A Construction of Typed Lambda Models Related to Feasible Computability

Vladimir Sazonov* Andrei Voronkov†

Abstract

In this paper we develop an approach to the notion of *computable functionals* in a very abstract setting not restricted to Turing 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 polytime 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 of functions, satisfying a very general `Assumption`.

1 Introduction

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-Zalessky, 152140, Russia (sazonov@logic.botik.yaroslavl.su). Supported by Russian Foundation of Fundamental Investigations (project 93-011-16016).

†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 (essentially for pure functional case) 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.

As more earlier closely related sources we mention Banach-Mazur functionals (of level 2; cf. [Rog 67]) based on the class of all total recursive functions over natural numbers and also Longo and Moggi's hereditary partial effective functionals [LM 84] based on the class of all partial recursive functions which prove to coincide with monotonic and continuous effective finite type functionals.

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 90]. 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.

Here 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. feasible functionals from [CU 89] based on bounded primitive recursion *are* computable in our $\bar{C}$-model, if we take polynomially computable *functions* on natural numbers as the basis.

After preliminaries we discuss the pure functional case in Sections 3 and 4 along two ways. One approach treats computable functionals as some elements of the ordinary *full* type structure D_α ($\bar{C}$-functionals of Definition 2). The other way (C-functionals of Definition 3) gives an extensional model of λ-calculus. We prove in Section 3 the combinatory completeness of both versions and in Section 4 that the second model is a factor of the first model by a suitable partial equivalence relation. In the subsequent sections 5 and 6 we show how to generalize the definition of $C$to allow sum and product types.

The authors are grateful to the anonimous referee for useful comments and to G.Longo for taking our attention to essential reference [LM 84].

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) $D_a, D_b, D_c, \ldots$. Let corresponding variables of basic types be $\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots$.

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 $D \to E \to \ldots \to F = D, E \ldots \to F$.

Domains D_τ of *full type structure* for each non-basic type τ 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). Elements of these domains (and of defined below domains $\bar{C}_\tau$ and C_τ) are called *functionals* of corresponding types (and levels). Denotation like $C_{\bar{\alpha}} \to C_\beta$ will abbreviate $C_{\alpha_1} \to \ldots \to C_{\alpha_n} \to C_\beta$.

Our main aim is the following. Given a class of (so-called) computable *functions*, i.e. level 1 functionals of pure functional types, to construct the set of corresponding computable *functionals* in all higher types. (Level 1 functionals are identified with n-place functions.) The higher type computable functionals must be defined in such a way that any combination of computable functionals cannot define a non-computable level 1 function.

We shall define two versions $\bar{C}$ and C of computable functionals. Domains $\bar{C}_\alpha$ are distinguished subsets of D_α. As we shall see later, each C_α, defined differently, 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 first-order term made of (names of) basic computable functions and of basic variables a_i and constant values of level 0. Then the function $\lambda \mathsf{a}_1 \ldots \lambda \mathsf{a}_k . t$ 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 pure functional types α of levels > 1 in the following way.

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

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

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 $d > 1$ (i.e. with some α_i not basic). Then*

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

Hereinafter we shall sometimes call as *C(respectively $\bar{C}$)-functionals* those from C ($\bar{C}$). By a *C-family* we mean any functional $g \in C_{\bar{a} \to \alpha}$ (or any tuple of functionals $g_i \in C_{\bar{a} \to \alpha_i}$) with basic and non-empty $\bar{a}$, and likewise for $\bar{C}$(cf. g_i in the above Definitions). So, we require that a computable functional composed with any computable family give again a computable function.

Correctness of the Definition. In contrast with Definition 2 we have to show that Definition 3 is correct, just that expressions $g_i \bar{\mathsf{a}}$ make sense. To this end it is sufficient to prove, simultaneously with the inductive step of Definition 3, that *the set of all C-functionals is closed under application,* i.e. $fx \in C_\beta$, whenever $x \in C_\alpha$ and $f \in C_{\alpha \to \beta}$, the fact important in itself.

As *induction hypothesis*, we assume that all C-functionals introduced on the previous steps for types with level $< d$ (e.g for g_i), are closed under application.

[1] This Definition and corresponding proofs below mainly follow and generalize [SS 86].

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 constant functional $h = \lambda x.f$ is also C-functional, provided that its level is also $< d$.

Let us show that C-functionals $f \in C_{\alpha_1 \ldots \alpha_n \to b}$ of type level d are also closed under application. So, let $\bar{x} = \bar{u}, \bar{v}$ be all n arguments for f. Then $f\bar{u} = \lambda\bar{v}.f\bar{u}\bar{v}$ with fixed $\bar{u} \in C$ is also C-functional, i.e., as required, $\lambda\bar{\mathsf{a}}.f\bar{u}(\bar{V}\bar{\mathsf{a}})$ is C-function of level 1 for any appropriate C-family $\bar{V}$ (because this is $= \lambda\bar{\mathsf{a}}.f(\bar{U}\bar{\mathsf{a}})(\bar{V}\bar{\mathsf{a}})$ with $\bar{U} = \lambda\bar{\mathsf{b}}.\bar{u}$ also a constant C-family, 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 $\leq d$. $\square$

We shall also use the following *variation* of Definition 3 (with $\bar{a}$ and b basic):

$$C_{\bar{a} \to \bar{\alpha} \to b} = \{f \in C_{\bar{a}} \to C_{\bar{\alpha}} \to C_b \mid \begin{array}{l} \lambda\bar{\mathsf{a}} : \bar{a}.\lambda\bar{\mathsf{b}}.f\bar{\mathsf{a}}(\bar{g}\bar{\mathsf{b}}) \text{ is } C\text{-function} \\ \text{for all } C\text{-families } \bar{g}\bar{\mathsf{b}} : \bar{\alpha} \\ \text{of any new basic variables } \bar{\mathsf{b}}\}. \end{array}$$

Furthermore, *domains C_α comprise an extensional model of typed lambda calculus.* This is because C-functionals are closed under application (thereby, any $C_{\alpha\to\beta}$ proves to be just a set of extensional functions from C_α to C_β, i.e. $C_{\alpha\to\beta} \subseteq C_\alpha \to C_\beta$) and by the following Proposition and its Corollary. The definition of (semantics of a well-typed) applicative term is standard.

Proposition 4 *Let $t = t[\bar{y}]$ be any applicative term of any type all variables of which are in $\bar{y}$. Then, for all C-families $\bar{Y}$ of appropriate types, $\lambda\bar{\mathsf{a}}.t[(\bar{Y}\bar{\mathsf{a}})/\bar{y}] = \lambda\bar{\mathsf{a}}.t[\bar{Y}\bar{\mathsf{a}}]$ (with basic $\bar{\mathsf{a}}$) is also a C-family.*

> *Proof.* The case t is a variable y_i or a constant is trivial. Let t be an application t_1t_2. By induction we may assume that $\lambda\bar{\mathsf{a}}.t_1[\bar{Y}\bar{\mathsf{a}}]$ and $\lambda\bar{\mathsf{a}}.t_2[\bar{Y}\bar{\mathsf{a}}]$ are C-families. Then, for any appropriate C-family $\bar{Z}$, the application $(t_1[\bar{Y}\bar{\mathsf{a}}])(t_2[\bar{Y}\bar{\mathsf{a}}])(\bar{Z}\bar{\mathsf{b}}) = (\lambda\bar{\mathsf{a}}.t_1[\bar{Y}\bar{\mathsf{a}}])\bar{\mathsf{a}}(t_2[\bar{Y}\bar{\mathsf{a}}])(\bar{Z}\bar{\mathsf{b}})$ of a basic type denotes a C-function of basic variables $\bar{\mathsf{a}}$ and appropriate new $\bar{\mathsf{b}}$. This holds because C-families $\bar{\mathsf{a}}$ and $t_2[\bar{Y}\bar{\mathsf{a}}]$ of variables $\bar{\mathsf{a}}$ and $\bar{Z}\bar{\mathsf{b}}$ of variables $\bar{\mathsf{b}}$ are substituted for all the arguments in C-functional $(\lambda\bar{\mathsf{a}}.t_1[\bar{Y}\bar{\mathsf{a}}])$. It follows by the variation of Definition 3 presented above (with $\bar{Z}$ in the role of $\bar{g}$) that $(t_1[\bar{Y}\bar{\mathsf{a}}])(t_2[\bar{Y}\bar{\mathsf{a}}])\bar{z}$ is C-functional of $\bar{\mathsf{a}}, \bar{z}$ and hence $(t_1[\bar{Y}\bar{\mathsf{a}}])(t_2[\bar{Y}\bar{\mathsf{a}}])$ is C-family of $\bar{\mathsf{a}}$. ∎

Corollary 5 (Combinatory Completeness of C) *For any applicative term t of type β with free variables 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, combinators $\mathbf{S} = \lambda xyz.xz(yz)$ and $\mathbf{K} = \lambda xy.x$ of any appropriate type are C-functionals.*

> *Proof.* Let $t\bar{y}$ be 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 $\lambda\bar{x}\bar{y}.t\bar{y} = \lambda\bar{x}.t$, which holds by extensionality of C, completes the proof. ∎

The same considerations prove

Corollary 6 (Combinatory Completeness of $\bar{C}$) *For any applicative term t of type β with free variables x_i of types α_i, the mapping $\lambda\bar{x}.t \in D_{\alpha_1} \to \ldots \to D_{\alpha_n} \to D_\beta$ is an element of $\bar{C}_{\alpha_1\to\ldots\to\alpha_n\to\beta}$. In particular, combinators $\bar{\mathbf{S}} = \lambda xyz.xz(yz)$ and $\bar{\mathbf{K}} = \lambda xy.x$ of any appropriate type are $\bar{C}$-functionals.*

There is a certain reason to study both models C and $\bar{C}$. If we introduce notions of realizability as in [Vor 90] and [Vor 91], then 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 still consider only the pure functional case. In contrast to $\{C_\alpha\}$ 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 $\bar{C}$)

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

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

Evidently, if there exist non-computable functions then, for level 1, this PER is a partial equality and, for levels > 1, this is a proper partial equivalence relation. Also define $\tilde{C}_\alpha = \{x \in \bar{C}_\alpha \mid x \sim_\alpha x\}$.

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 $C_\alpha \cong \bar{C}_\alpha/\sim_\alpha \; (= \tilde{C}_\alpha/\sim_\alpha = D_\alpha/\sim_\alpha)$.

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 also a *partial* mapping from D_α with the domain $\bar{C}_\alpha \subseteq D_\alpha$. Here $\dot{\to}$ is used for partial functions.

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

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 $\bar{a}$ empty) 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 $g \in C_{\bar{b}\to\alpha}$. Then $\lambda\bar{\mathsf{a}}.f_{\bar{\mathsf{a}}} \in C_{\bar{a}\to(\alpha\to\beta)}$ (and conversely).* $\square$

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

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 (for any $g \in \bar{C}_{\alpha\to\beta}$ and $f \in C_{\alpha\to\beta}$)

$$j_{\alpha\to\beta}g \rightleftharpoons j_\beta \circ g \circ i_\alpha : C_\alpha \to C_\beta \text{ and } i_{\alpha\to\beta}f \rightleftharpoons \overline{i_\beta \circ f \circ j_\alpha} : D_\alpha \to D_\beta,$$

with bar denoting any total extension of a partial mapping, 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 $C_{\alpha\to\beta} \subseteq C_\alpha \to C_\beta$ and $\bar{C}_{\alpha\to\beta} \subseteq D_\alpha \to D_\beta$.

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 **S** and **K** of C-model to the corresponding combinators of $\bar{C}$ or D. As for the reverse, we can easily show $j\bar{\mathbf{S}} = \mathbf{S}$ 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 $\tilde{j}_b : D_b \dot{\to} E_b$ be a *partial function onto*, for each basic type. Let $\tilde{j}_\alpha \subseteq D_\alpha \times E_\alpha$ be the corresponding induced logical relation (cf. [Sta 85] and [Mit 90]).

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

Proof. We shall simultaneously construct retraction (right inverse) total 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 (as above)

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

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, $f \circ \tilde{j}_\alpha = \tilde{j}_\beta \circ \tilde{i}_\beta \circ f \circ \tilde{j}_\alpha \subseteq \tilde{j}_\beta \circ \overline{\tilde{i}_\beta \circ f \circ \tilde{j}_\alpha} = \tilde{j}_\beta \circ g$. □

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 $\tilde{j}_\alpha x$ and $\tilde{j}_\alpha y$ 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 a logical PER on the typed combinatory algebra $\{\tilde{E}_\alpha\}$. □

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 both $\tilde{j}_b$ and $\tilde{i}_b$ are identity functions for each 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 Theorem 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 sums in C-version. For simplicity, we postpone product types to Section 6.

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. Remember, that cartesian product of any family of sets $\{S_b \mid b \in B\}$ indexed by the set of all basic types B is just $\prod_b S_b \rightleftharpoons \{f : B \to \bigcup S_b \mid f(b) \in S_b,$ for each $b \in B\}$.

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

$$
\begin{aligned}
C_{\gamma+\delta} \quad &\rightleftharpoons \quad \{f \in \textstyle\prod_b C_{(\gamma\to b)\to(\delta\to b)\to b} \mid \\
&\qquad \exists x \in C_\gamma \forall b \forall g \in C_{\gamma\to b} \forall h \in C_{\delta\to b}(fbgh = gx) \ \textit{or} \\
&\qquad \exists y \in C_\delta \forall b \forall g \in C_{\gamma\to b} \forall h \in C_{\delta\to b}(fbgh = hy)\},
\end{aligned}
$$

$$
\begin{aligned}
C_{\alpha_1\ldots\alpha_n\to b} \quad &\rightleftharpoons \quad \{f \in C_{\alpha_1} \to \ldots \to C_{\alpha_n} \to C_b \mid \\
&\qquad \forall \bar{a} \ \textit{basic and} \ \forall g_i \in C_{\bar{a}\to\alpha_i}, \\
&\qquad \lambda \bar{\mathsf{a}}.f(g_1\bar{\mathsf{a}})\ldots(g_n\bar{\mathsf{a}}) \in C_{\bar{a}\to b}\},
\end{aligned}
$$

$$
\begin{aligned}
C_{\alpha_1\ldots\alpha_n\to\gamma+\delta} \quad &\rightleftharpoons \quad \{f \in C_{\alpha_1} \to \ldots \to C_{\alpha_n} \to C_{\gamma+\delta} \mid \forall b \ \textit{basic}, \\
&\qquad \lambda \bar{x} : \bar{\alpha}.f\bar{x}b \in C_{\bar{\alpha}\to(\gamma\to b)\to(\delta\to b)\to b}\}.
\end{aligned}
$$

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 disjoint union of C_γ and C_δ *factorized* by the following equivalence relation: $x \sim y$ iff x and y are nondistinguishable by any functionals with basic values. More exactly, if e.g. $x, y \in C_\gamma$ then $x \sim y$ iff $gx = gy$ holds for all basic b and $g \in C_{\gamma\to b}$. If $x \in C_\gamma$, $y \in C_\delta$ then $x \sim y$ iff $gx = hy$ for all b and $g \in C_{\gamma\to b}$, $h \in C_{\delta\to b}$.

It is easy to show that $C_{\gamma+\delta}$ is canonically isomorphic to this factorized union of C_γ and C_δ. Moreover, if we additionally assume that *some basic domain D_b contains at least two different elements, say 0 and 1, and that all pairs of different basic elements* a, b *(of the same type) are distinguishable* (by some *unique* corresponding function, like **if** $x = \mathsf{a}$ **then** 0 **else** 1) then the above factorizing proves to be *trivial*. Therefore *under this assumption so defined sum is, in fact, isomorphic to the ordinary set theoretic separated sum!*

In fact, for the following considerations we *need no* such additional assumption on distinguishability of data obects because even in this general case the domain $C_{\gamma+\delta}$ proves to satisfy all formal properties of co-product.

Correctness of Definition 14 (whose well-foundedness is obtained in *two* steps, for the last clause), the formulation and proof of *combinatory completeness* of domains $\{C_\tau\}$ are 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 a variable 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*

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

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

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

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

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 $[F\circ\mathrm{inl}, F\circ\mathrm{inr}]\circ\mathrm{inr} = F\circ\mathrm{inr}$. ■

6 The General Case (with Product Types)

To account product types, let us add the following two items to Definition 14:

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

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

where $\langle x,y\rangle_1 = x$, $\langle x,y\rangle_2 = y$. All considerations are now the same as above or straightforward (with some additional complication of the construct [-,-] definition) 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 *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, 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.

[LM 84] G.Longo and E.Moggi. The hereditary partial effective functionals and recursion theory in higher types. *The Journal of Symbolic Logic*, 49(4) 1319-1332, 1984.

[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.

[Rog 67] H.Jr.Rogers. *Theory of recursive functions and effective computability*. McGraw-Hill, New York, 1967

[Saz 90] V.Yu. Sazonov. An equivalence between polynomial constructivity of Markov's principle and the equality $P = NP$. 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, Varna, Bulgaria*, pages 351–360. Plenum Press, NY, 1990. (A short English version.)

[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*, pages 16–34. Novosibirsk, 1986.

[Sta 85] R. Statman. Logical relations and the typed λ-calculus. *Information and Control*, 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.