional, intensional or hyperintensional. The novel contribution of this paper is a formally worked-out, philosophically motivated criterion of hyperintensional individuation, which is defined in terms of a slightly more carefully formulated version of α-conversion and β-conversion by value, which amounts to a modification of Church's Alternative (A1).

Keywords. Procedural semantics, β-conversion by value, procedural isomorphism, transparent intensional logic, synonymy.

1 Introduction

The phenomenon of synonymy has been of a central interest for both linguists and logicians, and though it is an important theoretical relation existing in language, a satisfactory criterion of synonymy is still a hot issue. A seemingly simple definition of synonymy as the identity of meaning evokes many problems including, inter alia, questions like what the meaning of an expression is and how fine-grained meanings should be.

This is a pressing issue, because many paradoxes and invalid inferences arise from a too coarse-grained analysis of premises. The purpose of logic is to differentiate between valid and invalid arguments so that all valid arguments be provable and invalid ones rejected. This in turn is closely connected with the granularity of individuation of meaning.

Possible-world semantics (PWS) that arose around 1960 models meanings as mappings defined on a domain of logical possibilities ('possible worlds'). Possible-world semantics gained respectability and proved to be highly deployable in various areas, in particular, in modal logic and its variants like temporal and deontic logics, by solving the granularity issue. Co-intensionality is tantamount to necessary co-extensionality. Hence, mappings that take the same worlds to the same extensions come out identical. Formally, let the variables f, g range over intensions and w over possible worlds:

$$\forall fg\ (\forall w\ (f(w) = g(w)) \rightarrow f = g)$$

The basic thing to understand about this individuation of possible-world intensions is that it offers an extensional account of intensional entities. If A, B are sets of possible worlds then if A, B share exactly the same elements then A is identical to B. This principle of individuation is generalized to properties, relations-in-intension, etc. The possible-world semantics for modal logic is firmly and comfortably immersed in set theory. Once necessary equivalence between intensions has been established, lots of valid inferences can be drawn, and many invalid inferences are blocked. Various mathematical properties, or their

Computación y Sistemas Vol. 18, No. 3, 2014 pp. 439–453
ISSN 1405-5546
DOI: 10.13053/CyS-18-3-2018

absence, of various formal systems of intensions can also be established, like soundness and completeness.

So far so good; however, already in 1947 [3, §§13*ff*] Carnap pointed out that there are attitudinal contexts that are neither extensional nor intensional, because the substitution of logically equivalent expressions for the complement of an attitude fails here. For instance, one can easily believe that it is false that *A* implies *B* without believing that *A* is true and *B* false. Yet the truth-conditions of these two embedded clauses are identical, hence they are indiscernible in possible-world semantics.

Moreover, possible-world semantics is out of place in case of mathematics; all true mathematical statements are necessarily equivalent, hence identical from the PWS point of view. For instance, the set of equilateral triangles is identical with the set of equiangular triangles, but the property of being equilateral is not identical with the property of being equiangular. Drawing an equilateral triangle is a task different from drawing an equiangular triangle, and one can be able to do the former without being able to do the latter.

Ludwig & Ray said:

"In general, one term can be substituted for another in 'that'-clauses salva veritate only if they are synonymous. Perhaps the most popular solution to the problem of providing a compositional semantics for natural languages aims to exploit this fact by treating 'that'-clauses as referring to intensional entities – entities (at least) as finely individuated as the meanings of sentences."

Thus since the late 60s, the issue of structured, hyperintensional meanings has been studied. The topic of hyperintensionality was born out of *negativity*, as it were. As mentioned above, Carnap noticed that there are attitudinal contexts that are neither extensional nor intensional, because the substitution of logically equivalent expressions fails here. Cresswell defines any individuation as hyperintensional that is finer than logical/necessary/strict equivalence. Hyperintensionality became originally a matter of *blocking* unwanted and unwarranted inferences, by pointing out that the correct substituends are hyperintensions. Indeed, any hyperintensional logic and formal semantics worth its name must be able to block various invalid inferences. But there is the other side of the coin, which is the *positive* topic of which inferences should be *validated*. That is, how hyper are hyperintensions? If there is one central question permeating hyperintensional logic and semantics then that is this one.

The problem how fine-grained 'intensional entities' hence meanings should be was of the utmost importance to Church who considered several alternatives of constraining these entities.[1] Senses are identical if the respective expressions are (A0) 'synonymously isomorphic', (A1) mutually λ-convertible, (A2) logically equivalent.[2] (A2), the weakest criterion, was refuted already by Carnap, and was not acceptable to Church as well. The alternative (A0) arose from Church's criticism of Carnap's notion of intensional isomorphism, and it is synonymy resting on α-equivalence and meaning postulates for semantically simple terms.

(A1) is presumably considered to be the right criterion of synonymy. Yet it was subjected to a fair amount of criticism in particular due to the involvement of β-reduction. For instance, Salmon in [17] adduces examples of expressions that should not be taken as synonymous yet their meanings are mutually β-convertible. Moreover, partiality throws a spanner in the works; β-reduction is not guaranteed to be an equivalent transformation as soon as partial functions are involved. Though Church's formal apparatus in which meanings are rendered is the typed λ-calculus of total functions, we cannot avoid the work with *partial* functions, because there are meaningful terms like the 'King of France' that lack a reference in the actual possible world and time. Or, in mathematics there are terms like 'the

[1] The concept of hyperintensionality is not exactly new; the terms 'hyperintensionality' and 'fine-grained intensionality' are. The standard term for this concept was simply 'intensionality' (maybe coined by Leibniz), and is still in use. However, possible-world semantics has used the terms 'intensionality', 'intensional logic', etc., for its own concept of intensionality.

[2] For details see [1] and [4].

greatest prime' that lack a reference regardless of possible worlds and times, they do not denote anything. Yet they have meaning.

Church also considered Alternative (A1') that is (A1) plus η-convertibility. Yet similar defects of η-convertibility as those connected with β-convertibility are evincible.

Thus the problem of the proper granularity of structured meanings remains open. This is a pressing issue, because in hyperintensional contexts only strictly synonymous expressions can be mutually substituted. We encounter the problem of hyperintensional contexts in particular in sentences expressing propositional attitudes like believing, knowing, etc., and objectual attitudes of seeking, wishing, solving, designing, calculating, and the like. Substitution of (merely) equivalent expressions yields the proliferation of agent's knowledge or abilities here, and the problem of logical/mathematical omniscience crops up.

In my background theory which is Tichý's Transparent Intensional Logic (TIL), hyperintensionality is defined in a positive rather than negative way.[3] Any context in which the meaning of an expression is *displayed* rather than *executed* is hyperintensional. Moreover, we explicate hyperintensions as abstract procedures rigorously defined as TIL *constructions* which are assigned to expressions as their context-invariant meanings.[4] The semantics is tailored to the hardest hyperintensional contexts, and generalized from there to simpler intensional and extensional ones. This entirely anti-contextual and compositional semantics is, to the best of my knowledge, the only one that deals with all kinds of context, whether extensional, intensional or hyperintensional, in a uniform way. The same extensional logical laws are valid invariably in all kinds of context. In particular, there is no reason why Leibniz's law of substitution of identicals, and the rule of existential generalization were not valid. What differ according to the context are not the rules themselves but the types of objects to which these rules are applicable. In an extensional context they are values of the functions denoted by the respective expression; in an intensional context they are the denoted functions themselves; finally in a hyperintensional context they are the displayed procedural meanings themselves. Due to its stratified ontology of entities organized in a ramified hierarchy of types, TIL is a logical framework within which such an extensional logic of hyperintensions has been introduced.[5]

In an effort to solve the problem of the procedural identity we introduced the notion of *procedural isomorphism.* Procedural isomorphism is a nod to Carnap's intensional isomorphism and Church's synonymous isomorphism. Any two terms or expressions whose respective meanings are procedurally isomorphic are deemed semantically indistinguishable, hence synonymous and substitutable in any context.

The goal of this paper is to define the rule for substitution of identicals in hyperintensional contexts. Since in such contexts only synonymous expressions can be mutually substituted, we need a criterion of synonymy. The novel contribution of this paper is the proposal of a new criterion of synonymy. It is an adjustment of Church's Alternative (A1). The adjustment consists in a more carefully formulated definition of α-conversion and the new definition of β-conversion, to wit, the β-conversion by value.

The rest of the paper is organized as follows. Section 2 sets out the logical foundations of TIL. The main results are introduced in Section 3 where the relation of procedural isomorphism, Alternative (A1''), is defined. I prove that the so defined relation is a strict equivalence on the set of constructions. Concluding remarks are in Section 4.

2 Logical Foundations of TIL

The syntax of TIL is Church's (higher-order) typed λ-calculus the terms of which are procedurally interpreted, which means that they denote structured *modes of presentation* (that is, TIL constructions) of functions rather than set-

[3] See [12, §§ 2.6, 2.7], and also [11].

[4] Similar conception of meaning has been propagated also by Moschovakis. For details see [15].

[5] See, for instance, [6], [7], [8], [9].

theoretic functions. Thus, lambda abstraction transforms into the molecular procedure of forming a function and application into the molecular procedure of applying a function to an argument. The identity of TIL constructions is rigorously determined by a definition. Yet we still have to deal with the issue of granularity of meanings/procedures, because TIL constructions are a bit too fine-grained from the procedural point of view. The main issue here is the following. Constructions that differ at most by using different λ-bound variables of the same type differ so slightly that from the semantic point of view they should be treated as identical procedures, because in natural languages we do not express λ-bound variables and thus do not reflect such differences.

Constructions are *the* key entities of TIL. They are algorithmically structured procedures, of one or multiple constituent parts and they serve to explicate linguistic meanings. Importantly, the constituent parts of a construction *C* are its *executed* subconstructions rather than the product (if any) of *C* which is located beyond *C.* Just to be clear, constructions are not set-theoretic mapping/functions, nor are they formulae or otherwise linguistic entities. Their inductive definition below enumerates six different constructions, which is the logical core of TIL. The stratified ontology of TIL is organized in the ramified hierarchy of types. For the sake of simplicity we first define *simple types of order 1*, then *constructions* together with the types of their products, and finally the *ramified hierarchy of types*.

Definition 1 (***types of order 1***). Let *B* be a *base*, where a base is a collection of pair-wise disjoint, non-empty sets. Then

(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 (α β_1 ... β_m) of all *m*-ary partial mappings from $\beta_1 \times ... \times \beta_m$ into α is a functional *type of order 1 over B*;

(iii) nothing else is a *type of order 1 over B*.

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

ο: a set of truth-values {**T**, **F**};
ι: a set of individuals (constant universe of discourse);
τ: a set of real numbers (doubling as temporal continuum);
ω: a set of logically possible worlds (logical space).

Within this type system we define possible-world *intensions* and *extensions*. An α-*intension* is a function of type (αω), or frequently ((ατ)ω), the types that we abbreviate as $\alpha_{\tau\omega}$; an α-*extension* is an object of type β, where $\beta \neq (\alpha\omega)$ for any α.

Definition 2 (*construction*).

(i) *Variable x* is a *construction* that *v-constructs* an object that a valuation *v* assigns to *x.*

(ii) The *Trivialization* ^0X is a *construction.* Let *X* be any object whatsoever. Then ^0X *constructs X* without any change of *X*.

(iii) *Composition* [*X* $Y_1 \ldots Y_m$] is a *construction*. Let *X v-construct* a function *f* of type (α $\beta_1 \ldots \beta_m$), and let $Y_1,\ldots,Y_m$ *v-construct* entities $B_1,\ldots,B_m$ of types $\beta_1,\ldots,\beta_m$, respectively. Then [*X* $Y_1 \ldots Y_m$] *v-constructs* the value (an entity, if any, of type α) of *f* on the tuple argument $\langle B_1,\ldots,B_m\rangle$. Otherwise the *Composition* [*X* $Y_1 \ldots Y_m$] does not *v-construct* anything and so is *v-improper*.

(iv) λ-*Closure* [λx_1 … x_m *Y*] (or *Closure*) is a *construction*. It *v-constructs* the following function *f* of type (α $\beta_1 \ldots \beta_m$). Let variables $x_1,\ldots x_m$ *v*-construct entities of types $\beta_1,\ldots,\beta_m$, and let *Y v-construct* an entity of type α. Let $v(B_1/x_1,\ldots,B_m/x_m)$ be a valuation identical with *v* at least up to assigning objects B_1/β_1, …, B_m/β_m to variables x_1, …, x_m. If *Y* is $v(B_1/x_1,\ldots,B_m/x_m)$-improper (see iii), then *f* is undefined on $\langle B_1, \ldots, B_m\rangle$. Otherwise the value of *f* on $\langle B_1, \ldots, B_m\rangle$ is the α-entity $v(B_1/x_1,\ldots,B_m/x_m)$-*constructed* by *Y*.

(v) *Single Execution* ^1X is a *construction.* Let *X v-construct* object *o*. Then ^1X *v-constructs o*. Let *X* be either a non-*construction* or a *v-*

improper *construction*. Then ^1X is *v-improper*.

(vi) *Double Execution* ^2X is a *construction.* Let X *v-construct* a *construction* Y and let Y *v-construct* an object Z (possibly itself a *construction*). Then the *Double Execution* ^2X *v-constructs* Z. Otherwise ^2X is *v-improper*.

(vii) Nothing else is a *construction*.

Here are some explicative remarks. A *variable* constructs an object by having that object as its value dependent on a valuation function v arranging variables and objects in a sequence. *Trivialization* is TIL's objectual counterpart of a non-descriptive constant term, which simply provides a particular object. Variables and Trivializations are the *atomic* constructions of TIL, Composition, Closure and Executions are the *molecular* constructions. An *atomic* construction is a structured whole with but one constituent part, namely, the construction itself. A *molecular* construction is a structured whole with more parts than just itself. Importantly, the only part of ^0X is ^0X and not X, which is located beyond ^0X: the product of a procedure is no part of the procedure.

The definition of the typed universe of TIL amounts to a definition of the ramified hierarchy of types which is divided into three parts: firstly, simple types of order 1, which were already defined by Definition 1; secondly, constructions of order n; thirdly, types of order n+1.

Definition 3 (ramified hierarchy of types).

T_1 (types of order 1). See Def. 1.

C_n (constructions of order n).

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

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

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

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

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

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

i) $*_n$ and every type of order n are *types of order n*+1 over B;

ii) if $m > 0$ and $\alpha, \beta_1,...,\beta_m$ are types of order n+1 over B, then $(\alpha\ \beta_1\ ...\ \beta_m)$ (see T_1 ii)) is a *type of order n* + 1 *over B*;

iii) nothing else is a *type of order n* + 1 *over B*.

Logical objects like *truth-functions* and *quantifiers* are extensional: $\wedge$ (conjunction), $\vee$ (disjunction) and $\supset$ (implication) are of type $(o\,o\,o)$, and $\neg$ (negation) of type $(o\,o)$. The *quantifiers* $\forall^\alpha$, $\exists^\alpha$ are type-theoretically polymorphous, 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. Furthermore, 'X/α' means that an object X is (a member) of type α. '$X \rightarrow_v \alpha$' means that the type of the object *valuation*-constructed by X is α. This holds throughout: the variables $w \rightarrow_v \omega$ and $t \rightarrow_v \tau$. If $C \rightarrow_v \alpha_{\tau\omega}$ then the frequently used Composition $[[Cw]\ t]$, which is the intensional descent (a.k.a. extensionalization) of the α-intension v-constructed by C, will be encoded as 'C_{wt}'. When using constructions of truth-functions, we often omit Trivialization and use infix notation to conform to standard notation in the interest of better readability. Also when using constructions of identities of α-entities, $=_\alpha/(o\alpha\alpha)$, we omit Trivialization, the type subscript, and use infix notion when no confusion can arise. Moreover, the outermost brackets of Closure will be occasionally omitted.

Definition 4 (**subconstruction**). Let C be a construction. Then

i) C is a *subconstruction of C*;

ii) if C is ^0X, ^1X or ^2X and X is a construction then X is a *subconstruction of C*;

iii) if C is $[X\ X_1 \ldots X_n]$ then X, X_1, …, X_n are *subconstructions of C*;
iv) if C is $[\lambda x_1 \ldots x_n\ Y]$ then Y is a *subconstruction of C*;
v) if A is a *subconstruction of B* and B is a *subconstruction of C* then A is a *subconstruction of C*;
vi) a construction is a *subconstruction of C* only if it so follows from (i) – (v).

There are two modes in which a subconstruction D of a construction C may occur, to wit, *displayed* and *executed*. If the latter, then we say that D is a *constituent* of C. The Trivialization ${}^0C$ of a construction C displays the construction C *and* all the subconstructions of C. Hence C is not a constituent part of ${}^0C$; it is not executed, and so does not obtain an object beyond it. We say that C occurs hyperintensionally. It is however important to realize that Double Execution executes a construction twice over. Thus in ${}^{20}C$ the subconstruction C is a constituent part of ${}^{20}C$.

If C is an executed constituent of D then C can occur *intensionally* or *extensionally.* In principle, constituent C occurs in D intensionally if C is not composed with a construction of the argument of the function f v-constructed by C. Hence the whole function f is an object of predication within D. As a limiting case, a constituent C that constructs an atomic entity, which is a 0-ary function without arguments, occurs intensionally. On the other hand, a constituent C of D occurs extensionally if it is composed with a construction of an argument of the function f, and C does not occur within an intensional context of D. Hence the *value* (if any) of the function f is an object of predication within D.

These three ways in which a subconstruction C of a construction D can occur give rise to three kinds of context within D:[6]

– *hyperintensional context*: a construction C occurs *displayed* and serves itself as a functional argument to be operated on in D (though a construction one order higher needs to be executed in order to produce the displayed construction);
– *intensional context*: construction C occurs *executed* in order to produce a function f but not the value of f; moreover, the executed construction C does not occur within another hyperintensional context. (Hence the entire *function v*-constructed by C serves as a *functional argument* to be operated on in D);
– *extensional context*: construction C occurs *executed* in order to produce a particular value of the function v-constructed by C; moreover, the executed construction does not occur within another intensional or hyperintensional context. (Hence the *value* of the function v-constructed by C serves as a *functional argument* to be operated on in D).

Higher context is dominant over a lower one. It means that all the subconstructions of a displayed construction occur hyperintensionally as well, and all the subconstructions of an executed construction that occurs intensionally occur intensionally as well.

Example. Below I analyze three sentences in which the meaning of the term 'temperature in Prague' denoting a magnitude of type $\tau_{\tau\omega}$ occurs extensionally, intensionally and hyperintensionally, respectively. Note that there is no reference shift and the meaning and denotation of this term is the same in all three types of context. It is the construction of the denoted magnitude, to wit, the Closure

$$\lambda w \lambda t\, [{}^0\mathit{Temperature\_in}_{wt}\ {}^0\mathit{Prague}] \rightarrow \tau_{\tau\omega}\,.$$

a) Extensional context:

"*The temperature in Prague is 30^0 Celsius*."

The types of the objects that the sentence talks about are these: *Temperature_in*/$(\tau\iota)_{\tau\omega}$: attribute; *Prague*/ι; $\lambda w \lambda t\ [{}^0\mathit{Temperature\_in}_{wt}\ {}^0\mathit{Prague}] \rightarrow \tau_{\tau\omega}$: magnitude; $30C/\tau$.

The whole sentence denotes a proposition of type $o_{\tau\omega}$ constructed by this Closure:

$$\lambda w \lambda t\, [\lambda w \lambda t\, [{}^0\mathit{Temperature\_in}_{wt}\ {}^0\mathit{Prague}]_{wt} = {}^0 30\mathrm{C}].$$

In the Composition

$$[\lambda w \lambda t\, [{}^0\mathit{Temperature\_in}_{wt}\ {}^0\mathit{Prague}]_{wt} = {}^0 30C],$$

[6] Here I present just a summary. For exact definitions see [12, §2.6].

the construction $\lambda w\lambda t\,[{}^0\mathit{Temperature\_in}_{wt}\ {}^0\mathit{Prague}]$ of the magnitude occurs extensionally, because the value of this magnitude in a given world *w* at time *t* of evaluation is constructed, and this value is an object of predication, to wit, this value equals 30*C*. Within this Composition the construction ${}^0\mathit{Temperature\_in}$ of the attribute occurs extensionally as well.

b) Intensional context:

"*The temperature in Prague is rising*."

Additional *type*. *Rising*/$(o\tau_{\tau\omega})_{\tau\omega}$: the property of a magnitude.

The analysis of the sentence is the following:

$$\lambda w\lambda t\,[{}^0\mathit{Rising}_{wt}\ \lambda w\lambda t\,[{}^0\mathit{Temperature\_in}_{wt}\ {}^0\mathit{Prague}]].$$

Now, $\lambda w\lambda t\,[{}^0\mathit{Temperature\_in}_{wt}\ {}^0\mathit{Prague}]$ occurs intensionally in the Composition

$$[{}^0\mathit{Rising}_{wt}\ \lambda w\lambda t\,[{}^0\mathit{Temperature\_in}_{wt}\ {}^0\mathit{Prague}]].$$

The function (magnitude) rather than its value is an object of predication that this function is rising. Due to the dominancy of a higher context over a lower one, the construction ${}^0\mathit{Temperature\_in}$ of the attribute occurs intensionally as well, though it is extensionalized (composed with *w* and *t*) and composed with the construction ${}^0\mathit{Prague}$ of its argument.

c) Hyperintensional context:

"*Tilman believes that the temperature in Prague is* 30^0 *Celsius*."

Additional *type*. *Believe*/$(o\iota*_n)_{\tau\omega}$: relation-in-intension between an individual and a hyperproposition.

The analysis of the sentence is this:

$$\lambda w\lambda t\,[{}^0\mathit{Believe}_{wt}\ {}^0\mathit{Tilman}\ {}^0[\lambda w\lambda t\,[{}^0\mathit{Temperature\_in}_{wt}\ {}^0\mathit{Prague}]_{wt} = {}^0\mathrm{30C}]].$$

Here we construe *Believe* as a relation-in-intension of an individual to a hyperproposition. The reason is this: the proposition that the temperature in Prague is 30^0 Celsius can be equivalently expressed, e.g., that the temperature in Prague is 86^0 Fahrenheit. Yet one can believe the former without believing the latter, and vice versa.

Hence $\lambda w\lambda t\,[{}^0\mathit{Temperature\_in}_{wt}\ {}^0\mathit{Prague}]$ occurs hyperintensionally in the Composition

$$[{}^0\mathit{Believe}_{wt}\ {}^0\mathit{Tilman}\ {}^0[\lambda w\lambda t\,[{}^0\mathit{Temperature\_in}_{wt}\ {}^0\mathit{Prague}]_{wt} = {}^0\mathrm{30C}]].$$

This is due to the fact that the mode of presentation, or conceptualization, or construction

$$[\lambda w\lambda t\,[{}^0\mathit{Temperature\_in}_{wt}\ {}^0\mathit{Prague}]_{wt} = {}^0\mathrm{30C}]$$

of the proposition that the temperature in Prague is 30^0C is an object of Tilman's belief rather than the proposition itself. Thus all the subconstructions of this construction occur hyperintensionally as well. In brief, the left most Trivialization in ${}^0[\lambda w\lambda t\,[{}^0\mathit{Temperature\_in}_{wt}\ {}^0\mathit{Prague}]_{wt} = {}^0\mathrm{30}\mathit{C}]$ raises the context up to the hyperintensional level.

3 Procedural Isomorphism

As mentioned at the outset, TIL can be viewed as an extensional logic of hyperintensions. By 'extensional' I mean that the extensional rules like existential generalization and Leibniz's laws of substitutivity of identicals are validly applicable in all contexts, whether extensional, intensional or hyperintensional. The rules of existential quantification in intensional and hyperintensional contexts have been specified by Duží & Jespersen in [8], [9] and [11].

As for the substitution of identicals in an extensional or intensional context, there is no problem. In an *intensional* context analytically *equivalent* terms are mutually substitutable, while in an *extensional* context also *co-referring* terms are substitutable.

Terms are *co-referring* if they refer to the same object in a given world *w* and time *t* of evaluation. More generally, terms are co-referring if their meanings are *v*-congruent, that is *v*-construct functions that happen to have the same value at the same argument. Terms are analytically *equivalent* if their meanings *v*-construct the same object for every valuation *v*. Obviously, equivalent terms are co-referring, but not vice versa.

Thus, for instance, the following argument is valid.

The temperature in Amsterdam is 30^0 Celsius.
The temperature in Amsterdam is the same as in Prague.
———————————————————
The temperature in Prague is 30^0 Celsius.

The 'temperature in Amsterdam' as well as the 'temperature in Prague' occur extensionally both in the premises and the conclusion. Moreover, the second premise establishes their co-reference, as the analysis reveals:

$$[\lambda w \lambda t\, [[^0\mathit{Temperature\_in}_{wt}\ ^0\mathit{Amsterdam}]_{wt} = \lambda w \lambda t\, [^0\mathit{Temperature\_in}_{wt}\ ^0\mathit{Prague}]_{wt}].$$

On this assumption the extensional occurrences of these two terms are mutually substitutable. However, this argument is invalid:[7]

The temperature in Amsterdam is rising.
The temperature in Amsterdam is the same as in Prague.
———————————————————
The temperature in Prague is rising.

The reason is this: in the first premise 'the temperature in Amsterdam' occurs intensionally. Particular value of the magnitude cannot be rising; rather, being rising is a property of the whole function, magnitude in this case. However, the second premise establishes only co-reference of the two terms rather than equivalency. Hence the substitution is not valid, because in such an intensional context only equivalent terms can be substituted.

However, substitution of (analytically) equivalent terms is not valid in *hyperintensional* contexts. From a linguistic point of view, in a hyperintensional context only synonymous expressions can be substituted, because the very meaning of expressions is displayed. Our thesis is that synonymous expressions have structurally isomorphic meanings. And since meaning is a procedure, we need to define the relation of *procedural isomorphism* between constructions, because constructions are a bit too fine-grained from the procedural point of view. The main issue here is the following. Constructions that differ at most by using different λ-bound variables of the same type differ so slightly that we wish to say that such constructions are one and the same procedure. For instance, the Closures

$$\lambda w \lambda t\, [\lambda x\, [[^0\mathit{Happy}_{wt}\ x] \wedge [^0\mathit{Child}_{wt}\ x]]],$$
$$\lambda w \lambda t\, [\lambda y\, [[^0\mathit{Happy}_{wt}\ y] \wedge [^0\mathit{Child}_{wt}\ y]]],$$
$$\lambda w \lambda t\, [\lambda z\, [[^0\mathit{Happy}_{wt}\ z] \wedge [^0\mathit{Child}_{wt}\ z]]], \ldots,$$

are by Def. 2 different constructions of the property of being a happy child. Yet from the procedural point of view they are isomorphic. They consist of these procedural steps:

- in any world w (λw) at any time t (λt) do this:
- take *any* individual x (or y, or z, …),
- take the property of being happy (by $^0\mathit{Happy}$),
- apply the (extensionalized) property of being happy ($^0\mathit{Happy}_{wt}$) to the chosen individual x ($[^0\mathit{Happy}_{wt}\ x]$), or to y, or z, …,
- take the property of being a child (by $^0\mathit{Child}$),
- apply the (extensionalized) property of being a child ($^0\mathit{Child}_{wt}$) to the chosen individual x (or y, or z, …),
- check whether the chosen individual has both the properties ($^0\wedge$),
- abstract over the chosen individual (λx, or λy, or λz, …).

Using the terminology of λ-calculus, the above Closures are α-equivalent. Church's Alternative (A0) includes α-conversion and meaning postulates for atomic constants such as 'bachelor' and 'fortnight'. Of course, we need meaning postulates to specify synonymy of semantically simple terms. This is a matter of building up linguistic ontology. Now we are, however, interested in the synonymy of molecular terms, which is structural isomorphism.

α-conversion is certainly the rule that must be included. The question is, which other rules (and whether any other rules) could be included as well. Church's (A0) and (A1) leave room for additional Alternatives. To this end we consider β- and η-conversion. Yet, as explained above, we are not content with Church's Alternatives (A1) or (A1') due to their non-equivalency.

In this paper I suggest a new definition of the criterion of structured synonymy, (A1″). This variant is close to Church's (A1) and includes an adjusted versions of α- and β-conversion. Thus we exclude η-conversion, and introduce a new version of β-conversion, to wit, the conversion *by value*. In this proposal I follow two necessary

[7] I use a variant of Barbara Partee's example.

conditions that the meanings of synonymous expressions should meet. They must be (a) strictly equivalent in the sense that in every possible world at any time they either denote the same entity or lack a denotation, and (b) their meanings must have the same constituents.

There are two reasons for not including η-conversion. First, it does not satisfy the condition (b). The η-expanded construction of the form $\lambda x\,[F\,x]$ has two more constituents than the η-reduced construction F, because the former adds the steps of applying the function *v*-constructed by F to the value *v*-constructed by the variable x followed by the abstraction over the value of x. The second reason is the fact that η-conversion does *not preserve logical equivalence* in a logic of *partial functions* such as TIL. Hence it does not satisfy the condition (a). To see this, consider the function $F \to ((\alpha\beta)\gamma)$ that *v*-constructs a function that is not defined at the argument *v*-constructed by $A \to \gamma$. Then the Composition $[F\,A] \to (\alpha\beta)$ is *v*-improper. However, the η-expanded construction $\lambda x\,[[F\,A]\,x] \to (\alpha\beta)$, $x \to \beta$, *v*-constructs a *degenerate function*, which is a function undefined at all its arguments. To be sure, due to the *v*-improperness of $[F\,A]$, the Composition $[[F\,A]\,x]$ is also *v*-improper. But λ-abstraction raises the context to an intensional one, hence the Closure $\lambda x\,[[F\,A]\,x]$ *v*-constructs a degenerate function, which *is* an object, if a bizarre one. Hence the constructions $[F\,A]$ and $\lambda x\,[[F\,A]\,x]$ are not strictly equivalent.[8]

In practice the exclusion of η-conversion from the definition of procedural isomorphism is going to be harmless. When analyzing expressions in TIL we apply our method of *literal analysis*, which consists of three steps: (i) assigning types to the objects mentioned by the sub-terms of the analyzed expression E; (ii) combining the Trivializations of the objects mentioned by the semantically simple sub-terms of E in order to obtain the construction of the object (if any) denoted by E; (iii) checking whether the resulting construction is type-theoretically coherent. Due to step (ii) the application of this method yields a construction (namely the meaning of E) that does not contain η-expanded subconstructions. For instance, the literal analysis of "The Pope is wise" is the Closure

$$\lambda w\lambda t\,[{}^0Wise_{wt}\,{}^0Pope_{wt}]$$

rather than

$$\lambda w\lambda t\,[\lambda w\lambda t\,[\lambda x\,[{}^0Wise_{wt}\,x]]_{wt}\,{}^0Pope_{wt}],$$

because the literal analysis of the predicate 'is wise' is the Trivialization ${}^0Wise$ rather than the Closure $\lambda w\lambda t\,[\lambda x\,[{}^0Wise_{wt}\,x]]$. The types are $Wise/(o\iota)_{\tau\omega}$; $Pope/\iota_{\tau\omega}$; $x \to \iota$.

3.1 β-Conversion: 'by Name' vs. 'by Value'

The reasons for excluding unrestricted β-conversion are similar to the above. Though it is the fundamental computational rule of the λ-calculi, it is underspecified by the commonly acknowledged rule

$$[[\lambda x\,C(x)]\,A] \mid\!-\, C(A/x).^9$$

The procedure of applying the function *v*-constructed by $[\lambda x\,C(x)]$ to the argument *v*-constructed by A can be executed in two different ways: *by value* or *by name*. If by name then the *procedure* A is substituted for all the occurrences of x into C. In this case there are two problems.

First, conversion by name is not guaranteed to be a strictly equivalent transformation as soon as partial functions are involved. This is due to the fact that A occurs in an extensional context of the left-hand side construction, whereas when dragged into C its occurrence may become intensional or hyperintensional provided the context in which x occurs in C is intensional or even hyperintensional. For instance, the Composition

$$[\lambda x\,[\lambda y\,[{}^0+\,x\,y]]\,[{}^0:\,{}^0 3\,{}^0 0]]$$

is improper, because $[{}^0:\,{}^0 3\,{}^0 0]$ is improper. This is as it should be, for there is no value that might be substituted for the formal parameter x, because the Composition $[{}^0:\,{}^0 3\,{}^0 0]$ is improper by failing to

[8] I am grateful to J. Raclavský for calling our attention to this problem. See also Raclavský (2010).

[9] For the sake of simplicity, we now consider a unary function. Generalization for *n*-ary functions is obvious.

produce a result. However, the 'by-name' β-reduced construction

$$[\lambda y\, [{}^0+\, [{}^0{:}\ {}^0 3\ {}^0 0]\ y]]$$

is *not* improper as it constructs a degenerated function undefined at all its arguments. The improper construction $[{}^0{:}\ {}^0 3\ {}^0 0]$ has been drawn into the intensional context of the Closure $[\lambda y\, [{}^0+\, x\, y]]$.

The same problem crops up in the analysis of *de re* attitudes. For instance, the *de re* attitude expressed by

"*The Pope is believed by a to be wise*"

obtains on its intensional reading the analysis[10]

$$\lambda w \lambda t\, [\lambda x\, [{}^0 Believe_{wt}\, a\, \lambda w \lambda t\, [{}^0 Wise_{wt}\, x]]\ {}^0 Pope_{wt}].$$

In a world *w* and time *t* when the Pope does not exist, the Composition ${}^0 Pope_{wt}$ is *v*-improper and the so constructed proposition lacks a truth-value, because there is no individual to whom the property of being believed by *a* to be wise would be ascribable.

This is as it should be, because in the *de re* case there is an existential presupposition that the Pope exists. Yet the β-reduction by name transforms this analysis into the *de dicto* attitude:

$$\lambda w \lambda t\, [{}^0 Believe_{wt}\, a\, \lambda w' \lambda t'\, [{}^0 Wise_{w't'}\ {}^0 Pope_{wt}]].$$

The so constructed proposition can be true even in a ⟨*w,t*⟩-pair in which the Pope does not exist, which is not correct.

The second reason for refuting an unrestricted β-reduction by name is this: even in those cases when β-reduction by name is an equivalent transformation, it may yield loss of analytic information about which function has been applied to which argument.[11] For instance, the Composition

$$\lambda w \lambda t\, [[\lambda x\, [{}^0 Larger_{wt}\, x\, x]]\ a]$$

which is the meaning of "*a* is larger than itself" reduces to $\lambda w \lambda t\, [{}^0 Larger_{wt}\, a\, a]$, the meaning of "*a* is larger than *a*". Yet the two sentences are not strictly synonymous, because in the former the property of being larger than itself is applied to *a* while in the latter the binary relation larger than is applied to the pair (*a*,*a*).[12]

The idea of *conversion by value* is simple. Execute the procedure *A* first, and only if *A* does not fail to produce an argument value on which *C* is to operate, substitute (Trivialization of) this *value* for *x.* The solution preserves strict logical equivalence, avoids the problem of the loss of analytic information, and moreover, in practice it is more efficient. The efficiency is guaranteed by the fact that the procedure *A* is executed only once, whereas if this procedure is substituted for all the occurrences of the λ-bound variable it can subsequently be executed more than once.

To elucidate the problem even more, comparison with programming languages might be useful. Imagine one has a procedure (embodied as a program) *C*(*x*) with a 'hole' *x* (i.e., an unsaturated procedure with a formal parameter *x*), and a subprogram *A* that specifies the material (argument value) to be filled into the hole *x.* There are two ways of going about filling *x*:

(1) (*by name*) inserting into the hole *x* the whole subprogram *A* and then computing *C*(*A/x*)

(2) (*by value*) computing *A* first in order to obtain the argument value *a*, and then inserting *a* into the hole *x* and computing *C*(*a/x*)

In case (1) there may be undesirable side effects. Imagine that the subprogram *A* is somehow garbled and as a result the whole procedure *C* gets garbled as well by the insertion, damage being propagated upwards. Moreover, instead of the hole *x* one gets *A*, and *A* may conflict with *C*. This is a case of invalid β-reduction that fails to preserve equivalence. Furthermore, even if *A* does not damage *C* when computing *C*(*A*), after the execution of *C*(*A*) one will have lost track of *A.* The two procedures have been merged together. Suppose one wants to compute another procedure *E*(*x*) and to supply the same material for *x.* Even if the execution of *C*(*A*) turns out to be successful, *A* may have been changed by the execution. There is no guarantee that the same material will be supplied for *x* into

[10] In this case a hyperintensional analysis would be also the choice. For the sake of simplicity we consider the intensional one.

[11] For the notion of analytic information see [5].

[12] See [17].

E(*x*). This is the case of β-reduction preserving equivalence but yielding loss of analytic information.

In case (2) we keep *C*(*x*), *E*(*x*), and *A* separate. Procedure *A* is evaluated only if needed, and if so, only once. Everything is as it should be: no loss of analytic information arises and equivalence is preserved.

In programming languages the difference between cases 1 and 2 concerns *evaluation strategy* and is often called *passing by reference* vs. *passing by value*, respectively. Historically, call-by-value and call-by-name date back to *Algol 60*, a language designed in the late 1950s. Only purely functional languages such as *Clean* and *Haskell* use call-by-name. For instance, *Java* manipulates objects by reference that is by name. 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 calling procedure. 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.

The notion of *reduction strategy* in the λ-calculi is similar to the *evaluation strategy* in programming languages. My proposal amounts to a *specification of an evaluation strategy by-value as adapted to TIL* that is to hyperintensional, partial typed λ-calculus. Similar work has been done since the early 1970s, but merely for simple-typed or untyped λ-calculi. For instance, Plotkin in [16] proved that the two strategies are not operationally equivalent. Hence the call-by-name strategy *should not* be used in hyperintensional λ-calculi such as TIL due to operational non-equivalence. Our *substitution method* that I am going to define below is similar to Chang & Felleisen call-by-need reduction by value (see [2]). But their work is couched in an untyped λ-calculus. TIL, by contrast, is a hyperintensional, partial λ-calculus based on ramified theory of types. First we need to define the substitution function:

Definition 5 (***Sub**^{n}*). Function $Sub^n/(*_n*_n*_n*_n)$ operates on constructions in this way: let constructions C_1, C_2, C_3 *v*-construct constructions (of order *n*) D_1, D_2, D_3, respectively. Then the Composition $[^0Sub^n\ C_1\ C_2\ C_3]$ *v*-constructs the construction *D* that results from D_3 by collisionless substitution of D_1 for all occurrences of D_2 in D_3.

Occasionally we also need the polymorphic function Tr^α defined as follows.

Definition 6 (***Tr**$^\alpha$*). The function $Tr^\alpha/(*_n\ \alpha)$ returns as its value the Trivialization of its α-argument.

In what follows I simply write '*Sub*' and '*Tr*' omitting thus the type-superscripts whenever no confusion arises.

For instance, let variable *y* *v*-construct numbers of type τ such as π. Then $[^0Tr\ y]$ *v(π/y)*-constructs $^0\pi$. Therefore, the Composition

$$[^0Sub\ [^0Tr\ y]\ ^0x\ ^0[^0Sin\ x]]$$

v(π/y)-constructs the Composition $[^0Sin\ ^0\pi]$.

Note that there is a substantial difference between the *construction* Trivialization and the *function* Tr^α. Whereas ^0y constructs just the variable *y* regardless of valuation, *y* being ^{0}-bound in ^0y, $[^0Tr\ y]$ *v*-constructs the Trivialization of the object *v*-constructed by *y*. Hence *y* occurs free in $[^0Tr\ y]$.

Definition 7 ***(β-conversion by value)*** *Let* $Y \to_v \alpha$; x_1, $D_1 \to_v \beta_1$,…, x_n, $D_n \to_v \beta_n$, $[\lambda x_1 \ldots x_n\ Y] \to_v (\alpha\beta_1 \ldots \beta_n)$. 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-*conversion by value*.

Note that in the converted construction Double Execution is necessary. The function *Sub* operates on argument constructions, and as a result it produces again a construction; to wit, the construction in which all occurrences of the formal parameters x_1, …, x_n are replaced by the Trivializations of the respective argument values. In order to obtain an α-value (if any) produced by

the so pre-processed construction *Y* the resulting construction must be executed. Hence, Double Execution is performed.

Claim 1: 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 in the sense that for any valuation *v* they either *v*-construct one and the same entity or are both *v*-improper.

Proof. Let *C* be $[[\lambda x_1 \ldots x_n\ Y]\ D_1 \ldots D_n]$ and let *D* be $^2[^0Sub\ [^0Tr\ D_1]\ ^0x_1\ \ldots\ [^0Sub\ [^0Tr\ D_n]\ ^0x_n\ ^0Y]]$.

If some of the constructions $D_1,\ldots,D_n$ is *v*-improper then so are both *C* and *D*, according to Def. 2, iii) and vi). Otherwise, let $D_1,\ldots,D_n$ all be *v*-proper, *v*-constructing the objects $d_1,\ldots,d_n$, respectively. Then by Def. 2, iv) the Closure $[\lambda x_1 \ldots x_n\ Y]$ *v*-constructs the following function *f*. 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 *C* is $v(d_1/x_1,\ldots,d_n/x_n)$-improper according to Def. 2, iii). Otherwise 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 the entity $v(d_1/x_1,\ldots,d_n/x_n)$-constructed by *Y* be *e*. Then by Def. 2, iii) of Composition, the construction *C* *v*-constructs *e*. We are to show that the construction *D* also *v*-constructs *e*. The first Execution of *D* constructs $Y(x_1/^0d_1,\ldots,\ x_n/^0d_n)$, i.e., the construction *Y* where according to the definition of the functions *Sub* and *Tr* all the occurrences of variables $x_1,\ldots,x_n$ are replaced by $^0d_1,\ldots,^0d_n$, respectively. Since the Trivializations $^0d_1,\ldots,^0d_n$ construct the entities $d_1,\ldots,d_n$, respectively, the second Execution $v(d_1/x_1,\ldots,d_n/x_n)$-constructs the entity *e*, or else nothing in case *Y* is $v(d_1/x_1,\ldots,d_n/x_n)$-improper. Hence *C* and *D* come out strictly equivalent.

3.2 Alternative (A1'')

At the outset of this paper I formulated the problem of granularity of individuation meanings. Any individuation that is finer than necessary or logical equivalence qualifies as hyperintensional. Since we explicate hyperintensions procedurally, the problem transforms into the issue of individuation of procedures. In other words, under which conditions are we ready to consider two constructions as procedurally isomorphic? Church considered α-, β- and η-conversion. In the previous paragraph I summarized objections against Church's Alternatives (A1) or (A1'), and it should be clear now that η-conversion should not be included. An obvious plausible candidate is α-conversion, which would yield Alternative (A0).

In my opinion, we also need β-conversion, because application of a function to an argument is the very fundamental computational procedure. However, the objections against unrestricted β-conversion (by name) are serious. Fortunately there is a way out. The above defined β-conversion by value is immune against those objections. It is the correct way of applying the function constructed by the Closure $[\lambda x_1 \ldots x_n\ Y]$ to the arguments constructed by $D_1,\ldots,D_n$. According to Claim 1, it is a strictly equivalent conversion, unlike β-conversion by name. Moreover, it does not yield loss of analytic information, because the calling procedure *Y* and argument procedures $D_1,\ldots,D_n$ are kept separated.

Now I will demonstrate the thesis that β-value equivalent constructions should be considered as procedurally isomorphic. For the sake of simplicity I will again consider a one-argument Composition of the form $[\lambda x\ [F\ x]\ A]$. It is the specification of calling the procedure $\lambda x\ [F\ x]$ with the formal parameter *x* at the argument provided by the procedure *A*. My thesis is that the correct way of executing this procedure consists of these execution steps:

- execute *A* in order to obtain the argument value *a*; if *A* fails to *v*-construct anything (is *v*-improper) then abort the execution; else:
- take the argument value *a* (by the Trivialization of *a*) and substitute it for all the occurrences of the variable *x* in the procedure body *F*, and finally:
- Execute the result of the substitution.

The Composition $^2[^0Sub\ [^0Tr\ A]\ ^0x\ ^0F]$ has exactly the same constituents. These are

- *A*: execute *A* in order to obtain the argument value *a*; if *A* is *v*-improper then the entire Composition is *v*-improper; else:
- $[^0Tr\ A]$: obtain the Trivialization of *a*,
- $[^0Sub\ [^0Tr\ A]\ ^0x\ ^0F]$: substitute the Trivialization of *a* for *x* in *F*,
- $^2[^0Sub\ [^0Tr\ A]\ ^0x\ ^0F]$: execute the result.

We can see that β-conversion by value is an explicit specification of the procedure of applying the function constructed by λx [*F x*] to the argument constructed by *A*. Yet one might object that the above execution steps are not explicitly specified by the term '[λx [*F x*] *A*]'. This is true but this term can be taken as an abbreviation of the full-fledged application specification provided by $^{2}[^{0}Sub\ [^{0}Tr\ A]\ ^{0}x\ ^{0}F]$'. In other words, we can consider the left-hand side of the β-value rule as *definiendum* and the right-hand side as *definiens*.

For this reason it is reasonable to claim the two terms '[λx [*F x*] *A*]' and $^{2}[^{0}Sub\ [^{0}Tr\ A]\ ^{0}x\ ^{0}F]$' *ex definitione* synonymous and thus substitutable in all contexts, including hyperintensional ones.

In order to define *procedural isomorphism* on the set of constructions, we still need another definition, to wit, the definition of α-conversion. The standard definition that defines α-equivalent constructions as those that differ at most by using different λ-bound variables does not do. The reason is this: if β-value equivalent constructions of the form [λx [*F x*] *A*] and $^{2}[^{0}Sub\ [^{0}Tr\ A]\ ^{0}x\ ^{0}F(x)]$ are procedurally isomorphic, then it does not matter which particular variable of the respective type, whether *x*, or *y*, or *z*, …, has been used as a formal parameter. Hence since

$$[\lambda x\ [F\ x]\ A],\ [\lambda y\ [F\ y]\ A],\ \ldots$$

are α-equivalent, so should be

$$^{2}[^{0}Sub\ [^{0}Tr\ A]\ ^{0}x\ ^{0}F(x)],\ ^{2}[^{0}Sub\ [^{0}Tr\ A]\ ^{0}y\ ^{0}F(y)],\ \ldots$$

For instance, constructions

$$[\lambda x\ [^{0}+\ x\ ^{0}1]\ ^{0}5]$$
$$[\lambda y\ [^{0}+\ y\ ^{0}1]\ ^{0}5]$$

are α-equivalent according to the standard definition. Yet their β-reduced forms

$$^{2}[^{0}Sub\ [^{0}Tr\ ^{0}5]\ ^{0}x\ ^{0}[^{0}+\ x\ ^{0}1]]$$
$$^{2}[^{0}Sub\ [^{0}Tr\ ^{0}5]\ ^{0}y\ ^{0}[^{0}+\ y\ ^{0}1]]$$

would not be α-equivalent. But they should be, because from the procedural point of view it is irrelevant which variables are used as formal parameters for which the respective argument is substituted. Thus we define the following:

Definition 8 (***α-equivalence***) Let *C*, *D* be constructions. Then *C*, *D* are *α-equivalent*, if either *C*, *D* differ at most by using different λ-bound variables (of the same type), or their β-value equivalent forms differ at most by using different λ-bound variables (of the same type).

Claim 2. α-equivalent constructions are strictly equivalent being either *v*-improper or *v*-constructing one and the same entity.

Proof. Due to Claim 1 it suffices to prove that Closures of the form $[\lambda x_1 \ldots x_n\ Y(x_1,\ldots,x_n)]$, $[\lambda y_1 \ldots y_n\ Y(y_1,\ldots,y_n)]$, where $Y(x_1,\ldots,x_n)$ differs from $Y(y_1,\ldots,y_n)$ only by collisionless substitution of variables $x_1,\ldots,x_n$ for $y_1,\ldots,y_n$, respectively, *v*-construct one and the same function. But this immediately follows from Def. 2, iv).

Definition 9 (***procedural isomorphism***) Let *C*, *D* be constructions. Then *C*, *D* are *procedurally isomorphic* iff either *C* and *D* are identical or there are constructions $C_1,\ldots, C_n$ ($n > 1$) such that $^{0}C = {}^{0}C_1$, $^{0}D = {}^{0}C_n$, and for each C_i, C_{i+1} ($1 \leq i < n$) it holds that C_i, C_{i+1} are either α- or β-value equivalent.

Corollary. Procedural isomorphism is an equivalence relation defined on a set of constructions such that procedurally isomorphic constructions are strictly equivalent in the sense that for any valuation *v* they either *v*-construct one and the same entity or they are *v*-improper.

Proof. It follows immediately from Claims 1 and 2.

Having defined procedural isomorphism, we can now specify the *rule of substitution in hyperintensional contexts*.

In a hyperintensional context only procedurally isomorphic constructions are mutually substitutable.

Example. Consider the analysis of the sentence "Tilman is calculating the value of the function *sin(x)*:*cos(x)* at the argument equal to the (principal) square root of 2" that comes down to

$$\lambda w \lambda t\ [^{0}Calculate_{wt}\ ^{0}Tilman\ ^{0}[\lambda x\ [^{0}:\ [^{0}Sin\ x]\ [^{0}Cos\ x]]\ [^{0}\surd\ ^{0}2]]]. \qquad (1)$$

Types. Calculate/$(o\iota*_n)_{\tau\omega}$; Tilman/ι; **:I**/$(\tau\tau\tau)$; Sin, Cos/$(\tau\tau)$; $\surd$/$(\tau\tau)$: the principal, non-negative square root.

Since the construction

$$[\lambda x\ [^0\mathbf{:}\ [^0Sin\ x]\ [^0Sin\ x]]\ [^0\surd\ ^02]]$$

is procedurally isomorphic with its β-value equivalent form

$$^2[^0Sub\ [^0Tr\ [^0\surd\ ^02]]\ ^0x\ ^0[^0\mathbf{:}\ [^0Sin\ x]\ [^0Sin\ x]]],$$

it follows that Tilman calculates this Composition as well as its β-value and α-equivalent procedurally isomorphic variants:

$$\lambda w\lambda t\ [^0Calculate_{wt}\ ^0Tilman\ ^{02}[^0Sub\ [^0Tr\ [^0\surd\ ^02]]\ ^0x\ ^0[^0\mathbf{:}\ [^0Sin\ x]\ [^0Cos\ x]]]], \quad (2)$$

$$\lambda w\lambda t\ [^0Calculate_{wt}\ ^0Tilman\ ^{02}[^0Sub\ [^0Tr\ [^0\surd\ ^02]]\ ^0y\ ^0[^0\mathbf{:}\ [^0Sin\ y]\ [^0Cos\ y]]]]. \quad (3)$$

But it does *not* follow that Tilman calculates the ratio of Sine at $\surd 2$ and Cosine at $\surd 2$:

$$\lambda w\lambda t\ [^0Calculate_{wt}\ ^0Tilman\ ^0[^0\mathbf{:}\ [^0Sin\ [^0\surd\ ^02]]\ [^0Cos\ [^0\surd\ ^02]]]].$$

Note that $[^0\mathbf{:}\ [^0Sin\ [^0\surd\ ^02]]\ [^0Cos\ [^0\surd\ ^02]]]$ is the result of β-reduction by name of the original Composition $[\lambda x\ [^0\mathbf{:}\ [^0Sin\ x]\ [^0Cos\ x]]\ [^0\surd\ ^02]]$. While in the latter the square root of 2 is calculated only once, in the reduced construction it is calculated twice. Hence the two constructions are not procedurally isomorphic.

Since *Calculate* is a relation(-in-intension) of an individual to a *construction*, and the application of *Sub* produces a construction, a question arises here. Could we omit in (2) or (3) the Trivialization and Double Execution preceding the application of the function *Sub*? In other words, are the constructions (2) and (3) equivalent with this one:

$$\lambda w\lambda t\ [^0Calculate_{wt}\ ^0Tilman\ [^0Sub\ [^0Tr\ [^0\surd\ ^02]]\ ^0x\ ^0[^0\mathbf{:}\ [^0Sin\ x]\ [^0Cos\ x]]]]\ ? \quad (4)$$

Our answer is *no*, it is not. The reason is this. In (4) Tilman is related to the *product* of

$$[^0Sub\ [^0Tr\ [^0\surd\ ^02]]\ ^0x\ ^0[^0\mathbf{:}\ [^0Sin\ x]\ [^0Cos\ x]]]$$

which is the following Composition:

$$[^0\mathbf{:}\ [^0Sin\ ^01.4142135\ldots]\ [^0Cos\ ^01.4142135\ldots]].$$

Obviously, (4) follows from (2) but not vice versa.

Yet there are other interesting issues concerning the interplay between Trivialization and Double Execution. While Trivialization raises the context up to the hyperintensional level, Double Execution decreases it back to the intensional one. Hence for any construction *C* it holds that ^{20}C is logically equivalent to *C.* The question is whether the two constructions, ^{20}C and *C*, are not procedurally isomorphic as well. In our opinion they are *not.* The former contains two additional executive steps, to wit, Double Execution and Trivialization. Though in an ordinary vernacular this slight difference would most probably not matter, in the semantics of a programming language it does matter.

Thus we are considering whether it is philosophically wise to adopt several notions of procedural isomorphism. The definition I proposed in this paper is a very strict criterion of synonymy, and it is not improbable that several degrees of hyperintensional individuation are called for, depending on which sort of discourse happens to be analyzed. Thus we admit that slightly different definitions of procedural isomorphism are still thinkable. What appears to be synonymous in an ordinary vernacular might not be synonymous in a professional language like the language of, for instance, logic, mathematics or physics.

4 Conclusion

I demonstrated how to validly apply Leibniz's Law of the substitution of identicals in hyperintensional contexts. Since in such a context the meaning of an expression is displayed, only synonymous expressions with the same meaning can be mutually substituted. Our criterion of synonymy is procedural isomorphism of the constructions expressed by the respective expressions. The paper offers a formally worked-out and philosophically motivated criterion of hyperintensional individuation, which is the relation of procedural isomorphism. The definition of procedural isomorphism includes a slightly more carefully stated version of α-conversion and

β-conversion by value, which amounts to a modification of Church's Alternative (A1).

Acknowledgements

This work has been supported by the internal grant agency of VSB-Technical University Ostrava, Project No. SP2014/157, "Knowledge modeling, process simulation and design".

Versions of this paper have been presented at Logica 2013, Czech Republic, and CICLing 2014, Nepal.

References

1. **Anderson, C.A. (1998).** Alonzo Church's contributions to philosophy and intensional logic. *The Bulletin of Symbolic Logic,* 4(2), 129–171.
2. **Chang, S. & Felleisen, M. (2012).** The call-by-need lambda calculus, revisited. *Programming Languages and Systems*. *Lecture Notes in Computer Science*, 7211, 128–147.
3. **Carnap, R. (1947).** *Meaning and necessity*. Chicago: Chicago University Press.
4. **Church, A. (1993).** A revised formulation of the logic of sense and denotation. Alternative (1). *Noûs*, 27(2), 141–157.
5. **Duží, M. (2010).** The paradox of inference and the non-triviality of analytic information. *Journal of Philosophical Logic*, 39(5), 473–510.
6. **Duží, M. (2012).** Towards an extensional calculus of hyperintensions. *Organon F*, 19, supplementary issue 1, 20–45.
7. **Duží, M. (2013).** Deduction in TIL: From simple to ramified hierarchy of types. *Organon F*, 20, supplementary issue 2, 5–36.
8. **Duží, M. & Jespersen, B. (2010).** Transparent Quantification into Hyperintensional Contexts. In M. Peliš and V. Punčochář (eds.), *The Logica Yearbook 2010* (81–98). London: College Publications.
9. **Duží, M. & Jespersen, B. (2012).** Transparent quantification into hyperpropositional contexts de re. *Logique & Analyse, 55(*220), 513–554.
10. **Duží, M. & Jespersen, B. (2013).** Procedural isomorphism, analytic information, and β-conversion by value. *Logic Journal of the IGPL*, 21(2), 291–308.
11. **Duží, M., Jespersen, B**. (to appear). Transparent quantification into hyperintensional objectual attitudes. *Synthese*, special issue on Hyperintensionality
12. **Duží, M., Jespersen, B., & Materna, P. (2010).** *Procedural Semantics for Hyperintensional Logic: Foundations and Applications of Trasnsparent Intensional Logic*. Dordrecht: New York: Springer.
13. **Ludwig, K. & Ray, G. (1998).** Semantics for opaque contexts. *Philosophical Perspectives*, 12(S12), 141–166.
14. **Materna, P. (1998).** Concepts and Objects. *Acta Philosophica Fennica*, vol. 63. Helsinki: Edidit Societas Philosophica: Distribuit Akateeminen Kirjakauppa.
15. **Moschovakis, Y.N. (1993).** Sense and denotation as algorithm and value. *Lecture Notes in Logic*, 2, 210–249.
16. **Plotkin, G.D. (1975).** Call-by-name, call-by-value, and the lambda calculus. *Theoretical Computer Science,* 1, 125–159.
17. **Salmon, N. (2010).** Lambda in sentences with designators: an ode to complex predication. *Journal of Philosophy*, 107(9), 445–468.
18. **Tichý, P. (1968).** Smysl a procedura. *Filosofický časopis,* 16, 222–232. Translated as 'Sense and procedure' in (Tichý 2004: 77-92).
19. **Tichý, P. (1969).** Intensions in terms of Turing machines. *Studia Logica,* 24(1), 7–21. Reprinted in (Tichý 2004: 93–109).
20. **Tichý, P. (2004).** *Collected Papers in Logic and Philosophy*. Prague: Filosofia; Dunedin, N.Z.: University of Otago Press.

Marie Duží received her CSc. degree (roughly equivalent to Ph.D.) in 1992 from the Czech Academy of Sciences in Logic. Since 2001 she has been an Assistant Professor of Computer Science in VSB-Technical University Ostrava, Czech Republic.

Article received on 08/01/2014 on 06/02/2014.