A Construction of Typed Lambda Models Related to Feasible Computability

Vladimir Sazonov*   Andrei Voronkov†

February 9, 1993

1 Introduction

Abstract

In this paper we define an approach to the definition of *computable functionals.* Computability is considered in a very abstract setting and is not restricted to Turing computability or, say, polynomial computability. We assume to start from some *basic* class of domains and a *basic* class of functions defined on these domains. (An example may be natural numbers with polynomially computable functions). Then we define what are "all" corresponding functionals of higher types which add nothing new to these basic functions. We call such functionals *computable* or, more technically and adequately speaking, *substitutable.* (Similarly, in D.Scott's domains we say about *continuous* functionals as about far-reaching abstraction of computable ones.) Our results are applicable to quite arbitrary complexity classes, satisfying a very general `Assumption`

which says that the set of basic computable functions should be closed under composition and should contain all constant and projection functions.

The problem raised up in this paper is related to the general notions of computations in different programming paradigms over different data types. The problem may be stated as follows: what kind of constructions can a programming language use to define a specific class of computations. If we consider the set of the basic functions as the target for our programming language, then a programming language should give us a way to combine finite sets of predefined (built-in) functions (predicates, computations) to obtain new functions (predicates, computations). The constructions of a programming language can thus be considered as higher type *functionals*, which allow new functions to be obtained from the previously defined functions.

*Research Center of Artificial Intelligence, Program Systems Institute, Pereslavl-Zalesski, 152140, Russia (sazonov@logic.botik.yaroslavl.su)

†CRIN-CNRS and INRIA-Lorraine, Batiment Loria, BP 239, 54506 Vandœuvre-les-Nancy Cedex, France (voronkov@loria.fr)

There are additional motivations for studying this problem. Very similar definitions have been used in the paper of the first author [SS 86], where authors defined a denotational semantics of the language of Σ-expressions. In that paper a finite type structure was defined on the base of some given in advance class $\mathcal{P}$ of "sigma-like" predicates over some universe of basic objects (in fact, of sets). The point of [SS 86] was that the nature of any such universe (a model of the Kripke-Platek set theory) does not in general allow arbitrary predicates to be considered as "sigma-like" and, therefore, the problem of building finite type structure *corresponding* to a given appropriate class $\mathcal{P}$ was essential. Here we also consider that only some given basic functions are admissible (for whatever the aim) and look for the corresponding finite type structure.

Another general motivation comes from *realizability* notion [Kle 52] for some complexity classes, like various polynomial realizability approaches considered by Buss [Bus 86], Cook and Urquhart [CU 89] and the first author [Saz 89]. In [Vor 90] an abstract approach to realizability for different data types has been defined. The essential idea of [Vor 90] was to consider arbitrary predefined sets of data types (models) and to define realizability for those data types. That notion of realizability, however, depended on *enumerations* of a model, and thus it implicitly used natural numbers. In this paper the definition of computable (substitutable) functionals of higher types will be generalized to the most abstract level. We do not even assume having natural numbers as a predefined domain. We are going to consider realizability based on the computable functionals of this paper in a forthcoming publication. Different notions of realizability based on different lambda models have been considered in [Mit 86, MO 86, CFS 88].

The problem of defining computable functionals from computable functions is also related to the *interoperability* problem. Roughly speaking, the problem is how to combine different programming paradigms, or how to design constructions of a language which unifies these paradigms. Since we do not make any restrictions on the structure of basic domains and (almost) no restrictions on the structure of basic functions, we can consider them as representing different paradigms. In this case corresponding functionals will give us a way to combine the basic functions obtaining a unified paradigm.

After preliminaries we discuss the pure functional case in Sections 3 and 4. We show two ways to give such a definition. One way treats the corresponding functionals as elements of the family of *all* possible functions between two domains (C-functionals of Definition 2). The other way ($\bar{C}$-functionals of Definition 3) gives an extensional model of λ-calculus. In Section 3 we prove the combinatory completeness of both versions. Section 4 is devoted to the connection of the two definitions. In fact, we shall prove that the second model is a factor of the first model by a suitable partial equivalence relation.

We do not compare our approach with existing definitions of functionals, computable in some complexity classes. Our definition uses neither any concrete model of computation nor even natural numbers. We can however note, that e.g. the functional of R of bounded primitive recursion from [CU 89] *is* computable in our model, if we take polynomially computable *functions* on natural numbers as the basis.

In the subsequent sections 5 and 6 we show how to generalize the definition to allow sums $+$ and products $\times$ to be also considered.

2 Preliminaries

We assume to have a set of basic types, to be denoted by $a, b, c \ldots$ (unlike arbitrary types $\alpha, \beta, \gamma \ldots$) together with associated domains (non-empty sets) D_a. The same notation may be used for lists of types with corresponding type variables of basic types a,b,c,... *This agreement will be held throughout the paper, so whenever we denote a type by a roman letter, this type is a basic one.*

Definition 1 (Types and their levels) Define inductively

1. Each basic type is a type; if α, β are types then $\alpha \to \beta$, $\alpha \times \beta$, and $\alpha + \beta$, are types.
2. level(any basic type) is 0;
3. $\text{level}(\alpha \to \beta) = \max(1 + \text{level}(\alpha), \text{level}(\beta))$;
4. $\text{level}(\alpha \times \beta) = \max(\text{level}(\alpha), \text{level}(\beta))$,
5. $\text{level}(\alpha + \beta) = 2 + \max(\text{level}(\alpha), \text{level}(\beta))$.

Types not involving $\times$ and $+$ are called *pure functional types.*

We write any type $(\alpha_1 \to (\alpha_2 \to \ldots \to (\alpha_n \to \beta) \ldots))$ as $\alpha_1 \to \alpha_2 \to \ldots \to \alpha_n \to \beta$ or as $\alpha_1, \alpha_2, \ldots, \alpha_n \to \beta$. If $D, E, \ldots, F$ are any domains (sets) then $D \to E$ denotes the set of *all* mappings from D to E, and likewise for $D \to E \to \ldots \to F = D, E \ldots \to F$. Tuples $x_1, \ldots, x_n$ of consecutive arguments to a function(al) f will be denoted by $\bar{x}$, and tuples of corresponding types by $\bar{\alpha} = \alpha_1, \ldots \alpha_n$. We shall also use somewhat ambiguous but useful denotation of composition $g \circ f = \lambda \bar{x}.g(f\bar{x}) = \lambda x_1 \ldots \lambda x_n.g(fx_1 \ldots x_n)$, lambda abstraction being often considered as a *meta*mathematical operator.

Domains D_α of *full type structure* for each non-basic type α are defined by $D_{\alpha\to\beta} = D_\alpha \to D_\beta$, (the set of *all* functions from D_α to D_β), $D_{\alpha\times\beta} = D_\alpha \times D_\beta$ (the cartesian product) and $D_{\alpha+\beta} = D_\alpha + D_\beta$ (the separated union). Below we shall define *functionals* for each type α. If α has level n, then we say that any functional of type α is also of level n.

Our main aim is the following. Given a class of computable *functions,* i.e. level 1 functionals of pure functional types, to construct the set of computable *functionals* in all higher types. It is easy to see that level 1 functionals may be identified with n-place functions. The higher type computable functionals must be defined in such a way that an arbitrary combination of computable functionals cannot define a non-computable level 1 function.

We shall define two versions $\bar{C}$ and C of computable functionals. The first version defines domains $\bar{C}_\alpha$ as distinguished subsets of D_α. The second version defines domains C_α *not* as subsets of D_α. As we shall see later, C_α proves to be a factordomain of $\bar{C}_\alpha$. We let $C_a = \bar{C}_a = D_a$ for all basic types a and $C_\alpha = \bar{C}_\alpha \subseteq D_\alpha$ for types of level 1. We assume that the set of these *computable basic functions* satisfies the following general

`Assumption`. Let t be any well-typed term made of (names of) basic computable functions and of variables v_i and constant values of level 0. Then the function $\lambda v_1 \ldots \lambda v_k.t$ of level 1 is required to be computable.

3 Purely functional case

We first extend the definition of domains $\bar{C}_\alpha \subseteq D_\alpha$ of computable functionals for levels > 1 in the following way.

Definition 2 *Let τ be a type $\tau = \alpha_1, \ldots, \alpha_n \to b$ of level $n > 1$ (i.e. with some α_i not basic). Then*

$$\bar{C}_\tau \rightleftharpoons \{f \in D_{\alpha_1} \to \ldots \to D_{\alpha_n} \to D_b \mid \forall \bar{a} \text{ basic and } \forall g_i \in C_{\bar{a}\to\alpha_i}, \; \lambda\bar{x}.f(g_1\bar{x})\ldots(g_n\bar{x}) \in C_{\bar{a}\to b}\}.$$

The another version of computable functionals based on *the same* class of computable basic functions is defined similarly.

Definition 3 [1] *Let τ be a type $\tau = \alpha_1, \ldots, \alpha_n \to b$ of level $n > 1$ (i.e. with some α_i not basic). Then*

$$C_\tau \rightleftharpoons \{f \in C_{\alpha_1} \to \ldots \to C_{\alpha_n} \to C_b \mid \forall \bar{a} \text{ basic and } \forall g_i \in C_{\bar{a}\to\alpha_i}, \; \lambda\bar{x}.f(g_1\bar{x})\ldots(g_n\bar{x}) \in C_{\bar{a}\to b}\}.$$

Hereinafter we shall sometimes call function(al)s from C and $\bar{C}$ *C-functionals* and *$\bar{C}$-functionals*. By a *C-family* $\bar{f} \in C_{\bar{a}\to\alpha}$ we mean a tuple of functionals $f_i \in C_{a_i\to\alpha}$, and likewise for $\bar{C}$.

Correctness of the definition. To prove that Definition 3 is correct we have to show that the expression $\lambda\bar{x}.f(g_1\bar{x})\ldots(g_n\bar{x})$ makes sense. To this end it is sufficient to show that the set of all C-functionals is closed under application, i.e. $fx \in C_\beta$, whenever $x \in \alpha$ and $f \in \alpha \to \beta$.

The proof is by induction. We assume that all C-functionals introduced on the previous steps for types with level $< d$, are closed under application. Let us also assume that the class of C-functionals of type level $< d$ admits dummy arguments, i.e. for every such functional f, the functional $g = \lambda x.f$ is also C-functional, provided that its level also $< d$.

Let us show that any C-functional $f \in C_{\alpha_1\ldots\alpha_n\to b}$ of type level d are closed under application. So, let $\bar{x} = \bar{u}, \bar{v}$ be all arguments for f. Show that $f\bar{g} = \lambda\bar{v}.f\bar{g}\bar{v}$ is also C-functional,i.e. that such is the functional $\lambda\bar{x}.f\bar{g}(\bar{v}\bar{x})$, for $\bar{x}$ basic. The latter functional is equivalent to $\lambda\bar{x}.f((\lambda\bar{y}.\bar{g})\bar{x})(\bar{v}\bar{x})$, which is in C, by induction hypothesis.

It is also easy to see that C-functionals of type level $\leq d$ admit dummy arguments, when the type level of the resulting functional is $\leq d$. $\square$

It is also useful to note that, for $\bar{a}$ basic,

$$C_{\bar{a}\to\bar{\alpha}\to b} = \{f \in C_{\bar{a}} \to C_{\bar{\alpha}} \to C_b \mid \forall\bar{y} \text{ basic, } \lambda\bar{x}\bar{y}.f\bar{x}(\bar{g}\bar{y}) \text{ is } C\text{-function, for all basic } \bar{x}, \bar{y} \text{ and } C\text{-families } \bar{g}\}$$

[1]This Definition and corresponding proofs below mainly follow [SS 86]. Another equivalent and even simpler definition could be given by a (more general) approach of Section 5, when restricted to the pure functional case.

Furthermore, *the family of domains C_α comprises an extensional model of typed lambda calculus.* This is because C-functionals are closed under application; thereby, we can treat any $C_{\alpha\to\beta}$ as a set of extensional functions from C_α to C_β) and by the following Proposition and its Corollary. The definition of a (well-typed) applicative term is standard.

Proposition 4 *Let t be any applicative term all variables of which are in $\bar{y}$. For all C-families $\bar{f}$ and basic $\bar{x}$ of appropriate types, $\lambda\bar{x}.t[(\bar{f}\bar{x})/\bar{y}]$ is a C-function.*

Proof. The case t is a variable y_i or a constant is trivial. Let t be an application t_1t_2. Induction hypothesis gives that $\lambda\bar{x}.t_1[(\bar{g}\bar{x})/\bar{y}]$ and $\lambda\bar{x}.t_2[(\bar{g}\bar{x})/\bar{y}]$ are C-functions, for all $\bar{g}\in C$. By the `Assumption` we have that $\lambda\bar{x}.(\lambda\bar{x}.t_1[(\bar{g}\bar{x})/\bar{y}]\bar{x})(\lambda\bar{x}.t_2[(\bar{g}\bar{x})/\bar{y}]\bar{x}) = \lambda\bar{x}.(t_1[(\bar{g}\bar{x})/\bar{y}])(t_2[(\bar{g}\bar{x})/\bar{y}])$ is a C-function, for all $\bar{g}\in C$. Thus, t_1t_2 is in C. ■

Corollary 5 (Combinatory completeness of C) *For any applicative term t of type β all variables of which are in $\bar{x}$, x_i of types α_i, the mapping*

$$\lambda\bar{x}.t \in C_{\alpha_1} \to \ldots \to C_{\alpha_n} \to C_\beta$$

is an element of $C_{\alpha_1\to\ldots\to\alpha_n\to\beta}$.
In particular, the ordinary combinators $\mathbf{S} = \lambda xyz.xz(yz)$ and $\mathbf{K} = \lambda xy.x$ of any appropriate type are C-functionals.

Proof. Let $\bar{y}$ be variables such that $t\bar{y}$ is of a basic type b. Then $\lambda\bar{x}\bar{y}.t\bar{y}$ is a C-functional by Definition 3 and Proposition 4. Equality $\lambda\bar{x}\bar{y}.t\bar{y} = \lambda\bar{x}.t$ completes the proof. ■

The same considerations prove

Corollary 6 (Combinatory completeness of $\bar{C}$) *For any applicative term t of type β all variables of which are in $\bar{x}$, x_i of types α_i, the mapping*

$$\lambda\bar{x} \in \bar{C}_{\alpha_1} \to \ldots \to \bar{C}_{\alpha_n} \to \bar{C}_\beta$$

is an element of $\bar{C}_{\alpha_1\to\ldots\to\alpha_n\to\beta}$.
In particular, the ordinary combinators $\bar{\mathbf{S}} = \lambda xyz.xz(yz)$ and $\bar{\mathbf{K}} = \lambda xy.x$ of any appropriate type are C-functionals.

As an important example we note, that for the case of polynomial-time computability the functional R, (resp. $\bar{R}$) of bounded primitive recursion considered in [CU 89] also proves to be in C (resp. in $\bar{C}$).

There is a certain reason to study both models C and $\bar{C}$. If we introduce notions of realizability as is it done in [Vor 90] and [Vor 91]. The first way corresponds to the model $\bar{C}$, whilst the second — to C. The first way gives realizability logic with realizable formulas classically true, whilst the second can lead to formulas realizable but classically false, as in the Kleene realizability [Kle 52].

4 On the connection between C and $\bar{C}$

In this section we consider only the pure functional case. We remember that C_α, respectively $\bar{C}_\alpha$, denote domains of computable functionals of type α according to our two definitions. The model $\bar{C}_\alpha$ is not an extensional lambda model (it is a combinatory algebra). It contains "extra elements" and will be factorized.

Introduce on each $\bar{C}_\alpha$ a *logical* partial equivalence relation (PER) $\sim_\alpha$ which is induced by the equality relation on basic domains (see [Mit 90]). This means that (for all elements considered in $\bar{C}$)

$$x \sim_b x' \rightleftharpoons x = x' \in C_b (= \bar{C}_b = D_b) \text{ and}$$

$$g \sim_{\alpha\to\beta} g' \rightleftharpoons \forall (y \sim_\alpha y').(gy \sim_\beta g'y').$$

Here $\rightleftharpoons$ means "equal by definition". Also define $\tilde{C}_\alpha = \{x \in C_\alpha \mid x \sim_\alpha x\}$.

According to [Mit 90] $\bar{C}_\alpha / \sim_\alpha = \tilde{C}_\alpha / \sim_\alpha$ is an (extensional) lambda model. Moreover, after considerations of the following subsections, we shall prove

Theorem 7 $C_\alpha \cong \bar{C}_\alpha / \sim_\alpha = \tilde{C}_\alpha / \sim_\alpha \cong D_\alpha / \sim_\alpha$.

4.1 A computable retraction embedding of C into $\bar{C}$

For each type α we define a *computable retraction pair* $i_\alpha : C_\alpha \to \bar{C}_\alpha, j_\alpha : \bar{C}_\alpha \to C_\alpha$ with $j_\alpha \circ i_\alpha = id_\alpha$, where $j_\alpha : D_\alpha \dot{\to} C_\alpha$, is a *partial* mapping from D_α whose domain includes $\bar{C}_\alpha \subseteq D_\alpha$. Here $\dot{\to}$ is used for partial functions.

Computability of i_α and j_α means that their composition with arbitrary computable $f \in C_{\bar{a}\to\alpha}$, respectively, $g \in \bar{C}_{\bar{a}\to\alpha}$, is computable, i.e. belongs to $\bar{C}_{\bar{a}\to\alpha}$ or, respectively, to $C_{\bar{a}\to\alpha}$.

The following general lemmas on C- and $\bar{C}$-computability give a hint why this property is related to computability in C ($\bar{C}$):

Lemma 8 *Suppose $f_{\bar{\mathsf{a}}} : C_\alpha \to C_\beta$ with $\bar{\mathsf{a}} \in C_{\bar{a}}$ is a family of functions (or just one function, for $\bar{a}$ the empty list of basic types) such that $\lambda\bar{\mathsf{a}}\bar{\mathsf{b}}.f_{\bar{\mathsf{a}}}(g(\bar{\mathsf{b}})) \in C_{\bar{a},\bar{b}\to\beta} \subseteq C_{\bar{a},\bar{b}} \to C_\beta$ for each $\varphi \in C_{\bar{b}\to\alpha}$. Then $\lambda\bar{\mathsf{a}}.f_{\bar{\mathsf{a}}} \in C_{\bar{a}\to(\alpha\to\beta)}$ (and conversely).* □

Lemma 9 *Suppose $g_{\bar{\mathsf{a}}} : D_\alpha \to D_\beta$ with $\bar{\mathsf{a}} \in D_{\bar{a}}$ is a family of functions such that $\lambda\bar{\mathsf{a}}\bar{\mathsf{b}}.f_{\bar{\mathsf{a}}}(\bar{g}(\mathsf{b})) \in \bar{C}_{\bar{a},\bar{b}\to\beta} \subseteq D_{\bar{a},\bar{b}\to\beta}$ for each $\bar{g} \in \bar{C}_{\bar{b}\to\alpha}$. Then $\lambda\bar{\mathsf{a}}.f_{\bar{\mathsf{a}}} \in \bar{C}_{\bar{a}\to(\alpha\to\beta)}$ (and conversely).* □

Let i_b and j_b for basic types b be (obviously) computable identity mappings between C_b and $D_b = \bar{C}_b = C_b$. Generally, suppose i_α, j_α and i_β, j_β for some types α and β have been defined as appropriate computable retraction pairs. Then define

$$j_{\alpha\to\beta} g \rightleftharpoons j_\beta \circ g \circ i_\alpha,$$

if the latter mapping belongs to $C_{\alpha\to\beta}$, and let

$$i_{\alpha\to\beta} f \rightleftharpoons \overline{i_\beta \circ f \circ j_\alpha}$$

be any extension of the partial mapping $i_\beta \circ f \circ j_\alpha : D_\alpha \dot{\to} D_\beta$ so that $i_{\alpha\to\beta} f \in D_{\alpha\to\beta}$. Then Lemmas 8 and 9 imply that $i_{\alpha\to\beta}$ and $j_{\alpha\to\beta}$ also form a computable retraction pair between $C_{\alpha\to\beta}$ and $\bar{C}_{\alpha\to\beta}$. (We omit the proof.)

Note, that by definition of $i_{\alpha\to\beta}$ always

$$i_\beta(fx) = (i_{\alpha\to\beta} f)(i_\alpha x). \tag{1}$$

Thus the (extensional) lambda-model C may be considered as an applicative subalgebra of the (non-extensional) applicative combinatory algebra $\bar{C}$. However, such embedding of C into $\bar{C}$ is neither unique nor canonical because of our definition of $i_{\alpha\to\beta}$. Unfortunately, we cannot even guarantee that i maps combinators $\mathbf{S}$ and $\mathbf{K}$ of C-model to the corresponding combinators of $\bar{C}$ or D. As for the reverse, we can easily show

$$j\bar{\mathbf{S}} = \mathbf{S} \text{ and } j\bar{\mathbf{K}} = \mathbf{K}$$

using the equality $jgx_1 \ldots x_n = j(g(ix_1)\ldots(ix_n))$, identity (1) and the extensionality of C.

4.2 A connection between the full type structure D and any other typed lambda model E

Let E_α be an arbitrary (extensional) lambda model with pure functional types and $\tilde{j}_b : D_b \to E_b$ be a *partial function onto*, for each basic type. Let the family $\tilde{j}_\alpha : D_\alpha \to E_\alpha$ be the corresponding induced logical relation (cf. [Sta 85] and [Mit 90]).

Proposition 10 *All relations $\tilde{j}_\alpha$ are partial functions onto E_α.*

Proof. We shall simultaneously construct retraction (right inverse) mappings $\tilde{i}_\alpha : E_\alpha \to D_\alpha$ with

$$\tilde{j}_\alpha \circ \tilde{i}_\alpha = id_\alpha.$$

Assume $\tilde{i}_\alpha$ and $\tilde{i}_\beta$ are retractions of $\tilde{j}_\alpha$ and $\tilde{j}_\beta$ respectively (the induction hypothesis). Define

$$\tilde{i}_{\alpha\to\beta} f \rightleftharpoons \overline{\tilde{i}_\beta \circ f \circ \tilde{j}_\alpha},$$

where overlining denotes the result of an arbitrary extension of a partial mapping to a total one. From the definition of logical relations [Mit 90] it follows that

$$\tilde{j}_{\alpha\to\beta}(g, f) \text{ iff } f \circ \tilde{j}_\alpha \subseteq \tilde{j}_\beta \circ g.$$

The statement $\tilde{j}_{\alpha\to\beta}(g, f)$, the induction hypothesis and extensionality of the lambda model E imply $f = \tilde{j}_\beta \circ g \circ \tilde{i}_\alpha$. Thus f is uniquely defined by g. I.e. $\tilde{j}_{\alpha\to\beta} g = f$ is actually a (partial) function and this function is onto because for arbitrary $f \in D_{\alpha\to\beta} \subseteq D_\alpha \to D_\beta$ we have $\tilde{j}_{\alpha\to\beta} g = f$ for $g \rightleftharpoons \tilde{i}_{\alpha\to\beta} f = \overline{\tilde{i}_\beta \circ f \circ \tilde{j}_\alpha}$. Indeed, $f \circ \tilde{j}_\alpha = \tilde{j}_\beta \circ \tilde{i}_\beta \circ f \circ \tilde{j}_\alpha \subseteq \tilde{j}_\beta \circ \overline{\tilde{i}_\beta \circ f \circ \tilde{j}_\alpha} = \tilde{j}_\beta \circ g$. $\square$

Note, that even if all $\tilde{j}_b$ are total functions for a basic b, the mappings $\tilde{j}_\alpha$ are typically only partial and, moreover, even non-injective (because the corresponding retractions $\tilde{i}_\alpha$ are not uniquely defined). As any logical relation, $\tilde{j}$ relates corresponding combinators of lambda models D and E. In particular, $\tilde{j}$ is defined when applied to combinators.

Definition 11 *Let $x \sim^E_\alpha y \rightleftharpoons \tilde{j}_\alpha x = \tilde{j}_\alpha y$ (and both values are defined!) be PER on each D_α and $\tilde{E}_\alpha \rightleftharpoons dom(\tilde{j}_\alpha) = \{x \in D_\alpha \mid x \sim^E_\alpha x\}$.*

Proposition 12

1. *$g \sim^E g', y \sim^E y'$ imply $gy \sim^E g'y'$. In particular, $\{\tilde{E}_\alpha\}$ is a typed combinatory subalgebra of $\{D_\alpha\}$ (with the same combinators and application).*
2. *$\tilde{i}_\beta(fx) = (\tilde{i}_{\alpha\to\beta} f)(\tilde{i}_\alpha x)$.*
3. *$\forall(y \sim^E y').(gy \sim^E gy')$ iff $\exists f : E_\alpha \to E_\beta (f \circ \tilde{j} \subseteq \tilde{j} \circ g)$. The latter is equivalent to $g \sim^E g$ whenever the corresponding unique $f = \tilde{j} \circ g \circ \tilde{i}$, if any, is actually in $E_{\alpha\to\beta}$.*
4. *$\forall(y \sim^E y').(gy \sim^E g'y')$ iff $\exists f : E_\alpha \to E_\beta (f \circ \tilde{j} \subseteq \tilde{j} \circ g \& f \circ \tilde{j} \subseteq \tilde{j} \circ g')$.*
5. *$g \sim^E g$ or $g' \sim^E g'$ imply*

$$g \sim^E g' \text{ iff } \forall(y \sim^E y).(gy \sim^E g'y) \text{ iff } \forall(y \sim^E y').(gy \sim^E g'y').$$

Thus, $\{\sim^E_\alpha\}$ is logical PER on the typed combinatory algebra $\{\tilde{E}_\alpha\}$. $\square$

Theorem 13 *$E \cong D/\sim^E = \tilde{E}/\sim^E$.*

Proof. Follows from Proposition 12 (1,2). ∎

4.3 When E is C

The general construction of $\tilde{j}_\alpha$ and $\tilde{i}_\alpha$ has been defined relative to any given lambda model E_α (with some possible variations for $\tilde{i}_\alpha$, but not for $\tilde{j}_\alpha$!). Consider the special case $E_\alpha = C_\alpha$ with

$$\tilde{j}_\alpha \subseteq j_\alpha : D_\alpha \dot{\to} C_\alpha \text{ and } \tilde{i}_\alpha = i_\alpha : C_\alpha \to D_\alpha$$

for any given i_α and j_α from Subsection 4.1 for which $\mathrm{dom}(j_\alpha) = \bar{C}_\alpha$ (and $\mathrm{range}(i_\alpha) \subseteq \bar{C}_\alpha$) and therefore $\mathrm{dom}(\tilde{j}_\alpha) = \tilde{C}_\alpha \subseteq \bar{C}_\alpha$. Here $\tilde{C}_\alpha$ is used in the sense of Definition 11. In fact, it follows from this Subsection that these domains coincide with those from Theorem 7 having the same denotation.

Note, that now $\tilde{j}_b$ and $\tilde{i}_b$ are both identity functions for all basic b. Suppose that $\tilde{j}_\alpha \subseteq j_\alpha$ and $\tilde{j}_\beta \subseteq j_\beta$ for some types α and β and, moreover, $\tilde{i}_\alpha$ and $\tilde{i}_\beta$ have been defined according to Subsection 4.2, so that $\tilde{i}_\alpha = i_\alpha$, $\tilde{i}_\beta = i_\beta$. Then we may define according to that subsection

$$\tilde{i}_{\alpha\to\beta} f \rightleftharpoons i_{\alpha\to\beta} f \supseteq i_\beta \circ f \circ j_\alpha \supseteq \tilde{i}_\beta \circ f \circ \tilde{j}_\alpha.$$

Also, as required, $\tilde{j}_{\alpha\to\beta} \subseteq j_{\alpha\to\beta}$ because

$$\tilde{j}_{\alpha\to\beta} g = \tilde{j}_\beta \circ g \circ \tilde{i}_\alpha \subseteq j_\beta \circ g \circ i_\alpha = j_{\alpha\to\beta} g,$$

if the first term is defined (and is a total function; therefore, the inclusion is indeed an equality). It follows from Proposition 12 (5) that corresponding $\sim^C$ is a logical PER on $\tilde{C} \subseteq \bar{C}$ which, moreover, coincides with $\sim$ from Theorem 7. The latter holds according to Proposition 12 (4) and (3): it only suffices to note that $f = \tilde{j}_\beta \circ g \circ \tilde{i}_\alpha = j_\beta \circ g \circ i_\alpha = j_{\alpha\to\beta} g$ is in $C_{\alpha\to\beta}$ for $g \in \bar{C}_{\alpha\to\beta}$. Therefore the two versions of $\tilde{C}$ coincide.

By using Corollary 13 this simultaneously proves Theorem 7:

$$C \cong \bar{C}/\!\sim = \tilde{C}/\!\sim = D/\!\sim .$$

5 The case of sums

The rest of the paper is concerned with C-version. In this section we show how to introduce sums in addition to function spaces. For simplicity, we postpone considering product types to Section 6 and consider only the case of functional types and sums. Moreover, the reader is advised even to restrict himself first to the case with a unique basic type b. It simplifies all the details below considerably. *We also assume that some domain D_a contains at least two elements.*

Unfortunately, the ordinary set theoretical definition of $C_{\gamma+\delta}$ as the *separated union* of C_γ and C_δ leads to technical problems. Instead, we shall use the well known idea to represent "disjunction" $\gamma \vee \delta = \gamma + \delta$ by the second order proposition $\forall\beta.((\gamma \to \beta) \to (\delta \to \beta) \to \beta)$ where we consider the quantification $\forall\beta$ as the cartesian product $\prod_b$ restricted to basic types. More formally, let B be the set of all basic types. Let $\{S_b \mid b \in B\}$ be a family of sets indexed by B. Then

$$\prod_b S_b \rightleftharpoons \{f : B \to \bigcup S_b \mid f(b) \in S_b,\ \text{for each } b \in B\}$$

Definition 14 *The sets C_τ, with τ constructed from $+$ and $\to$, are defined as follows.*

$$
\begin{array}{lll}
C_{\gamma+\delta} & \rightleftharpoons & \{f \in \prod_b C_{(\gamma\to b)\to(\delta\to b)\to b} \mid \\
& & \quad \exists x \in C_\gamma \forall b \forall g \in C_{\gamma\to b} \forall h \in C_{\delta\to b}(fbgh = gx) \ \textit{or} \\
& & \quad \exists y \in C_\delta \forall b \forall g \in C_{\gamma\to b} \forall h \in C_{\delta\to b}(fbgh = hy)\} \\
\\
C_{\alpha_1\ldots\alpha_n\to b} & \rightleftharpoons & \{f \in C_{\alpha_1} \to \ldots \to C_{\alpha_n} \to C_b \mid \\
& & \quad \forall \bar{a} \ \textit{basic and} \ \forall g_i \in C_{\bar{a}\to\alpha_i}, \\
& & \quad \lambda\bar{x}.f(g_1\bar{x})\ldots(g_n\bar{x}) \in C_{\bar{a}\to b}\}. \\
\\
C_{\alpha_1\ldots\alpha_n\to\gamma+\delta} & \rightleftharpoons & \{f \in C_{\alpha_1} \to \ldots \to C_{\alpha_n} \to C_{\gamma+\delta} \mid \\
& & \quad \forall \bar{a} \ \textit{basic and} \ \forall g_i \in C_{\bar{a}\to\alpha_i} \forall b \ \textit{basic}, \\
& & \quad \lambda\bar{x}.f(g_1\bar{x})\ldots(g_n\bar{x})b \in C_{\bar{a}\to(\gamma\to b)\to(\delta\to b)\to b}\}.
\end{array}
$$

Note, that "or" in the definition of $C_{\gamma+\delta}$ means that this domain is a union. In fact, we could equivalently define $C_{\gamma+\delta}$ as the union of C_γ and C_δ (which for simplicity are supposed to be disjoint) *factorized* by the following equivalence relation: $x \sim y$ iff x and y are nondistinguishable by any functionals g and h with basic values (i.e. always $gx = hy$).

It is easy to show that $C_{\gamma+\delta}$ is canonically isomorphic to this factorized union of C_γ and C_δ. Moreover, this factorizing proves to be *trivial* because all basic and therefore higher type elements are indeed *distinguishable* due to the fact that some basic domain contains at least two elements. Therefore *so defined sum is in fact isomorphic to the separated sum!* We adopt this somewhat indirect definition due to its technical advantage.

As in the case of pure functional types, one can prove by induction embeddings

$$C_{\alpha\to\beta} \subseteq C_\alpha \to C_\beta \text{ and } C_{\gamma+\delta} \subseteq \prod_b C_{(\gamma\to b)\to(\delta\to b)\to b}$$

which show that Definition 14 is correct.

We can also obviously prove that the classes C_τ are closed under application and dummy arguments.

The formulation and proof of *combinatory completeness* of domains $\{C_\tau\}$ is essentially identical to that for the pure functional case This also may be suitably extended to the case of lambda abstraction Λb over basic types so that we can consistently identify the following closed lambda-terms with their values (obtained by combinatory completeness) in the corresponding domains:

$$\mathrm{inl} \rightleftharpoons \lambda x \Lambda b \lambda g \lambda h.gx \in C_{\alpha\to\alpha+\beta}$$

$$\mathrm{inr} \rightleftharpoons \lambda x \Lambda b \lambda g \lambda h.hx \in C_{\beta\to\alpha+\beta}$$

$$[-,-] \rightleftharpoons \lambda g \lambda h \lambda f.\lambda\bar{w}.fb(\lambda x.gx\bar{w})(\lambda y.hy\bar{w}) \in C_{(\alpha\to\delta)\to(\beta\to\delta)\to(\alpha+\beta)\to\delta},$$

where the last b is the basic type from the decomposition $\delta = \delta_1\ldots\delta_k \to b$ and $\bar{w} = w_1,\ldots,w_k$, $w_s : \delta_s$, $s = 1,\ldots,k$, or, otherwise, b is w_{k+1}, if $\delta = (\delta_1\ldots\delta_k \to \alpha' + \beta')$ and $\bar{w} = w_1,\ldots,w_{k+3}$, $w_s : \delta_s$, $s = 1,\ldots,k$, $w_{k+2} : \alpha' \to b$, $w_{k+3} : \beta' \to b$.

Proposition 15 *For all* $g \in C_{\alpha\to\delta}$, $h \in C_{\beta\to\delta}$ *and* $F \in C_{\alpha+\beta\to\delta}$ *we have*

$$[g,h] \circ \mathit{inl} = g,\ \ [g,h] \circ \mathit{inr} = h,\ \ [F \circ \mathit{inl}, F \circ \mathit{inr}] = F,$$

so that $C_{\alpha+\beta}$ *is the separated sum (i.e. the co-product) of* C_α *and* C_β.

Proof. The first (and likewise the second) identity follows from

$$[g,h](\mathrm{inl}\, x)\bar{w} = \mathrm{inl}\, x b(\lambda x.gx\bar{w})(\lambda y.hy\bar{w}) = (\lambda x.gx\bar{w})x = gx\bar{w}$$

The third identity follows from the first two by $C_{\alpha+\beta} = \{\mathrm{inl}\, x \mid x \in C_\alpha\} \bigcup \{\mathrm{inr}\, y \mid y \in C_\beta\}$ and $[F\circ\mathrm{inl}, F\circ\mathrm{inr}]\circ\mathrm{inl} = F\circ\mathrm{inl}$ and $[F\circ\mathrm{inl}, F\circ\mathrm{inr}]\circ\mathrm{inr} = F \circ \mathrm{inr}$. ■

We conclude that $\{C_\tau\}$ *is an extensional model for typed* λ-β-η*-calculus with additional* (considered as) *primitive constructs* $[-,-]$, inl *and* inr *satisfying the above three identities.*

6 The general case (with product types)

In this section we show how to incorporate the product type into our constructions. Define

$$C_{\gamma\times\delta} \rightleftharpoons \{\langle x,y\rangle \mid x \in C_\gamma,\ y \in C_\delta\}$$

and $\langle x,y\rangle_1 = x$, $\langle x,y\rangle_2 = y$. Then add the following item to Definition 14.

$$\begin{array}{rcl} C_{\alpha_1\ldots\alpha_n\to\gamma\times\delta} & \rightleftharpoons & \{f \in C_{\alpha_1} \to \ldots \to C_{\alpha_n} \to C_{\gamma+\delta} \mid \\ & & \lambda\bar{x}.(f\bar{x})_1 \in C_{\alpha_1\ldots\alpha_n\to\gamma} \text{ and} \\ & & \lambda\bar{x}.(f\bar{x})_2 \in C_{\alpha_1\ldots\alpha_n\to\delta} \end{array}$$

All considerations are now the same as above or strightforward and give rise to the following conclusion:

> Domains $\{C_\tau\}$ comprise an extensional model for typed λ-β-η-calculus with additional primitive constructs $[-,-]$, inl, inr satisfying the three identities of Proposition 15 and also with constructs of *pairing* $\langle -,-\rangle$ and *projections* $(-)_1$, $(-)_2$ satisfying the following identities
>
> $$(\langle x_1,x_2\rangle)_i = x_i, i = 1,2, \text{ and } \langle (x)_1,(x)_2\rangle = x.$$

References

[Bus 86] S.R. Buss. The polynomial hierarchy and intuitionistic bounded arithmetic. In *Structure in Complexity Theory*, volume 223 of *Lecture Notes in Computer Science*, pages 77–103, 1986.

[CFS 88] A. Carboni, P. Freyd, and A. Scedrov. A categorical approach to realizability and polymorphic types. In M. Main and *e.a.*, editors, *Proceedings of 3rd ACM Workshop on Mathematical Foundations of Programming Languages Semantics*, volume 298 of *Lecture Notes in Computer Science*, pages 23–42. Springer Verlag, New Orleans, April 1988.

[CU 89] S. Cook and A. Urquhart. Functional interpretations of feasibly constructive arithmetic. In *Proc. 21st ACM STOC*, pages 107–112, 1989.

[Kle 52] S.C. Kleene. *Introduction to Metamathematics*. Van Nostrand P.C., Amsterdam, 1952.

[Mit 86] J.C. Mitchell. Abstract realizability for intuitionistic and relevant implication (abstract). *Journal of Symbolic Logic*, 51(3):851–852, 1986.

[Mit 90] J.C. Mitchell. Type systems for programming languages. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 8, pages 367–458. Elsevier Science, 1990.

[MO 86] J.C. Mitchell and M.J. O'Donnel. Realizability semantics for error-tolerant logics. In *Theoretical Aspects of Reasoning about Knowledge*. Morgan Kaufman, Los Altos, CA, 1986.

[Saz 89] V.Yu Sazonov. An equivalence between polynomial constructivity of Markov's principle and the equality p=np, (in Russian). In *Mathematical Logic and Algorithmic Problems*, volume 12 of *Trudy Instituta Matematiki*, pages 138–165. Nauka, Novosibirsk, 1989.

[Saz 90] V.Yu Sazonov. An equivalence between polynomial constructivity of Markov's principle and the equality p=np, (in Russian). In P. Petkov, editor, *Mathematical Logic, Proceedings of the Summer School and Conference on Mathematical Logic, Dedicated to the Ninetieth Anniversary of Arend Heyting, held in 1988, in Chaika (near Varna), Bulgaria*, pages 351–360. Plenum Press, NY, 1990. (An English version of a more complete 6.)

[SS 86] V.Yu. Sazonov and D.I. Sviridenko. Denotational semantics of the language of σ-expressions (in Russian). In *Logical Problems in the Theory of Data Types*, volume 114 of *Vychislitelnye sistemy*. Novosibirsk, 1986.

[Sta 85] R. Statman. Logical relations and the typed λ-calculus. volume 65, pages 85–97, 1985.

[Vor 90] A.A. Voronkov. Towards the theory of programming in constructive logic. In N. Jones, editor, *Proc. of the European Symp. on Programming*, volume 432 of *Lecture Notes in Computer Science*, pages 421–435. Springer Verlag, 1990.

[Vor 91] A.A. Voronkov. N-realizability: one more constructive semantics. Technical Report 71, Department of Mathematics, Monash University, 1991.