Equality in Lambda Calculus, Weak Universality in Category Theory and Reversible Computations

Sergey Baranov
SPIIRAS, Russian Academy of Sciences, St.Petersburg, Russia
SNBaranov@gmail.com
Sergei Soloviev*
IRIT, University of Toulouse, France
soloviev@irit.fr

1 Introduction

Mathematical models based on category theory are often used in computer science [1], but the approaches to categories in category theory and in computer science are very different. Researchers in "mainstream" category theory usually seek higher levels of abstraction and universality, while in computer science categories are used (if at all) as a source of more or less concrete models and constructions with the main objective to provide a viable proof-of-concept for particular architectural solutions with respect to their consistency, completeness and other important properties. Even in case of a highly general and abstract categorical notion of monad, mostly concrete aspects of this notion are exploited, for example, in programming (the Haskell language), compilation (MLj), and development of (implementable) categorical abstract machines (cf. [2], [4]).

One point where this difference of approaches may be seen very clearly, is the role played by computational aspects of equality. In category theory the equality of objects and morphisms is included in definitions of categories, but seldom any attention is paid to the computational aspects of this equality. If computational aspects of equality are taken into account, it is done in connection with general questions of decidability/undecidability. In computer science their conceptual (and practical) importance is much greater.

There is, for example, an opposition between so called intensional and extensional equalities. Two functions are extensionally equal if they always produce equal outputs for equal inputs. Functions in computer science are usually represented by some syntactical expressions (programs). In difference from extensional equality, intensional equality is defined w.r.t. a certain system of conversions of these expressions (syntactic transformations corresponding to certain

*This author was partly supported by the Climt project, ANR-11-BS02-016.

basic identities). For example, $(\lambda x : A.M)N =_\beta [N/x]M$ (β-reduction and corresponding equality in λ-calculus).

Intensional equality plays in computer science much greater role than in "mainstream" mathematics based on classical logic. One would be not mistaken to say that verification of equality via syntactic transformations of programs and syntactic expressions is the principal method used for equality check in computer science. An obvious reason is that in general verification of extensional equality on a (potentially) infinite domain is indecidable. Even on a finite domain the complexity of this check may be overwhelming. Another source of difficulties concerning extensional equality is that the "static" equality check is only a special case. Dynamic equality check is in practice more common, e.g. the elements of datatypes may be generated dynamically by some process. In categories used in computer science the datatypes often play the role of objects. The importance of equality on objects and morphisms for various categorical constructions needs no further argument.

One of our main observations is that many standard universal constructions of category theory become weakly universal when considered in categories with intensional equality used in computer science. We explore this fact in context of on-going research in computer science concerning, for example, the notion of canonical elements of inductive types [20] (one may speak also about "concrete" and "abstract" elements), the theme of reversible computation [23] (where we introduce the notion of conditional reversibility), etc.

As a main "illustration tool" we define several categorical structures on simply typed lambda-calculus with inductive types and recursion operators T_{ind} (cf. [8, 9] where a slightly larger system is used). All have types as objects but differ by the notion of equality. There is a well studied structure of free cartesian closed category on simply typed lambda-calculus with surjective pairing and terminal object without inductive types [16]. We considered the calculus with inductive types and recursion because it strengthens and makes more explicit computational aspects, and we think that categorical structures on T_{ind} are of interest in themselves when computational aspects of category theory are studied.

The simple opposition of the approaches of the "mainstream" category theory and computer science does not, of course, give a complete picture of contemporary research in the domain. This is why, to complete this introduction, we have to outline the place of this paper with respect to recent research in categorical logic and type theory.

One of the first works where the relationship of extensional and intensional equality in type theory (including important categorical models) has been studied in depth was the habilitation thesis of Thomas Streicher [29]. The work of Streicher contains many profound results, but its main motivation lies in semantics of type theory: "In this thesis we will give semantic proofs of inderivability for most of these propositions which are derivable in extensional type theory but have resisted any attempt to derive them formally in ICST" (Intuitionistic Constructive Set Theory), [29], p.5.

The book by Bart Jacobs [13] on categorical logic and type theory also is

mostly devoted to categorical semantics: "The emphasis here lies on categorical semantics." [13], p.7.

The approach of this paper differs in that our interest lies in categorical structures defined on various systems of logic and type theory considered as a tool to study the phenomena that are more or less external to logic and type theory as such. For example, in his previous work the second author applied categorical structures defined on the systems of propositional logic to study coherence in categories [26], isomorphism of types [25, 7, 27] etc. In this paper we are interested in conditionally reversible computations as a possible application.

From certain point of view, our approach may have some affinity with the approach of "Homotopic Type Theory", or HOTT [12]. This subject is "hot" nowadays (slight pun intended). More seriously, we think that one of the motivations for development of this theory (independent of philosophical arguments for the so called Univalence Axiom [1]) lies in the fact that work with proof assistants based on intensional type theory contains many unpleasant surpises for the naive user, as a consequence of the difference between intensional and extensional equalities (cf. [29]).

To our opinion, the Univalence Axiom may force the collapse of many relevant mathematical structures (when isomorphic objects must be distinguished). It may be too strong, but it is not contradictory since there exist interesting models [12]. Due to its power, the efficient work with intensional equality in remaining structures may become possible, and the univalence foundations program will probably produce new and efficent tools for proof assistants.

2 Universal and Weakly Universal Constructions

In this section we kept, for the history's sake, the notation used by S. Mac Lane.

Mac Lane [19], p.55, defines the notion of a universal arrow as follows.

Definition 2.1 *If $S : D \to C$ is a functor and c is an object of C, then a universal arrow from c to S is a pair $\langle r, u\rangle$ consisting of an object r of D and an arrow $u : c \to Sr$ of C, such that to every pair $\langle d, f\rangle$ with d an object of D and $f : c \to Sd$ an arrow of C, there is a unique arrow $f' : r \to d$ of D with $Sf' \circ u = f$. In other words, every arrow f to S factors uniquely through the universal arrow u, as in the commutative diagram*

$$\begin{array}{ccc} c & \xrightarrow{u} & Sr \\ \downarrow & \| & \downarrow{\scriptstyle Sf'} \\ c & \xrightarrow{f} & Sd \end{array} \qquad \begin{array}{c} r \\ \downarrow{\scriptstyle f'} \\ d \end{array}$$

[1] The formulation of Univalence Axiom is not related to the theme of this paper, the only important aspect being that it may imply identification of many objects, functions etc. that remain different if it is not accepted.

Equivalently (Mac Lane continues), $u : c \to Sr$ is universal from c to S when the pair $\langle r, u\rangle$ is an initial object in the comma category $(c \downarrow S)$... As with any initial object, it follows that $\langle r, u\rangle$ is unique up to *isomorphism* in $(c \downarrow S)$; in particular, the object r of D is unique up to isomorphism in D.

The dual concept of universal arrows from a functor $S : D \to C$ to an object $c \in C$ can be defined as well. It is used, for example, to define a product in C ([19], p.58).

Let us elaborate this in slightly more details. Recall that *comma category* $(T \downarrow S)$ of two functors $T : E \to C$ and $S : D \to C$ is the category whose objects are triples $\langle e, d, f\rangle$ with $d \in Ob(D), e \in Ob(E)$ and $f : Te \to Sd \in Mor(C)$, and whose morphisms $\langle e, d, f\rangle \to \langle e', d', f'\rangle$ are pairs $\langle k, h\rangle$ of arrows, $k : e \to e' \in Mor(E)$, $h : d \to d' \in Mor(E)$ such that the diagram

$$\begin{array}{ccc} (*)\ Te) & \xrightarrow{Tk} & Te \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle f'} \\ Sd & \xrightarrow{Sh} & Sd' \end{array}$$

is commutative. The composite $\langle k', h'\rangle \circ \langle k, h\rangle$ is $\langle k' \circ k, h' \circ h\rangle$ when defined [19], p.46. All the cases considered above may be seen as the cases of this definition with a particular choice of functors. For example (as Mac Lane notices), in case of $(c \downarrow S)$ one may take the constant functor with value c as T.

Notice that the equality of objects in $(T \downarrow S)$ is the equality of triples:

- $\langle e, d, f\rangle = \langle e', d', f'\rangle$ iff $e = e'$ in E, $d = d'$ in D, and $f = f'$ in C.

The equality of morphisms comes from D and E: $\langle k, h\rangle = \langle k', h'\rangle$ iff $k = k'$ in E and $h = h'$ in D.

Remark 2.2 *Still, other equality relations may be of use. Below we shall consider, for example:*

$$\langle k, h\rangle =_w \langle k', h'\rangle : \langle e, d, f\rangle \to \langle e', d', f'\rangle \ \ iff \ \ Sh \circ f = Sh' \circ f'.$$

(Because of the commutativity of the square above, this is equivalent also to $f' \circ Tk = f' \circ Tk'$*.)*

Obviously, if we take the relation $=_w$ *instead of* $=$ *we obtain a factor category of* $(T \downarrow S)$ *that we will denote by* $(T \downarrow S)^*$*.*

Initiality of $\langle r, u\rangle$ above means that for any other object $\langle r', u'\rangle$ there exists a unique arrow $f : r \to r'$ that makes (*) commutative. If we have another realization $\langle r', u'\rangle$ of the universal arrow, then there exist unique $f : r \to r'$ and $f' : r' \to r$ that must be mutually inverse isomorphisms.

In spite of its triviality, let us recall the proof of this fact, since we will need in the end of this section to show exactly what the difference is in case of weak universality. First, let us take $\langle r, u\rangle$ itself as $\langle r', u'\rangle$. The identity morphism $1_r : r \to r$ may be taken as f' in the definition, and because of

unicity it is the only f' possible. Now, if we take a different initial pair $\langle r', u' \rangle$ then by definition we will have certain $f' : r \to r'$ and $f'' : r' \to r$, such that $u = Sf'' \circ (Sf' \circ u) = S(f'' \circ f') \circ u$. By unicity $f'' \circ f' = 1_r$. In a similar way, we derive that $f' \circ f'' = 1_{r'}$ and hence f' and f'' are mutually inverse isomorphisms in D.

The definition of a *weak universal arrow* ([19], p.235) differs from the definition of a universal arrow only in that f' in the diagram is *not* required to be unique. As Mac Lane remarks, it is possible to modify all the various types of universals, defining weak products, weak limits, weak coproducts (requiring just existence rather than uniqueness in each case). There is no more unicity up to isomorphism, but it does not mean that instead of isomorphisms we will have arbitrary arrows. Some *conditional* reversibility will be preserved.

In the proposition below we use the same notation as in the definition 2.1 above.

Proposition 2.3 *Let the pair $\langle r, u \rangle$ be a weak universal arrow. Then r is unique up to isomorphism in the factor category $(c \downarrow S)^*$.*

*Proof.*Without unicity condition, we still have the equalities $u = S(f'' \circ f') \circ u$ and $u' = S(f' \circ f'') \circ u'$, and they correspond exactly to the definition of isomorphism in the category $(c \downarrow S)^*$.

Remark 2.4 *The property that defines an isomorphism in $(c \downarrow S)^*$ may be seen as* conditional reversibility *(in this case, the reversibility that has the composition with u as a precondition, and "modulo" application of S).*

Of course, similar proposition will hold also for dual case.

3 The system T_{ind}

The system of λ-calculus considered below is a subsystem of the simply typed λ-calculus with inductive types, considered in detail in [8, 9, 7, 27]. It is more restricted: we excluded from the syntax the "canonical" terminal object and pairing. In [8, 9, 7, 27] the relationship of this "canonical" data with singletons and pairing defined using inductive type construction is studied in presence of additional reductions. Here we want to use it only to illustrate the general principles discussed above, and these extra data are not included.

The system considered in [8, 9, 7, 27] was itself obtained from the system UTT of Luo [18] by a series of simplifications. UTT is a dependent type theory closely related to Martin-Löf type theory and Calculus of Constructions. Our system was obtained by a) retaining only non-dependent types, b) exclusion of kinds, in particular the kind $Type$, type universes, unpredicative type $Prop$ and all logical part of UTT. All machinery concering inductive types that *was* retained is well known. It is a particular case of more general definitions for dependent types that can be found in the book of Z. Luo [18]. This is why below we do not give, for example, a self-contained definition of recursion operators over inductive types in T_{ind}.

Definition 3.1 *Types are either atomic types or obtained by application of type constructors.*

Atomic types are elements of a finite or infinite set $\mathcal{S} = \{\alpha, \beta, \ldots\}$ *of* type variables.

Type constructors are:

- $\rightarrow$ *for functional types, which constructs* $A \rightarrow B$ *for any types* A *and* B
- Ind*, defined as follows: let* $\mathcal{C}$ *be an infinite set of* introduction operators *(constructors of* elements *of inductive types), with* $\mathcal{C} \cap \mathcal{S} = \emptyset$*. an inductive type with* n *constructors* $c_1, \ldots, c_n \in \mathcal{C}$*, each of them having the arity* k_i *(with* $1 \leq i \leq n$*), has the form:*

$$Ind(\alpha)\{c_1 : A_1^1 \rightarrow \ldots \rightarrow A_1^{k_1} \rightarrow \alpha \mid \ldots \mid c_n : A_n^1 \rightarrow \ldots \rightarrow A_n^{k_n} \rightarrow \alpha\},$$

Here, every $A \equiv A_i^1 \rightarrow \ldots \rightarrow A_i^{k_i} \rightarrow \alpha$ *is an inductive schema, i.e.,* A_i^j *is:*

  - *either a type not containing* α*; (we call this* A_i^j *a* non-recursive operator*);*
  - *or a type of the form* $A_i^j \equiv C_1 \rightarrow \ldots \rightarrow C_m \rightarrow \alpha$*, where* α *does not appear in any* $C_{\ell \in 1..m}$ *(such* A_i^j *are called* strictly positive operators*).*

Here $\rightarrow$ *associates to the right, i.e.,* $C_1 \rightarrow C_2 \rightarrow ...\alpha$ *means* $(C_1 \rightarrow (C_2 \rightarrow ...\alpha)...)$*;* $Ind(\alpha)$ binds *the variable* α*.*

Example 3.2 *(The types* $Bool$*,* Nat*, functional space, and* T_ω*, the type of* ω*-trees, as inductive types.)*

$Bool =_{def} Ind(\alpha)\{T : \alpha \mid F : \alpha\}$
$Nat =_{def} Ind(\alpha)\{0 : \alpha \mid succ : \alpha \rightarrow \alpha\}$
$[A, B] =_{def} Ind(\alpha)\{fun : (A \rightarrow B) \rightarrow \alpha\}$
$T_\omega = Ind(\alpha)\{0_\omega : \alpha \mid succ_\omega : \alpha \rightarrow \alpha \mid lim_\omega : (Nat \rightarrow \alpha) \rightarrow \alpha\}$.

Definition 3.3 *Let* $\mathcal{V}$ *be an infinite set of variables* $\mathcal{V}$ *(with* $\mathcal{V} \cap \mathcal{S} = \mathcal{V} \cap \mathcal{C} = \emptyset$*). The set of* λ-terms *is generated by the following grammar rules:*

$$M \ ::= \ c \mid Rec^{B \rightarrow D} \mid x \mid (\lambda x : B \cdot M) \mid (M \ M)$$

where $x \in \mathcal{V}$*,* $c \in \mathcal{C}$*,* B *and* D *are arbitraty types, and* $Rec^{B \rightarrow D}$ *denotes the recursion operator from* B *to* D *(for details, see [8, 9], [18]).*

We write $M_0 M_1 ... M_n$ instead of $(...(M_0 M_1)...M_n)$ to reduce the number of parentheses (associativity to the left). All terms and types are considered up to α-conversion, *i.e.*, renaming of bound variables. Context Γ is a set of term variables with types $x_1 : A_1, ..., x_n : A_n$ ($x_1, ..., x_n$ should be distinct). Γ, Δ denotes union of the contexts Γ, Δ (we assume that Γ, Δ have no common term variables).

Definition 3.4 *There are the following typing axioms and rules for the terms defined above (A, B, D denote arbitrary types, Γ is an arbitrary context).*

Axioms:

- $\Gamma, x : A \vdash x : A$,;
- *For each inductive type* $C = Ind(\alpha)\{c_1 : A_1 \,|\, \ldots \,|\, c_n : A_n\}$ *and* $1 \leq i \leq n$

$$\Gamma \vdash c_i : A_i[C/\alpha]$$

(e.g., if $C = Nat$*, then* $\Gamma \vdash 0 : Nat$ *and* $\Gamma \vdash succ : Nat \to Nat$*);*

- *For* C *as above and any type* D *the axiom* [2]*:*

$$\Gamma \vdash Rec^{C \to D} : \Upsilon_C(A_1, D) \to ... \to \Upsilon_C(A_n, D) \to C \to D.$$

Typing rules.

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash (\lambda x : A \cdot M) : A \to B}(\lambda)$$

$$\frac{\Gamma \vdash M : A \to B \;\; \Gamma \vdash N : A}{\Gamma \vdash (M \;\; N) : B}(app)$$

The constant $Rec^{C \to D}$ is called the *recursor* from C to D. Notice that applying it (using the rule *app*) to the terms $M_1 : \Upsilon_C(A_1, D), ..., M_n : \Upsilon_C(A_n, D)$ we define the function $Rec^{C \to D} M_1 ... M_n : C \to D$. The following derived rule is often included:

$$\frac{\Gamma \vdash M_i : \Upsilon_C(A_i, D) \;\; (1 \leq i \leq n)}{\Gamma \vdash (Rec^{C \to D} \;\; M_1 \;\ldots\; M_n) : C \to D}(elim)$$

Normalization and intensional equality. The terms of the system T_{ind} are considered up to equality generated by conversion relation. The α-conversion (renaming of bound variables) was already mentioned. Other conversions are:[3] (i) β-conversion $(\lambda x : A.M)N = [N/x]M$; (ii) η-conversion $\lambda x : A.(Mx) = M$ (where x must not be free in M); (iii) and ι-conversion for recursion. The ι-conversion corresponds to one step in recursive computation. For example, in case of $Rec^{Nat \to Nat}$ it is

- $(Rec^{Nat \to Nat} ag)(0) \to_\iota a$,
- $(Rec^{Nat \to Nat} ag)(succ\, x) \to_\iota gx((Rec^{Nat \to Nat} ag)x)$.

[2] $\Upsilon_C(A, D)$ are certain auxilliary types used to define recursion from C to D. They correspond to the types of functions that appear in standard recursive equations over C. For example, if $C = Nat$, $A_1 = Nat$ (the type of constant 0), $A_2 = Nat \to Nat$ (the type of successor S) in the definition of Nat, then $\Upsilon_{Nat}(A_1, D) = D, \Upsilon_{Nat}(A_2, D) = Nat \to D \to D$. In more general dependent type case a detailed description of these auxilliary types may be found in [18], p.178.

[3] We omit the contexts and types of terms.

T is confluent and strongly normalizing with respect to $\beta\eta\iota$-reductions (directed conversions), i.e. every reduction sequence is finite and ends by normal form which is unique up to α-conversion. Detailed description and normalization theorems for T_{ind} can be found in [8, 9]. Thus, the equivalence relation on terms based on conversion (often called $\alpha\beta\eta\iota$-equality) is decidable.

Closed terms and canonical elements.

As usual, closed terms are terms that have no free variables. In T_{ind} they include such terms as $succ(succ\,0) : Nat$, $\lambda x : Bool.x : Bool \to Bool$ etc. The terms that do not include variables at all, like $succ(succ\,0)$ are called constant terms.

Lemma 3.5 *Let C be an inductive type, and $\vdash M : C$ in T_{ind} some closed term. If M is normal, then M has the form $c_i M'$ where c_i is one of the constructors (introduction operators) of C.*

Proof. We use standard properties of normal forms in typed lambda-calculus (cf [8, 9]) and proceed by induction on the length of M. Since M has type C and C, being unductive type, does not contain $\to$, M cannot begin with λ. In this case M is necessarily an application of the form $M_0 M_1 ... M_n$ where all terms $M_0, ..., M_n$ are normal closed terms, and M_0 is not an application. M_0 cannot be a variable (it would be free). It cannot begin with λ (the term M would be not normal). Two remaining possibilities are that M_0 is an inductive type constructor c_i (then we are done) or that M_0 is a recursor. If it is a recursor, then it is a recursor from some inductive type C' to C, and M_1 should be of type C'. M_1 is closed normal term, and by induction it begins by some constructor of C'. In this case ι-reduction is applicable to M and it is not normal.

Some inductive types, like ω-trees, have constructors that may take functions as arguments, and this makes the precise (and useful) definition of canonical elements difficult. When functional arguments are excluded (such types are sometimes called 0-recursive [8, 9]), the lemma above permits to identify the canonical elements with closed terms and to show that they are the same as constant terms obtained by application of type constructors.

Definition 3.6 *Let us call an inductive type 0-recursive, if it is defined according to the defintion 3.1 with additional restriction applied recursively:*

- *in each inductive schema $A \equiv A_i^1 \to \ldots \to A_i^{k_i} \to \alpha$, A_i^j is either a 0-recursive type (if it does not contain α freely) or α (without any premise C_m).*

Theorem 3.7 *Let an inductive type C be 0-recursive, and $\vdash M : C$ in T_{ind} be some closed term. If M is normal, then M is constant term built by application using only constructors of 0-recursive inductive types.*

Proof by induction on the length of M (using standard properties of normal terms). By lemma 3.5 M has the form $c_i M_1 ... M_n$ where $M_1, ..., M_n$ are closed terms whose number and types are defined by the inductive schema corresponding to c_i.

Let this schema be $A \equiv A_i^1 \to \ldots \to A_i^{k_i} \to \alpha$.

The type of c_i is $[C/\alpha]A$. Those A_j^i in A that do not contain α are not changed (they are 0-recursive) and those that are α are replaced by C. Thus, the terms $M_1, ..., M_n$ are all closed terms of 0-recursive types, and inductive hypothesis can be applied.

Definition 3.8 *Let C be a 0-recursive inductive type. We shall call its canonical elements the terms $\vdash M : C$ built by application using only constructors of 0-recursive inductive types (including constructors without arguments, like $0 : Nat$).*

Example 3.9 *An inductive type usually called product is defined as*

$$A \times B =_{def} Ind(\alpha)\{pair : A \to (B \to \alpha)\}$$

(cf. [8, 9], [18]). If we take $Nat \times Nat$, then all normal closed terms of this type will be of the form $pair(succ...(succ\,0)...)(succ...(succ\,0)...)$ (possibly not the same number of succ).

Remark 3.10 *If we take some type that is not 0-recursive, say, T_ω, there are closed terms of the form*

$$lim((Rec^{Nat \to T_\omega}(succ_\omega 0_\omega))(\lambda x : Nat.\lambda y : T_\omega.y)) : T_\omega$$

that contain not only constructors (introduction operators). Another example is function space $[A, B] = Ind(\alpha)\{fun : (A \to B) \to \alpha\}$. The type $A \to B$ at the right is not 0-recursive. Now, if we take $A = B$ there are closed terms like $fun(\lambda x : A.x)$ that are not canonical elements in the above-mentioned sense.

Categorical structures on T_{ind}. For all variants of categorical structure we shall consider, the objects of the category T_{ind} are types, described above. Equality of objects is syntactic identity[4]. The morphisms from A to B are closed terms (i.e., terms that do not contain free term variables) of type $A \to B$, i.e. $\vdash f : A \to B$ should be derivable in T_{ind}.

The categorical structures will differ only by equality (equivalence relation) on morphisms. Speaking about morphisms, we shall usually omit $\vdash$. The composition of $f : A \to B$ and $f' : B \to C$ is defined as the (equivalence class of) the term $\lambda x : A.(f'(fx))$, $f' \circ f =_{def} \lambda x : A.(f'(fx))$. The identity is defined as the (equivalence class of) $id_A =_{def} \lambda x : A.x : A \to A$.

Below we shall consider two main equivalence relations on morphisms. If closed terms $f : A \to B$ are considered up to $\alpha\beta\eta\iota$-equality, we shall speak about T_{ind} with intensional equality.

Another equivalence relation, that we shall call extensional equality, is defined by the following condition. Let $f, g : A \to B$. We shall call f and g extensionally equal iff $ft =_{\alpha\beta\eta\iota} gt$ for every closed term $t : A$ in T_{ind}.

The axioms of category are trivially satisfied for T_{ind} with both variants of equality.

[4]Technically it is more convenient to compare different kinds of equality only on morphisms.

Remark 3.11 *The intensional equality on T_{ind} is decidable: indeed, to verify $f =_{\alpha\beta\eta\iota} g$ the terms are reduced to normal form (T_{ind} is strongly normalizing [8, 9]) and then α-convertibility is trivially verified. To the contrary, extensional equality is not: the use of recursors permits to represent, e.g., all primitive recursive functions $f : Nat \to Nat$. Their equality on canonical elements of Nat coincides with ordinary equality of functions, and for primitive recursive functions it is not decidable [14].*

Remark 3.12 *In general, an extensional equality on terms of functional types $f, g : A \to B$ is defined by some condition of the form:*

- *For all t of type A satisfying certain condition $ft =_{\alpha\beta\eta\iota} gt$.*

The equality $f =_{\alpha\beta\eta\iota} g$ implies $ft =_{\alpha\beta\eta\iota} gt$. The extensional equality we are considering is sometimes called extensional equality in closed term model. Thus the extensional equality always contains the intensional equality based on $\alpha\beta\eta\iota$.

Remark 3.13 *If the type A in $f : A \to B$ is 0-recursive, then, according to theorem 3.7, all closed terms $t : A$ represent canonical elements, and extensional equality we introduced corresponds to ordinary set-theoretical equality of functions on the sets of canonical elements.*

By T^0_{ind} we shall denote the full subcategory of T_{ind} whose objects are 0-recursive inductive types.

4 Case Studies

4.1 Intensional and Extensional Equality in T_{ind}

Let us consider two terms of T_{ind}:

- $f_1 = succ : Nat \to Nat$ and
- $f_2 = Rec^{Nat\to Nat}(\lambda y : Nat.succ)(succ\, 0) : Nat \to Nat$.

Each term is a morphism of T_{ind}. Each term represents also a function on the terms of type Nat defined by $f_i(t) =_{def} f_i t$.

Canonical elements of Nat are represented by the terms $0, ..., succ(...(succ\, 0))$, and on any canonical element $n : Nat$ both f_1 and f_2 have the value Sn. At the same time f_1 and f_2 are not intensionally equal: both are already in normal form and these normal forms are different.

We can define also a one side inverse to f_1 with respect to intensional equality, given by $f'_1 =_{def} Rec^{Nat\to Nat}(\lambda x : Nat.\lambda y : Nat.x)0$ (the value of f'_1 on $succ\, n$ will be n, the value on 0 will be 0). For the composition with f_1,

$$\lambda x : Nat.((Rec^{Nat\to Nat}(\lambda x : Nat.\lambda y : Nat.x)0)(succ\, x)) \to_\iota$$

$$\lambda x : Nat.(\lambda x : Nat.\lambda y : Nat.x)x(succ\, x)) =_\beta \lambda x : Nat.x =_{def} id_{Nat}.$$

If we compose f_1' with f_2, applying the composition to canonical elements we have

$$f_1'(f_2 0) =_{\beta\eta\iota} 0, \;\; f_1'(f_2(n)) =_{\beta\eta\iota} f'(succ\, n) =_{\beta\eta\iota} n,$$

but the composition itself (Rec means $Rec^{Nat\to Nat}$)

$$\lambda x : Nat.((Rec(\lambda x : Nat.\lambda y : Nat.x)0)(Rec(\lambda y : Nat.succ)(succ\, 0)x))$$

is normal (i.e.,does not admit any reduction) and so it is not equal to id_{Nat}.

- It can be shown that f_2 does not have left inverse w.r.t. intensional equality at all.

  Outline of a proof. Observe that the term $f_2 x = (Rec^{Nat\to Nat}(\lambda y : Nat.succ)(succ\, 0))x$ is normal and has type Nat. Consider any composition $f_2' \circ f_2 = \lambda x : Nat.(f_2'(f_2 x))$ where f_2' also is normal. If it would reduce to id, at least one reduction would be possible, and, taking into account the form of the terms, this reduction could be only β-reduction, in particular f_2' must be of the form $\lambda y : Nat.f_2''$. Simple case analysis shows that the term $[f_2 x/y]f_2''$ will not allow any further reductions (will be normal). This term contains Rec and so cannot represent id.

- Of course, both f_1 and f_2 have left inverses w.r.t. extensional equality. This behaviour can be seen as a case of conditional reversibility: f_2 is reversible at the left if the arguments are canonical elements.

- More categorical view at this conditional reversibility would be that some functor from the category T_{ind} to the category of sets such that the types become sets of their canonical elements is applied first (and equality of morphisms in this "target" category is the extensional equality of functions represented by λ-terms).

4.2 Weak Terminal Objects in T_{ind}

An inductive type with one element may be defined in T_{ind} as $Ind(\alpha)\{c : \alpha\}$. Allowing some abuse of notation, we shall denote this type by $\{c\}$. The constant c may be considered as (the name of) its unique element. The related typing axiom is $\Gamma \vdash c : \{c\}$. There are other such types, obtained by changing c.

The definition of a terminal object $\top \in T_{ind}$ as a universal construction (in the strong sense) is equivalent to the condition that for every object $A \in T_{ind}$ there exists the unique $f : A \to \top$. For a weak terminal object only the existence is required. If we take any of the types $\{c\}$, for any A there is $\lambda x : A.c : A \to \{c\}$, but other non-equivalent closed terms of the same type may exist (for example, defined using recursors).

The recursor $Rec^{\{c\}\to A}$ has the type $A \to \{c\} \to A$, i.e., the functions from $\{c\}$ to A are defined by application of $Rec^{\{c\}\to A}$ to $a : A$, an obvious interpretation is that they are defined by their value on the unique element $c : \{c\}$. Still, with respect to intensional equality $Rec^{\{c\}\to\{c'\}}c' : \{c\} \to \{c'\}$ is not equal to $\lambda x : \{c\}.c' : \{c\} \to \{c'\}$. Moreover, with respect to this equality

the morphisms $\lambda x : \{c\}.c' : \{c\} \to \{c'\}$, $\lambda x : \{c'\}.c : \{c'\} \to \{c\}$, $Rec^{\{c\}\to\{c'\}}c' : \{c\} \to \{c'\}$, $Rec^{\{c'\}\to\{c\}}c : \{c'\} \to \{c\}$ are not mutually inverse isomorphisms. For example, the composition of first two gives

$$\lambda y : \{c\}.(\lambda x : \{c\}.c'((\lambda x : \{c'\}.c)y)) =_{\beta\eta\iota} \lambda y : \{c\}.c \neq \lambda y : \{c\}.y =_{def} id_{\{c\}}.$$

The composition of second two is a normal term and so also is not equal to $id_{\{c\}}$. It is possible to show that with respect to intensional equality they are not isomorphisms at all.

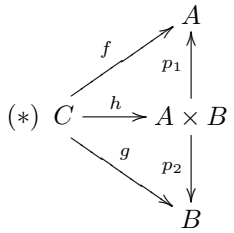
The same remark as in the end of the previous subsection can be added concerning conditional reversibility.

4.3 Product as a Weakly Universal Construction

Let us take as an example the notion of product $A \times B$ of two objects A, B of a category K. It can be defined using the notion of universal arrow from diagonal functor $\Delta : K \to K \times K$ (in functor category) to the functor $F : \{1, 2\} \to K$ from discrete category $\{1, 2\}$ to K (with $F(1) = A, F(2) = B$). The details can be found in [19], p.69. We shall skip them (only the fact that this may be seen as a particular case of the notion of universal arrow is important) and pass directly to more common equivalent definition using projections.

The object $A \times B \in Ob(K)$ is called product of two objects $A, B \in Ob(K)$ iff

- there exist the unique arrows $p_1 : A \times B \to A$ and $p_2 : A \times B \to B$ (called projections) such that
- for every object $C \in Ob(K)$ and two arrows $f : C \to A, g : C \to B$ there exists a unique arrow $h : C \to A \times B$ that makes the following diagram commute:



The arrow h is denoted $\langle f, g\rangle$. It is usually called product (or pair) of f, g, and f, g are called its components. Universality (in the strong sense) of this construction is reflected by the condition of unicity of projections and h. One of the consequences is that $A \times B$ is unique up to isomorphism.

Let us consider now, how all this will work in T_{ind}. Given two types A, B, an inductive type usually called product of A, B (cf. [18]) is defined as follows:

$$A \times B =_{def} Ind(\alpha)(pair : A \to (B \to \alpha)).$$

Its canonical elements are terms of the form $(pair\, s)t$ where $s : A$ and $t : B$. There may be other elements that do not have the constructor *pair* at their head. For example, if we admit open terms as elements of objects, the variable $x : A \times B$ is a non-canonical element.

It turns out that in T_{ind} with intensional equality $A \times B$ can not be considered as product in the sense of strong universality.

As any inductive type, $A \times B$ in T_{ind} comes equipped with recursion operators. The recursion operator from $A \times B$ to D is a constant $R : (A \to (B \to D)) \to (A \times B \to D)$. The corresponding ι-conversion is $(Rf)((pair\, t_1)t_2) = (ft_1)t_2$ with $f : A \to (B \to D)$, $t_1 : A, t_2 : B$. Notice that if $s : A \times B$ is not of the form $(pair\, t_1)t_2$ then the conversion is not applicable. Let us denote R_1 and R_2 the recursion operators from $A \times B$ to A and B respectively. Projections are defined now as $p_1 = R_1(\lambda x : A.\lambda y : B.x) : A \times B \to A$ and $p_2 = R_2(\lambda x : A.\lambda y : B.y) : A \times B \to B$.

Given two terms $f : C \to A$ and $g : C \to B$, h of the diagram $(*)$ may be defined as $h = \langle f, g\rangle =_{def} \lambda z : C.(pair\, f(z))g(z)$. The diagram will be commutative, but what about the unicity of h?

Let $C = A \times B$ and $h : C \to A \times B$ be $id_{A\times B} = \lambda x : A \times B.x$. Let $f = p_1 \circ h = p_1, g = p_2 \circ h = p_2$. The diagram $(*)$ will be commutative. Let us take $h' = \langle p_1, p_2\rangle$. The diagram will be commutative, but h and h' are not equal w.r.t. the intensional equality in T_{ind}.

The product in T_{ind} with intensional equality is only weakly universal. It is possible to define another product as $A \times' B =_{def} Ind(\alpha)(pair' : A \to (B \to \alpha))$ (the only modification is the name of the constructor). The "products" $A \times B$ and $A \times' B$ will *not* be isomorphic in T_{ind} with intensional equality.

More precisely, let $\langle f, g\rangle' =_{def} \lambda z : C.((pair' f(z))g(z))$. The "candidates" to the role of isomorphisms are obvious:

$$\theta = \langle p_1, p_2\rangle' : A \times B \to A \times' B,\ \ \theta' = \langle p'_1, p'_2\rangle : A \times' B \to A \times B,$$

but they are not mutually inverse w.r.t. intensional equality. Using the technique similar to that we used in 4.1, it is possible to show that there is no isomorphism at all.

Remark 4.1 *In fact, it is possible to consider the extensions of T_{ind} that include explicitly some product operator, and even add well-behaving reductions like $\langle p_1 \circ h, p_2 \circ h\rangle = h$ but this will not completely solve the problem, as the absence of unicity of product shows. To establish "equivalence" of different product operators, it will be necessary to introduce more reductions each time when one more product operator is added (cf. [8, 9]).*

Remark 4.2 *(Conditional inversibility.) Consider the following diagram:*

$$C \xrightarrow{h} A \times B \underset{\theta'}{\overset{\theta}{\rightleftarrows}} A \times' B\ .$$

The morphisms θ and θ' are not mutually inverse w.r.t. intensional equality, but they are mutually inverse conditionally, in the following sense. If $h = \langle f, g\rangle$

for some $f : C \to A$ and $g : C \to B$ then $(\theta' \circ \theta) \circ h = h$. (Cf. with the defintion of equality in the categories $(S \downarrow T)^*$.)

4.4 Product and Extensional Equality in T^0_{ind}.

Obviously, T^0_{ind} is closed w.r.t. product defined as inductive type. Notice that it is not the same with functional space $[A, B]$. Below we consider T^0_{ind} with extensional equality.

Theorem 4.3 *In T^0_{ind} with extensional equality the product construction described above is universal in ordinary sense.*

Proof. We need to show the unicity of h in diagram $(*)$ above. Notice that the type $A \times B$ is 0-recursive. Consider another morphism $h' : C \to A \times B$ that makes the diagram commutative. Let us take any canonical element $c : C$. Since f, g, h, h' are represented by closed terms, the terms $fc, gc, hc, h'c$ are closed as well, and theorem 3.7 can be applied. It follows immediately that $h'c \mathit{pair}\, a\, b$ for some canonical elements $a : A, b : B$. Notice that $hc =_\beta \mathit{pair}\, (fc)\, (gc)$. Application of p_1 and p_2 gives $a = fc, b = gc$ and thus h and h' are extensionnally equal.

Corollary 4.4 *T^0_{ind} with extensional equality and product $\times$ defined as above is cartesian. Product is unique up to extensional isomorphism in T^0_{ind}.*

5 Discussion, Applications and Perspectives

The simple cases studied above may easily give an impression of "toy examples". To render them their due significance, we need to discuss them in a broader context, consider possible applications and perspectives of future research.

5.1 Discussion

The calculus T_{ind} has been chosen because of relative simplicty of its description, but it has considerable computational power: the inductive types of T_{ind} together with the associated recursion operators are sufficient to define all functionals of finite type [10, 15, 30].

The definition of inductive types and recursion in T_{ind} is a direct restriction to the simply typed case of the general definition used in powerful dependent type theories (we used, similarly to [8, 9, 27] the restricted form of the definitions from Z. Luo's system UTT [18]. Luo's UTT is not very much different in this respect from Martin-Löf type theory or the Calculus of Constructions used in Coq).

In the sense of metatheory, all inductive types of T_{ind} are also definable in UTT or in proof assistant Coq, but the category that contains these types *only* is not definable internally.

We did not include in T_{ind} the types $Prop$, Prf, identity types, etc. In fact, we were not really interested in logical power of T_{ind}, but only in its computational properties.

In type theories with unductive types η-rules usually are understood in generalized sense. E.g., in [18] it is explained how in UTT a *logical η-rule* can be defined for an inductive type A defined by any finite sequence of inductive schemata $\Theta_1, ..., \Theta_n$ and logical validity of η-rules is proved (p.201)[5].

It is well known, that in the presence of η-rules and identity types the type-checking is indecidable since conversion depends on whether the types are inhabited (non-empty), cf. [13]). To our opinion, this makes the type theory with these rules and types useless as an underlying system for introduction of a categorical structure, because even the composability of morphisms will be undecidable [6].

Notice that if in the dependent type theories mentioned above the η-rules are not included at all, the situations similar to the situations considered in our examples will be easily reproduced.

Another reason why we did not include logical machinery in T_{ind} is that one of the main features of logical frameworks is that they permit to define application-oriented type theories. These theories may include some (not all) of the types that may be defined in the theory (e.g., some inductive types, some types such as $Prop$, Prf etc.), may contain, or not η-rules, and even contain some additional user-defined conversions. The question, what kind of categorical structure may exist on such a type theory (e.g. monoidal, cartesian, cartesian closed etc.) is of great interest, but there is no general theorem describing in advance all necessary properties of the underlying system, in particular with respect to extensional and intensional equality. To consider within this paper not only computational, but also logical properties of such intermediate systems would be a distraction.

To make this remark more clear, let us consider in more detail some ideas and results of [8, 9, 27] [7].

The principal idea explored in these works was that some reductions (interesting from computational point of view) may be added in such a way that strong normalization (SN) and Church-Rosser property (CR) will be preserved. The system considered there was an extended version of T_{ind} described above, including canonical surjective pairing and terminal object (below we shall call this system T_{ind} as well). One may note, that a dependent type system (Luo's UTT) extended with these reductions was considered in the thesis (in french) of another ph.d. student of S. Soloviev, Lionel Marie-Magdeleine [21] but there

[5] Another important class of rules is the class of *filling-up rules*. As Luo notices in [18], p.201, "The logical η-rules express that every object is equal to a canonical object and the filling-up rules express that the elimination operator covers all of the use of the inductive data type."

[6] The distinction has to be made between the existence of certain categorical models of a logical system that may be useful for its semantics, and introduction of an application-oriented categorical structure on the system itself that is mostly considered in this paper.

[7] [9, 7, 27] are closely connected with ph.d. thesis of David Chemouil [8] (in French). Second author was the supervisor of his thesis.

is no easily accessible publications of his work.

Among new reductions studied there was the reduction for isomorphism of "copy" between inductive types. For every inductive type A in T_{ind} and its copy A' that differs only by different choice of names of introduction operators in its definition there exist canonical closed terms $c : A \to A'$ and $c' : A' \to A$ (c and c' are defined by recursion in T_{ind} over A and A' respectively). The new reduction (called χ-reduction) was defined by rewriting rules $c'(ct) \to t$ and $c(c't') \to t'$. One may say, that χ-reduction makes copy an intensional isomorphism.

Other reductions included: η-reductions for products defined as inductive types; the reductions (similar to χ-reduction) that "make intensional" the isomorphisms between products defined as inductive types and canonical product defined by pairing; η-rules for finite types.

It was shown that T_{ind} with these reductions is SN and CR. Of course new examples that show the difference between extensional and intensional equality similar to elementary examples considered above may be constructed in the extended system. The fundamental difference between extensional and intensional equality cannot be cancelled by "local" extensions of the notion of intensional equality.

5.2 Applications

The importance of categorical models for computer science motivates also the study of behaviour of extensional and intensional equality in these models. We would like to attract attention to reversible computations as a possible domain of applications. In particular, the link between weak universality and conditional reversibility of computations may be exploited.

Reversible computations are actively studied since 1960es. Works on reversible computations link together such distant domains of science as logics, theory of algorithms, physics, thermodynamics and even biology, cf. [3, 17, 23, 31]. In practice, though, most of the computations considered as reversible are reversible only in more or less idealized models. Sometimes (but not always) the pre-conditions that make these models adequate may be clearly identified.

For example, the conditions may be purely mathematical. They may also concern the treatment of information (history of computations), the physical properties of a system (quantum state) etc.. Mathematical conditions may be concrete, e.g., expressed in terms of values of certain parameters, like non-zero determinant of a matrix, or more abstract (expressed in general terms characterizing the environment or the history of computations). Below we take into account only theoretical aspects of reversible computations related to mathematics and computer science.

From the categorical point of view, the reversible computations may be considered as morphisms of some category that are isomorphisms, or sometimes have only one-side inverse. If these morphisms are to be treated by computers, they should have some sort of termal representation, and this means that arise the problems concerning extensional and intensional equality. (Usually efficient treatment of isomorphisms by computers requires intensional equality.)

In the literature on reversible computations the history of computations is usually understood in the sense derived from Turing-machine protocols (commonly used are Turing machines with an additional *history tape*, cf. [3, 31]). So, pre-conditions of reversibility formulated in terms of history would require complete or partial preservation of history in this sense. From the point of view of category theory, natural are pre-conditions expressed in categorical terms, e.g. preliminary composition of a given morphism with some other morphism, application of a functor etc. This understanding of conditions of reversibility and that expressed in terms of 'history tape' are not mutually exclusive, but the categorical view accentuates other aspects of computation.

Let us consider several examples.

- If take the composition $\theta \circ h$ as in remark 4.2 of previous section, if $h =< f, g >$ then there exists θ' such that $\theta' \circ (\theta \circ h) = h$ (the part of computation represented by θ can be reversed).

- Let $f : A \to B$ be any morphism in T^0_{ind}, and assume that f is inversible on canonical elements of A and B, i.e., there exists $f' : B \to A$ such that for every canonical element $a : A$ $f'(fa) = a$ and for every canonical element $b : B$ $f(f'b) = b$. Then by theorem 3.7 f is inversible in the sense of extensional equality. From the point of view of reversible computations, the condition of reversibility is that f is applied to a closed term.

- The same may be expressed more "diagrammatically". Let us consider a diagram of the form

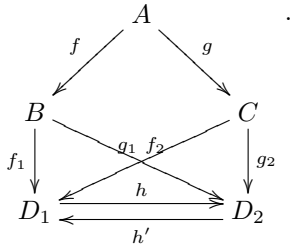
  $$A_0 \stackrel{f_0}{\to} A_1 \stackrel{f_1}{\to} ... \stackrel{f_{n-1}}{\to} A_n \stackrel{f_n}{\to} A \stackrel{f}{\to} B$$

  in T_{ind}, where A_0, A and B are 0-recursive and f is inversible on canonical elements. Intermediate types $A_1, ..., A_n$ may be arbitrary types of T_{ind}. Then there exists f' such that $(f' \circ f) \circ (f_n \circ ... \circ f_0)) = f_n \circ ... \circ f_0$.

- The following example comes from our study of graph rewriting. In categorical graph rewriting [24] most often are used so called single pushout (SPO) and double pushout (DPO) approaches. Single (respectively, double) pushout construction[8] is used to define graph transformation rules. Working on generalization of SPO and DPO to the case of attributed graphs [5, 22] we arrived to the situations where only the existence of a weak pushout is guaranteed. Consider two weak pushouts generated by

[8] We do not go into details here how the categories of graphs, equality of morphisms in these categories etc. are defined.

the same "span" (f, g):



Because of weak universality h and h' are not in general mutually inverse isomorphisms, but at the same time weak universality means commutativity of all diagrams above. For h and h', in particular, it means $(h' \circ h) \circ (f_1 \circ f) = f_1 \circ f$ and $(h \circ h') \circ (g_2 \circ g) = f_1 \circ g$. This can be seen as conditional reversibility.

These observations show the interest of application of diagrammatic methods and category theory to the theory of reversible computations. At the practical side, the use of graph rewriting techniques for treatment of diagrams may be advised.

5.3 Perspectives

In this paper we considered as main applications of our analysis of weak universality the applications to the study of reversible computations.

As we have outlined above, weak universality is related to conditional reversibility. Our motivating examples were coming mostly from categorical type theory. The same modelling language, categorical type theory, suggests that other forms of conditional reversibility would be interesting to study in future.

One of such forms could be context-dependent reversibility. Let us recall that the notion of retraction in λ-calculus, first defined in [6] (cf. also [28]) is context-dependent [9].

More general question is the meaning of different kinds of equality when reversibility of computations is studied.

It seems that there is little interaction between research communities studying reversible computations in connection with their physical (technical) realisations, computer architecture, thermodynamics etc., and more theoretical aspects such as meaning of reversibility itself. What does change when we consider the notion of reversibility with respect to different kinds of equality? When extensional and intensional equality are considered? When we modify the notion of equality between terms (programs), introducing new conversions? All this seems to be an important subject for future study.

[9] The type ρ is a retract of type τ if there are terms $C : \rho \to \tau$ and $D : \tau \to \rho$ (witnesses) such that $D \circ C =_{\beta\eta} \lambda x^{\rho}.x$ [28].Witness terms in this definition may contain free variables. If a context (a list of typed free variables) is fixed, no witness may exist for some contexts.

At more theoretical side, it would be interesting to explore the behaviour of categorical universal constructions with respect to different types of equality in more powerful systems of λ-calculus (with or without inductive types).

6 Conclusion

In purely mathematical approach to category theory various types of equality are treated indifferently, as part of definition of categorical structure, and no special role is given to intensional equality.

We considered in this paper several examples of categories based on T_{ind}, a system of lambda-calculus with inductive types and recursion. The aim was to underline the connection between the "strength" of categorical universal constructions and equality of morphisms treated under the angle of computational efficiency (decidable intensional equality and extensional equality that is in general undecidable). The examples were rather elementary, but illustrated specific properties of categories with intensional equality (typically, used in computer science) with respect to basic universal constructions.

We paid special attention to another domain of research interesting for practical computing. So called "reversible computations" are actively studied nowadays. From categorical point of view, the reversible computations may be considered as morphisms of some category that are isomorphisms, or sometimes have one-side inverse.

The fact that in most cases there exist only weakly universal constructions may be connected with the notion of *conditional reversibility*. Category theory and categorical logic (type theory, lambda calculus) suggest new forms of reversibility conditions that were not considered before in the study of reversible computations, for example, reversibility in the sense $(f' \circ f) \circ g = g$ instead of $f^{-1} \circ f = id$ (pre-composition with g as a condition of reversibility). They are much lighter (and may be more practical) than more traditional conditions concerning the history of computation in the style of Turing machine protocols.

References

[1] A. Asperti and G. Longo. Categories, types and structures - an introduction to category theory for the working computer scientist. *Foundations of computing*, MIT Press, 1991.

[2] M. S. Ager, O. Danvy, J. Midtgaard. A functional correspondence between monadic evaluators and abstract machines for languages with computational effects.- *Theoretical Computer Science*, 342, 149-172, 2005.

[3] C.H. Bennett. Logical Reversibility of Computation. -*IBM J.Res. Develop.,* 17, 525-532 (1973).

[4] N. Benton, J. Hughes, and E. Moggi. Monads and effects. In *G. Barthe, P. Dybjer, L. Pinto, and J. Saraiva, eds., Applied Semantics Advanced Lec-*

tures, LNCS, 2395, 42-122, Caminha, Portugal, September 2000. Springer-Verlag.

[5] B. Boisvert, L. Féraud, and S. Soloviev. Typed lambda-terms in categorical attributed graph rewriting. TOOLS 2011, June 30th, 2011, Zurich, Switzerland . Electronic Proceedings in Theoretical Computer Science (2011).

[6] K.Bruce, G. Longo. Provable isomorphisms and domain equations in models of typed languages. - *Proc. 17th symposium on Theory of Computing*, ACM, 263-272 (1985).

[7] D. Chemouil, S. Soloviev. Remarks on isomorphisms of simple inductive types.- *ENTCS, v. 85, Elsevier* (2003).

[8] . D. Chemouil. Types inductifs, isomorphismes et récriture extensionnelle. - Ph.D. Thesis, IRIT, University Toulouse-3, 2004.

[9] D. Chemouil. Isomorphisms of simple inductive types through extensional rewriting. *Math. Structures in Computer Science*, 15(5), 875-917, 2005.

[10] K. Gödel. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. -*Dialectica 12*, 280-287.

[11] J.R. Hindley. Basic Simple Type Theory. Cambridge University Press (2008).

[12] Homotopy Type Theory: Univalent Foundations of Mathematics. - Institute for Advanced Study, Princeton (2013).

[13] B. Jacobs. Categorical Logic and Type Theory. -*Studies in Logic and the Foundations of Mathematics*, 141 (1999).

[14] S. C. Kleene. Introduction to Metamathematics. - North-Holland PLC, 1952.

[15] G. Kreisel. Interpretation of analysis by means of constructive functionals of finite type. - In: A. Heyting (ed.). Constructivity in Mathematics, Amsterdam, North-Holland (1959), 101-128.

[16] J.Lambek, P.J.Scott. Introduction to Higher-Order Categorical Logic.- *Cambridge studies in advanced mathematics 7*, Cambridge University Press, 1988.

[17] R. Landauer. Irreversibility and heat generation in the computing process.- *IBM J. Res. Develop.*, 3, 183-191 (1961).

[18] Z. Luo. Computation and Reasoning. A Type Theory for Computer Science. *International Series of Monographs on Computer Science 11*, Oxford Science Publications, Clarendon Press, Oxford, 1994.

[19] S. Mac Lane. Categories for the Working Mathematician, 2nd edition, *Graduate texts in mathematics*, 5. Springer-Verlag, 1998.

[20] P. Martin-Löf. Intuitionistic type theory. Notes by G. Sambin of a series of lectures given in Padua, June 1980. Bibliopolis, 1984.

[21] L. Marie-Magdeleine. - Sous-typage coercitif en présence de réductions non-standards dans un système aux types dépendants. -Ph.D. Thesis, University Toulouse-3, 2009.

[22] M. Rebout, L. Féraud, L. Marie-Magdeleine, and S. Soloviev. Computations in Graph Rewriting: Inductive types and Pullbacks in DPO Approach. In Szmuc, T., Szpyrka, M., and Zendulka, J., eds., CEE-SET 2009, Krakow, Poland, October 2009, LNCS, 7054, 150 - 163. Springer-Verlag (2011).

[23] M. Saeedi, I.L.Markov. Syntheisis and Optimization of Reversible Circuits. - A Survey. - *arXiv:* 1110.2574v1[cs.ET](12 Oct.2011).

[24] Rozenberg, G., ed. . Handbook of Graph Grammars and Computing by Graph Transformations. - Volume 1: Foundations. World Scientific (1997).

[25] S. Soloviev. The Category of Finite Sets and Cartesian Closed Categories.- In: Theoretical applications of mathematical logic III, vol.105 of "Zapiski Nauchnykh Seminarov LOMI, 174-194 (1981), eng. transl.: Journal of Soviet Mathematics 22 (3), 1387-1400 (1983).

[26] S. Soloviev.Proof of a conjecture of S. Mac Lane.-*Annals of Pure and Applied Logic*, 90, 1-3, p.101-162 (1997).

[27] S. Soloviev, D. Chemouil. Some Algebraic Structures in Lambda-Calculus with Inductive Types. - In: Proc. Types 03, LNCS, v. 3085 (2004).

[28] C. Stirling. Proof Systems for Retracts in Simply Typed Lambda Calculus. ICALP (2) 2013: 398-409.

[29] T. Streicher. Investigations Into Intensional Type Theory. Habilitationsschrift. München, November 1993.

[30] W. Tait. Intensional interpretation of functionals of finite type. I. - *The journal of symbolic logic 32*, 198-212 (1967).

[31] P. Vitanyi. Time, Space and Energy in Reversible Computing. *arXiv:* cs/0504088v1[cs.CC](20 apr. 2005).