UPMAIL Technical Report (draft)
November 11, 1998
Time 11:25
ISSN 1100-0686

A Construction of typed lambda models related to feasible computability (draft of an extended version)

Vladimir Sazonov

Research Center of Artificial Intelligence
Program Systems Institute
Pereslavl-Zalessky 152140
Russia
sazonov@logic.botik.ru

Andrei Voronkov

Computing Science Department
Uppsala University
Box 311
S 751 05 Uppsala, Sweden
voronkov@csd.uu.se

Abstract

We define an approach to the notion of *computable functionals* in an abstract setting not restricted to Turing or, say, polynomial computability. We assume to start with 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 preserving basic functions. We call such functionals *computable* or, more technically and adequately speaking, *substitutable*. (Similarly, in D.Scott's domains the *continuous* functionals are a far-reaching abstraction of the computable ones.) Our results are applicable to quite arbitrary (complexity) classes of functions, satisfying a very general Assumption 2.6.[*][†]

[†]Vladimir Sazononv was partially supported by a grant from the Swedish Royal Academy of Sciences, by INTAS (project 93-0845), and by the Russian Foundation of Basic Research (project 96-01-017-17). Andrei Voronkov is partially supported by TFR grants.

Contents

1 Introduction **1**

2 Preliminaries **3**

3 Purely Functional Case **4**
3.1 Maximality of the lambda model $\mathbf{C}$ 7

4 On the Connection Between C and $\bar{\mathbf{C}}$ **7**
4.1 When $\tilde{\mathbf{C}} = \bar{\mathbf{C}}$ 8
4.2 A Computable Retraction Embedding of $\mathbf{C}$ into $\bar{\mathbf{C}}$ 9
4.3 A Connection Between the full Type Structure $\mathbf{D}$ and any Other Typed Lambda Model E 9
4.4 When E is $\mathbf{C}$ 10

5 The Case of Sums **11**

6 The General Case (with Product Types) **12**

Bibliography **13**

1 Introduction

The problem raised up in this paper is related to the general notion of computations in various programming paradigms over various 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 some set $\mathcal{F}$ of functions (predicates, computations) 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 to obtain functions in $\mathcal{F}$. In this paper we only study extensional aspects of such a definition, i.e. we only consider functions defined as by input-output relations. 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.

There are additional motivations for studying this problem. Very similar definitions (essentially for pure functional case) were used in the paper of the first author [Sazonov and Sviridenko 1986], defining a denotational semantics of the language of Σ-expressions. In that paper a finite type structure was constructed on the basis of some given in advance class $\mathcal{P}$ of "sigma-like"[3] predicates over some universe of basic objects (in fact, of sets). The point of [Sazonov and Sviridenko 1986] 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

[3] In this context Σ-like can be considered as an abstract analogue of "computable".

also assume that only some given *basic* functions are admissible (for whatever purpose) and look for a corresponding finite type structure.

As earlier closely related sources we mention the Banach-Mazur functionals (of level 2; see [Rogers 1967]) based on the class of all total recursive functions over natural numbers and also Longo and Moggi's hereditary partial effective functionals [Longo and Moggi 1984] 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 the notion of *realizability* [Kleene 1952] for some complexity classes, like various polynomial realizability approaches considered in [Buss 1986, Cook and Urquhart 1989, Sazonov 1990]. Realizability allows one to extract programs (computable functions) from constructive proofs using higher type functionals. Various modifications of realizability [**?**] yield various classes of functions which can be extracted from proofs. Most this modifications are defined for natural numbers or some other concrete domains, like (a suitable notion of) real numbers. A more abstract approach to realizability used in [Voronkov 1990] defines a realizability notion for arbitrary datatypes whose elements can be enumerated by natural numbers. However, that notion of realizability, depends on a *numeration* of a datatype, which corresponds to its particular representation.

Assume that we want to define a more abstract notion of realizability that would allow only a certain class of functions to be extracted from proofs. Then (like [Buss 1986]) we shall have to define the corresponding class of higher-order functionals. In this paper the definition of computable (substitutable) functionals of higher types will be raised to a 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. Various notions of realizability based on various lambda models have been considered in [Mitchell 1986, Mitchell and O'Donnel 1986, Carboni, Freyd and Scedrov 1988].

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 [Cook and Urquhart 1989] based on bounded primitive recursion *are* computable in our $\bar{\mathbf{C}}$-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 $\mathbf{D}_\alpha$ ($\bar{\mathbf{C}}$-functionals of Definition 3.1). The other way ($\mathbf{C}$-functionals of Definition 3.2) 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 $\mathbf{C}$ to allow sum and product types.

The authors are grateful to G.Longo for taking our attention to an essential reference [Longo and Moggi 1984].

2 Preliminaries

We assume to have a set of *basic types,* to be denoted by $a, b, c, \ldots$ together with associated non-empty sets $\mathbf{D}_a, \mathbf{D}_b, \mathbf{D}_c, \ldots$

DEFINITION 2.1 (Types)

1. Each basic type is a type;

2. If α, β are types then $\alpha \to \beta$, $\alpha \times \beta$ and $\alpha + \beta$ are types.

Types constructed using only $\to$ are called *purely functional.*

Arbitrary types will be denoted by $\alpha, \beta, \gamma, \ldots$ 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 simply as $\alpha_1, \alpha_2, \ldots, \alpha_n \to \beta$. If $D, E, \ldots, F$ are any sets then $D \to E$ denotes the set of *all* mappings from D to E, and likewise for $D \to \ldots \to E \to F = D \ldots E \to F$. Notation $\rightleftharpoons$ means "equal by definition". $D + E$ denotes the *separated sum* of sets D, E.

DEFINITION 2.2 (Sets $\mathbf{D}_\tau$) We shall extend $\mathbf{D}_\tau$ to all non-basic types as follows

$$\begin{array}{rcl} \mathbf{D}_{\alpha\to\beta} & \rightleftharpoons & \mathbf{D}_\alpha \to \mathbf{D}_\beta \\ \mathbf{D}_{\alpha\times\beta} & \rightleftharpoons & \mathbf{D}_\alpha \times \mathbf{D}_\beta \\ \mathbf{D}_{\alpha+\beta} & \rightleftharpoons & \mathbf{D}_\alpha + \mathbf{D}_\beta \end{array}$$

Denotation like $\mathbf{C}_{\bar{\alpha}} \to \mathbf{C}_\beta$ is an abbreviation for $\mathbf{C}_{\alpha_1} \to \ldots \to \mathbf{C}_{\alpha_n} \to \mathbf{C}_\beta$.

DEFINITION 2.3 (Level)

1. level(any basic type) is 0;

2. $\mathrm{level}(\alpha \to \beta) \rightleftharpoons \max(1 + \mathrm{level}(\alpha), \mathrm{level}(\beta))$;

3. $\mathrm{level}(\alpha \times \beta) \rightleftharpoons \max(\mathrm{level}(\alpha), \mathrm{level}(\beta))$;

4. $\mathrm{level}(\alpha + \beta) \rightleftharpoons 2 + \max(\mathrm{level}(\alpha), \mathrm{level}(\beta))$.

Elements of $\mathbf{D}_\alpha$ (and of the sets $\bar{\mathbf{C}}_\alpha$ and $\mathbf{C}_\alpha$ defined below) are called *functionals* of the type α (and of the corresponding level).

Our aim can be informally explained as follows. Elements of sets $\mathbf{D}_{a_1,\ldots,a_n\to b}$, i.e. of all level 1 purely functional types can be identified with n-place functions from $\mathbf{D}_{a_1} \times \ldots \times \mathbf{D}_{a_n}$ to $\mathbf{D}_b$. Assume that some elements of $\mathbf{D}_{a_1,\ldots,a_n\to b}$ are fixed and called *computable.* Our aim is to define the corresponding *computable functionals* of all finite types preserving the given class of computable functions. An example might be to construct polynomial time computable functionals from functions of natural numbers computable in polynomial time.

We shall define two versions $\bar{\mathbf{C}}$ and $\mathbf{C}$ of computable functionals. Functionals in $\bar{\mathbf{C}}_\alpha$ are distinguished elements of $\mathbf{D}_\alpha$. Sets $\mathbf{C}_\alpha$ are defined differently and prove to be factor sets of $\bar{\mathbf{C}}_\alpha$. We define $\bar{\mathbf{C}}_a \rightleftharpoons \mathbf{C}_a \rightleftharpoons \mathbf{D}_a$ for all basic types a and assume $\bar{\mathbf{C}}_\alpha = \mathbf{C}_\alpha \subseteq \mathbf{D}_\alpha$ for the types of level 1.

DEFINITION 2.4 *Basic functions* are elements of $\mathbf{D}_\alpha$ of level 1. *Basic computable functions* are elements of $\bar{\mathbf{C}}_\alpha$ of level 1.

For representing functionals we shall use a language with an infinite supply of *constants* of every type α and an infinite supply of *variables* of every type α. Variables of basic types will be denoted by $\mathsf{a}, \mathsf{b}, \mathsf{c} \ldots$, and variables of arbitrary types by $x, y, z \ldots$

DEFINITION 2.5 Let C be any set of constants and V be any set of variables. *Applicative terms* with constants in C and variables in V (or simply applicative terms *over* C, V) are defined inductively as follows:

1. Every $c \in C$ or $v \in V$ of a type α is an applicative term of the type α over C, V
2. If t_1 is an applicative term of the type $\alpha \to \beta$ over C, V and t_2 is an applicative term of the type α over C, V then $(t_1 t_2)$ is an applicative term of the type β over C, V.

λ-terms over C, V are defined similarily by replacing "applicative term" by "λ-term" and adding

3. If t is a λ-term of the type β over C, V and $x \in V$ is of the type α then $\lambda x.t$ is a λ-term of the type $\alpha \to \beta$ over C, V.

The set of *free variables* of a λ-term is defined as usual.

As usual, we omit parentheses in applicative terms and λ-terms whenever possible, denoting by $t_1 t_2 \ldots t_n$ the applicative term $(\ldots(t_1 t_2) \ldots t_n)$. When interpretations of constants in an applicative term or a λ-term are fixed, we can often give a natural interpretation to the term itself. We shall usually mix applicative terms and their intended interpretations when the interpretation is clear from the context and it does not cause ambiguity.

We assume that the sets $\bar{\mathbf{C}}_\alpha$ for basic α satisfy the following

ASSUMPTION 2.6 *Let C be a set of level 1 constants interpreted by computable basic functions, $V \rightleftharpoons \{\mathsf{a}_1, \ldots, \mathsf{a}_k\}$ be a set of variables of basic types $a_1, \ldots, a_k$ and t be an applicative term over C, V of a basic type b. Then $\lambda \mathsf{a}_1 \ldots \lambda \mathsf{a}_k.t \in \bar{\mathbf{C}}_{a_1, \ldots, a_k \to b}$.*

In fact, Assumption 2.6 means that an arbitrary applicative term composed of computable basic functions gives a computable basic function.

EXAMPLE 2.7 Let there be only one basic sort ω with $\mathbf{D}_\omega \rightleftharpoons N$ (the set of natural numbers). Assume that $\bar{\mathbf{C}}_\alpha$ consists of polynomial time computable functions. Then Assumption 2.6 is obviously satisfied.

Other well-know examples are natural numbers with primitive recursive or total recursive functions.

3 Purely Functional Case

We extend the definition of domains $\bar{\mathbf{C}}_\alpha \subseteq \mathbf{D}_\alpha$ of computable functionals for pure functional types α of levels > 1 in the following way.

DEFINITION 3.1 Let τ be a type $\alpha_1, \ldots, \alpha_n \to b$ of level $d > 1$ (i.e. with some α_i not basic). Then

$$\bar{\mathbf{C}}_\tau \rightleftharpoons \{f \in \mathbf{D}_{\alpha_1} \to \ldots \to \mathbf{D}_{\alpha_n} \to \mathbf{D}_b \mid \begin{array}{l} \text{for all basic types } a_1, \ldots, a_k \\ \text{and every } g_i \in \bar{\mathbf{C}}_{a_1,\ldots,a_k \to \alpha_i}, \\ \lambda \bar{\mathsf{a}}.f(g_1\bar{\mathsf{a}}) \ldots (g_n\bar{\mathsf{a}}) \in \bar{\mathbf{C}}_{a_1,\ldots,a_n \to b}\} \end{array}$$

Note that this definition is well-founded since the levels of all functional g_i used in the definition is strictly less than the level of f.

The sets $\mathbf{C}_\alpha$ are defined similarly:

DEFINITION 3.2 [4] Let τ be a type $\alpha_1, \ldots, \alpha_n \to b$ of level $d > 1$ (i.e. with some α_i not basic). Then

$$\mathbf{C}_\tau \rightleftharpoons \{f \in \mathbf{C}_{\alpha_1} \to \ldots \to \mathbf{C}_{\alpha_n} \to \mathbf{C}_b \mid \begin{array}{l} \text{for all basic types } a_1, \ldots, a_k \\ \text{and every } g_i \in \mathbf{C}_{a_1,\ldots,a_k \to \alpha_i}, \\ \lambda \bar{\mathsf{a}}.f(g_1\bar{\mathsf{a}}) \ldots (g_n\bar{\mathsf{a}}) \in \mathbf{C}_{a_1,\ldots,a_n \to b}\} \end{array}$$

Before proving the correctness of this definition, we introduce a few notions.

DEFINITION 3.3 $\bar{\mathbf{C}}$*-functionals (respectively* $\mathbf{C}$*)-functionals* are elements of $\bar{\mathbf{C}}$ (respectively $\mathbf{C}$).

A class C of functionals (where C can be any subset of $\mathbf{D}_\beta$ or $\mathbf{C}_\beta$) *admits dummy arguments* of a type α iff for every functional $f \in C$, every variables $x_1, \ldots, x_n$ of appropriate types and every variable y of the type α we have $\lambda x_1 \ldots \lambda x_n \lambda y.fx_1 \ldots x_n \in C$.

By a $\bar{\mathsf{a}}$-family of $\bar{\mathbf{C}}$*-functionals* of a type α we mean either any functional $g \in \bar{\mathbf{C}}_{\bar{a}\to\alpha}$ or any tuple of functionals $g_i \in \bar{\mathbf{C}}_{\bar{a}\to\alpha_i})$ with basic and non-empty $\bar{a}$ (and likewise for $\mathbf{C}$-functionals).

In fact, Definitions 3.1 and 3.2 can be stated as statements about $\bar{\mathsf{a}}$-families g_i.

Hereinafter instead of $f \in \mathbf{C}_\alpha$ we shall usually simply write $f : \alpha$.

In order to be certain that Definition 3.2 is correct we have to prove that $g_i\bar{\mathsf{a}}$ used in the definition makes sense. The proof is based on the following lemma that should be proved in parallel with the definition of $\mathbf{C}$-functionals of level d:

LEMMA 3.4 *For every level $d \geq 1$ we have*

1. *The set of* $\mathbf{C}$*-functionals of level d is closed under application, i.e. $fx : \beta$, whenever $x : \alpha$ and $f : \alpha \to \beta$;*

2. *The set of* $\mathbf{C}$*-functionals of level d admits dummy arguments of levels $\leq d$.*

PROOF. By induction on d. For $d = 1$ both statements are immediate from Assumption 2.6.

Assume now that $d > 1$ and for all levels $< d$ the statement holds.

1. Suppose $f : \alpha_1, \alpha_2, \ldots, \alpha_n \to b$ and $h : \alpha_1$. Let $\bar{a}$ be a tuple of basic types and $\bar{\mathsf{a}}$ be a corresponding tuple of variables. By induction hypothesis we can add dummy arguments to h. Hence, $\lambda\bar{\mathsf{a}}.h : \bar{a} \to \alpha_1$. Take any $g_i : \bar{a} \to \alpha_1$, for $i \geq 2$. Since $f : \alpha_1, \alpha_2, \ldots, \alpha_n \to b$, we have $f((\lambda\bar{\mathsf{a}}.h)\bar{\mathsf{a}})(g_2\bar{\mathsf{a}})(g_n\bar{\mathsf{a}}) : \bar{a} \to b$. Hence, $fh(g_2\bar{\mathsf{a}})(g_n\bar{\mathsf{a}}) : \bar{a} \to b$. Since g_i were arbitrary, $fh : \alpha_2, \ldots, \alpha_n \to b$.

[4] This Definition and corresponding proofs below mainly follow and generalize [Sazonov and Sviridenko 1986].

2. Suppose $f : \alpha_1, \ldots, \alpha_n \to b$ and x is a variable of a type β of a level $< d$. We have to prove $\lambda x_1 \ldots x_m x. f x_1 \ldots x_m x : \alpha_1, \ldots, \alpha_m, \beta, \alpha_{m+1}, \alpha_n \to b$. Denote this functional by h. Indeed, take arbitrary tuple of basic types $\bar{a}$, a corresponding tuple of variables $\bar{\mathsf{a}}$ and arbitrary $g : \bar{a} \to \beta$, $g_i : \bar{a} \to b$. Then by the definition of h we have $\lambda \bar{\mathsf{a}}.h(g_1\bar{\mathsf{a}}) \ldots (g_m\bar{\mathsf{a}})(g\bar{\mathsf{a}})(g_{m+1}\bar{\mathsf{a}}) \ldots (g_n\bar{\mathsf{a}}) = \lambda \bar{\mathsf{a}}.h(g_1\bar{\mathsf{a}})(g_n\bar{\mathsf{a}}) : \bar{a} \to b$. Since $g, g_1, \ldots, g_n$ were arbitrary, $h : \alpha_1, \ldots, \alpha_m, \beta, \alpha_{m+1}, \alpha_n \to b$.

□

We shall also use the following *variation* of Definition 3.2 (with $\bar{a}$ and b basic):

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

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

PROPOSITION 3.5 *Let $t = t[\bar{y}]$ be any applicative term of any type all variables of which are in $\bar{y}$. Then, for all $\mathbf{C}$-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 $\mathbf{C}$-family.*

PROOF. The case t is a variable y_i or a constant is trivial. Let t be an application $t_1 t_2$. 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 $\mathbf{C}$-families. Then, for any appropriate $\mathbf{C}$-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 $\mathbf{C}$-function of basic variables $\bar{\mathsf{a}}$ and appropriate new $\bar{\mathsf{b}}$. This holds because $\mathbf{C}$-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 $\mathbf{C}$-functional $(\lambda\bar{\mathsf{a}}.t_1[\bar{Y}\bar{\mathsf{a}}])$. It follows by the variation of Definition 3.2 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 $\mathbf{C}$-functional of $\bar{\mathsf{a}}, \bar{z}$ and hence $(t_1[\bar{Y}\bar{\mathsf{a}}])(t_2[\bar{Y}\bar{\mathsf{a}}])$ is $\mathbf{C}$-family of $\bar{\mathsf{a}}$. □

COROLLARY 3.6 (Combinatory Completeness of $\mathbf{C}$) *For any applicative term t of type β with free variables x_i of types α_i, the mapping $\lambda\bar{x}.t \in \mathbf{C}_{\alpha_1} \to \ldots \to \mathbf{C}_{\alpha_n} \to \mathbf{C}_\beta$ is an element of $\mathbf{C}_{\alpha_1\to\ldots\to\alpha_n\to\beta}$. In particular, the combinators $\mathbf{S} = \lambda xyz.xz(yz)$ and $\mathbf{K} = \lambda xy.x$ of any appropriate type are $\mathbf{C}$-functionals.*

PROOF. Let $t\bar{y}$ be of a basic type b. Then $\lambda\bar{x}\bar{y}.t\bar{y}$ is a $\mathbf{C}$-functional by Definition 3.2 and Proposition 3.5. Equality $\lambda\bar{x}\bar{y}.t\bar{y} = \lambda\bar{x}.t$, which holds by extensionality of $\mathbf{C}$, completes the proof.

□

The same considerations prove

COROLLARY 3.7 (Combinatory Completeness of $\bar{\mathbf{C}}$) *For any applicative term t of type β with free variables x_i of types α_i, the mapping $\lambda\bar{x}.t \in \mathbf{D}_{\alpha_1} \to \ldots \to \mathbf{D}_{\alpha_n} \to \mathbf{D}_\beta$ is an element of $\bar{\mathbf{C}}_{\alpha_1\to\ldots\to\alpha_n\to\beta}$. In particular, combinators $\bar{\mathbf{S}} = \lambda xyz.xz(yz)$ and $\bar{\mathbf{K}} = \lambda xy.x$ of any appropriate type are $\bar{\mathbf{C}}$-functionals.*

There is a certain reason to study both models $\mathbf{C}$ and $\bar{\mathbf{C}}$. If we introduce notions of realizability as in [Voronkov 1990] and [Voronkov 1991], then the first way corresponds to the model $\bar{\mathbf{C}}$, whilst the second — to $\mathbf{C}$. 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 [Kleene 1952].

3.1 Maximality of the lambda model C

Let E_α be any pure functional lambda model, i.e. a typed combinatory algebra satisfying $E_{\alpha\leftarrow\beta} \subseteq E_\alpha \leftarrow E_\beta$. Define partial ordering on lambda models by taking

$$E \leq E' \rightleftharpoons \forall\pi[\forall\tau \prec \pi(E_\tau = E_{\tau'}) \Leftarrow E_\pi \subseteq E'_\pi]$$

where $\tau \prec \pi \rightleftharpoons \mathbf{level}(\tau) < \mathbf{level}(\pi)$.

Evidently, $E \leq E' \Leftarrow \forall b(E_b \subseteq E'_b)$. Conversely, $\forall b(E_b \subseteq E'_b)\&\exists b(E_b \neq E'_b) \Leftarrow E \leq E'$.

Reflexivity of $\leq$ is trivial.

To show transitivity $E \leq E' \leq E'' \Leftarrow E \leq E''$ let us suppose $\forall\tau \prec \pi(E_\tau = E''_\tau)$. Then by induction on the level of $\tau \prec \pi$ we get $E_\tau \subseteq E'_\tau \subseteq E''_\tau$ and therefore $E_\tau = E'_\tau = E''_\tau$. It follows that $E_\pi \subseteq E'_\pi \subseteq E''_\pi$ and hence $E_\pi \subseteq E''_\pi$ as required.

Finally, $\leq$ is antisymmetric: $E \leq E'$ and $E' \leq E$ give $E_\pi \subseteq E'_\pi$ and $E'_\pi \subseteq E_\pi$ and therefore $E_\pi = E'_\pi$, a contradiction.

THEOREM 3.8 $\mathbf{C}$ *is the largest lambda model containing given level one functions.*

PROOF. Let us suppose that $\forall b(E_b = \mathbf{C}_b)$ and for all level 1 types $E_\tau \subseteq \mathbf{C}_\tau$ and show $E \leq \mathbf{C}$. We need to suppose additionally that $\forall\tau \prec \pi(E_\tau = \mathbf{C}_\tau)$ and show that $E_\pi = \mathbf{C}_\pi$. However, this is evident by the definition of $\mathbf{C}$ and by the fact that E, as a lambda model, is closed under compositions of functionals.

□

4 On the Connection Between C and $\bar{\mathbf{C}}$

In this section we still consider only the pure functional case. In contrast to $\{\mathbf{C}_\alpha\}$ the model $\{\bar{\mathbf{C}}_\alpha\}$ is not an extensional lambda model (it is a combinatory algebra). It contains "extra elements" and will be factorized.

Introduce on each $\bar{\mathbf{C}}_\alpha$ a *logical* partial equivalence relation (PER) $\sim_\alpha$ which is induced by the equality relation on basic domains (see [Mitchell 1990]). This means that (for all elements considered in $\bar{\mathbf{C}}$)

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

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

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 $\tilde{\bar{\mathbf{C}}}_\alpha = \{x \in \bar{\mathbf{C}}_\alpha \mid x \sim_\alpha x\}$.

According to [Mitchell 1990] $\bar{\mathbf{C}}_\alpha/\sim_\alpha = \tilde{\bar{\mathbf{C}}}_\alpha/\sim_\alpha$ is an (extensional) lambda model. Moreover, after considerations of the following subsections, we shall prove

THEOREM 4.1 $\mathbf{C}_\alpha \cong \bar{\mathbf{C}}_\alpha/\sim_\alpha \; (= \tilde{\bar{\mathbf{C}}}_\alpha/\sim_\alpha = \mathbf{D}_\alpha/\sim_\alpha)$.

4.1 When $\tilde{\mathbf{C}} = \bar{\mathbf{C}}$

Evidently, if there exists non-computable functions then, for levels ≤ 1, this PER is the identity relation, and for level 2 this is a proper equivalence relation. Note, that typically $\tilde{\mathbf{C}} = \bar{\mathbf{C}}$.

A function(al) $\mathcal{F} : \bar{\alpha} \leftarrow d$ is called *A-valued* if the range of $\mathcal{F}$ in the basic type d is the set $A \subseteq D_d$.

Let us consider the following additional *assumption* on the class of computable functions of level 1:

> (*) If there exists non-computable function (of level 1) then for every two elements $d_1 \neq d_2$ of the same basic type there exists a non-computable $\{d_1, d_2\}$-valued function.

To satisfy the conclusion of condition (*) it is sufficient to require that

1. at least two basic d_1 and d_2 and one non-computable $\{d_1, d_2\}$-valued function must exist and
2. given any basic elements $\mathsf{a}_1 \neq \mathsf{a}_2 : a$ and $\mathsf{b}_1 \neq \mathsf{b}_2 : b$ of any basic types, there exists a computable function $f : a \leftarrow b$ such that $f\mathsf{a}_i = \mathsf{b}_i$, $i = 1, 2$.

Proposition 4.2 *(*) is equivalent to $\tilde{\mathbf{C}} = \bar{\mathbf{C}}$.*

Proof. ($\Leftarrow$): The case if *all* level 1 functions are computable is trivial because it follows $D = \mathbf{C} = \bar{\mathbf{C}} = \tilde{\mathbf{C}}$. In the opposite case we may use the conclusion of (*). Let us prove that $\mathcal{F} \sim \mathcal{F}$, say, for any $\bar{\mathbf{C}}$-functional $\mathcal{F} : \alpha \leftarrow d$, of only one argument. This means that $x_1 \sim x_2 : \alpha$ implies $\mathsf{d}_1 \rightleftharpoons \mathcal{F}x_1 = \mathcal{F}x_2 \rightleftharpoons \mathsf{d}_2$.

Let us suppose the contrary for some F of a minimal level, i.e. that $x_1 \sim x_2$, but $\mathsf{d}_1 \neq \mathsf{d}_2$. Let $h : \bar{b} \leftarrow d$ be any non-computable $\{d_1, d_2\}$-valued function. Now define a *computable* family $X : \bar{b} \leftarrow \alpha$ by

$$X\bar{\mathsf{b}} \rightleftharpoons x_1 \text{ if } h\bar{\mathsf{b}} = \mathsf{d}_1 \text{ and } X\bar{\mathsf{b}} \rightleftharpoons x_2 \text{ if } h\bar{\mathsf{b}} = \mathsf{d}_2.$$

It is computable because for any other computable family $\bar{Y}\bar{\mathsf{c}}$ of basic $\bar{\mathsf{c}}$ such that $X\bar{\mathsf{b}}(\bar{Y}\bar{\mathsf{c}})$ is of a basic type this composition is computable level 1 function on $\bar{\mathsf{b}}, \bar{\mathsf{c}}$:

$$X\bar{\mathsf{b}}(\bar{Y}\bar{\mathsf{c}}) = x_1(\bar{Y}\bar{\mathsf{c}}) = x_2(\bar{Y}\bar{\mathsf{c}}) \text{ by } x_1 \sim x_2 \text{ and } x_1(\bar{Y}\bar{\mathsf{c}}) \sim x_2(\bar{Y}\bar{\mathsf{c}})$$

and because $\bar{Y}\bar{\mathsf{c}}$ has a lower level than $\mathcal{F}$. Here $\bar{\mathsf{b}}$ prove to be dummy arguments. Then we have $\mathcal{F}(X\bar{\mathsf{b}}) = h\bar{\mathsf{b}}$ by considering the two possible cases

1. $h\bar{\mathsf{b}} = \mathsf{d}_1$. Then $X\bar{\mathsf{b}} \rightleftharpoons x_2$ and $\mathcal{F}(X\bar{\mathsf{b}}) = fx_2 = \mathsf{d}_1 = h\bar{\mathsf{b}}$.
2. $h\bar{\mathsf{b}} = \mathsf{d}_2$. This is analogous to the case 1.

We have a contradiction that $\mathcal{F}(X\bar{\mathsf{b}}) = h\bar{\mathsf{b}}$ is non-computable function of $\bar{\mathsf{b}}$ despite the computability of $\mathcal{F}$ and X.

($\Leftarrow$): Let us suppose that (*) fails and prove $\tilde{\mathbf{C}} \neq \bar{\mathbf{C}}$.

The negation of (*) means that

(i) there exists a non-computable function $f : \bar{a} \leftarrow b$ and

(ii) for some $d_1 \neq d_2$ of the same basic type all $\{d_1, d_2\}$-valued functions are computable.

Consider any computable functional $G_1 : (\bar{a} \leftarrow b) \leftarrow d$. Let G_2 be the same as G_1, except that $G_1 f \neq G_2 f$. Then G_2 is evidently also computable and $G_1 \sim G_2$ (because $\sim_{\bar{a} \leftarrow b}$ coincides with the equality relation on $\bar{\mathbf{C}}_{\bar{a} \leftarrow b}$). Define a functional $\mathcal{F}$ by letting $\mathcal{F}G_1 \rightleftharpoons d_1$, $\mathcal{F}G_2 \rightleftharpoons d_2$ and $\mathcal{F}G \rightleftharpoons d_1$ or d_2 anyway on all other arguments G. Then $\mathcal{F}$ is also computable by (ii). We have $G_1 \sim G_2$ but $\mathcal{F}G_1 = d_1 \not\sim d_2 = \mathcal{F}G_2$ what means that $\mathcal{F} \not\sim \mathcal{F}$. It follows that $\mathcal{F} \in \bar{\mathbf{C}} \setminus \tilde{\mathbf{C}}$ and $\tilde{\mathbf{C}} \neq \bar{\mathbf{C}}$, as required. □

4.2 A Computable Retraction Embedding of C into $\bar{\mathbf{C}}$

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

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

LEMMA 4.3 *Suppose* $f_{\bar{\mathsf{a}}} : \mathbf{C}_\alpha \to \mathbf{C}_\beta$ *with* $\bar{\mathsf{a}} \in \mathbf{C}_{\bar{a}}$ *is a family of functions (or just one function, for* $\bar{a}$ *empty) such that* $\lambda\bar{\mathsf{a}}\bar{\mathsf{b}}.f_{\bar{\mathsf{a}}}(g(\bar{\mathsf{b}})) \in \mathbf{C}_{\bar{a},\bar{b}\to\beta} \subseteq \mathbf{C}_{\bar{a},\bar{b}} \to \mathbf{C}_\beta$ *for each* $g \in \mathbf{C}_{\bar{b}\to\alpha}$. *Then* $\lambda\bar{\mathsf{a}}.f_{\bar{\mathsf{a}}} \in \mathbf{C}_{\bar{a}\to(\alpha\to\beta)}$ *(and conversely).* □

LEMMA 4.4 *Suppose* $f_{\bar{\mathsf{a}}} : \mathbf{D}_\alpha \to \mathbf{D}_\beta$ *with* $\bar{\mathsf{a}} \in \mathbf{D}_{\bar{a}}$ *is a family of functions such that* $\lambda\bar{\mathsf{a}}\bar{\mathsf{b}}.f_{\bar{\mathsf{a}}}(g(\bar{\mathsf{b}})) \in \bar{\mathbf{C}}_{\bar{a},\bar{b}\to\beta} \subseteq \mathbf{D}_{\bar{a},\bar{b}\to\beta}$ *for each* $g \in \bar{\mathbf{C}}_{\bar{b}\to\alpha}$. *Then* $\lambda\bar{\mathsf{a}}.f_{\bar{\mathsf{a}}} \in \bar{\mathbf{C}}_{\bar{a}\to(\alpha\to\beta)}$ *(and conversely).* □

Let i_b and j_b for basic types b be (obviously) computable identity mappings between $\mathbf{C}_b$ and $\mathbf{D}_b = \bar{\mathbf{C}}_b = \mathbf{C}_b$. 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 $g \in \bar{\mathbf{C}}_{\alpha\to\beta}$ and $f \in \mathbf{C}_{\alpha\to\beta}$)

$$j_{\alpha\to\beta}g \rightleftharpoons j_\beta \circ g \circ i_\alpha : \mathbf{C}_\alpha \to \mathbf{C}_\beta \text{ and } i_{\alpha\to\beta}f \rightleftharpoons \overline{i_\beta \circ f \circ j_\alpha} : \mathbf{D}_\alpha \to \mathbf{D}_\beta,$$

with bar denoting any total extension of a partial mapping, so that $i_{\alpha\to\beta}f \in \mathbf{D}_{\alpha\to\beta}$. Then Lemmas 4.3 and 4.4 imply that $i_{\alpha\to\beta}$ and $j_{\alpha\to\beta}$ also form a computable retraction pair between $\mathbf{C}_{\alpha\to\beta} \subseteq \mathbf{C}_\alpha \to \mathbf{C}_\beta$ and $\bar{\mathbf{C}}_{\alpha\to\beta} \subseteq \mathbf{D}_\alpha \to \mathbf{D}_\beta$.

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 $\mathbf{C}$ may be considered as an applicative subalgebra of the (non-extensional) applicative combinatory algebra $\bar{\mathbf{C}}$. However, such embedding of $\mathbf{C}$ into $\bar{\mathbf{C}}$ 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 $\mathbf{C}$-model to the corresponding combinators of $\bar{\mathbf{C}}$ or $\mathbf{D}$. As for the reverse, we can easily show $j\bar{\mathbf{S}} = \mathbf{S}$ and $j\bar{\mathbf{K}} = \mathbf{K}$ using the equality $jgx_1 \dots x_n = j(g(ix_1)\dots(ix_n))$, identity (1), and the extensionality of $\mathbf{C}$.

4.3 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 $\tilde{j}_b : \mathbf{D}_b \dot{\to} E_b$ be a *partial function onto*, for each basic type. Let $\tilde{j}_\alpha \subseteq \mathbf{D}_\alpha \times E_\alpha$ be the corresponding induced logical relation (cf. [Statman 1985] and [Mitchell 1990]).

PROPOSITION 4.5 *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 [Mitchell 1990] 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 $f\in\mathbf{D}_{\alpha\to\beta}\subseteq\mathbf{D}_\alpha\to\mathbf{D}_\beta$ 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 $\mathbf{D}$ and E. In particular, $\tilde{j}$ is defined when applied to combinators.

Definition 4.6 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 $\mathbf{D}_\alpha$ and $\tilde{E}_\alpha \rightleftharpoons \mathrm{dom}(\tilde{j}_\alpha) = \{x\in\mathbf{D}_\alpha \mid x\sim^E_\alpha x\}$.

Proposition 4.7

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 $\{\mathbf{D}_\alpha\}$ (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 4.8 *$E\cong\mathbf{D}/\sim^E = \tilde{E}/\sim^E$.*

Proof. Follows from Proposition 4.7 (1,2). □

4.4 When E is $\mathbf{C}$

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 $E_\alpha = \mathbf{C}_\alpha$ with

$$\tilde{j}_\alpha \subseteq j_\alpha : \mathbf{D}_\alpha \dot{\to} \mathbf{C}_\alpha \text{ and } \tilde{i}_\alpha = i_\alpha : \mathbf{C}_\alpha \to \mathbf{D}_\alpha$$

for any given i_α and j_α from Subsection 4.2 for which $\mathrm{dom}(j_\alpha) = \bar{\mathbf{C}}_\alpha$ (and $\mathrm{range}(i_\alpha)\subseteq\bar{\mathbf{C}}_\alpha$) and therefore $\mathrm{dom}(\tilde{j}_\alpha) = \tilde{\bar{\mathbf{C}}}_\alpha \subseteq \bar{\mathbf{C}}_\alpha$. Here $\tilde{\bar{\mathbf{C}}}_\alpha$ is used in the sense of Definition 4.6. In fact, it follows from this Subsection that these domains coincide with those from Theorem 4.1 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 4.3, 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 4.7 (5) that corresponding $\sim^{\bar{\mathbf{C}}}$ is a logical PER on $\tilde{\bar{\mathbf{C}}} \subseteq \bar{\mathbf{C}}$ which, moreover, coincides with $\sim$ from Theorem 4.1. The latter holds according to Proposition 4.7 (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 $\mathbf{C}_{\alpha\to\beta}$ for $g \in \bar{\mathbf{C}}_{\alpha\to\beta}$. Therefore the two versions of $\tilde{\bar{\mathbf{C}}}$ coincide.

By using Theorem 4.8 this simultaneously proves Theorem 4.1:

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

5 The Case of Sums

The rest of the paper is concerned with sums in $\mathbf{C}$-version. For simplicity, we postpone product types to Section 6.

Unfortunately, the ordinary set theoretical definition of $\mathbf{C}_{\gamma+\delta}$ as the *separated union* of $\mathbf{C}_\gamma$ and $\mathbf{C}_\delta$ 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 $\prod_b S_b \rightleftharpoons \{f : B \to \bigcup S_b \mid f(b) \in S_b,\ \text{for each } b \in B\}$.

Definition 5.1 The sets $\mathbf{C}_\tau$, with τ constructed from $+$ and $\to$, are defined as follows.

$$\begin{array}{lcl}
\mathbf{C}_{\gamma+\delta} & \rightleftharpoons & \{f \in \prod_b \mathbf{C}_{(\gamma\to b)\to(\delta\to b)\to b} \mid \\
& & \quad \exists x \in \mathbf{C}_\gamma \forall b \forall g \in \mathbf{C}_{\gamma\to b} \forall h \in \mathbf{C}_{\delta\to b} (fbgh = gx) \text{ or} \\
& & \quad \exists y \in \mathbf{C}_\delta \forall b \forall g \in \mathbf{C}_{\gamma\to b} \forall h \in \mathbf{C}_{\delta\to b} (fbgh = hy)\}, \\[2ex]
\mathbf{C}_{\alpha_1\ldots\alpha_n\to b} & \rightleftharpoons & \{f \in \mathbf{C}_{\alpha_1} \to \ldots \to \mathbf{C}_{\alpha_n} \to \mathbf{C}_b \mid \\
& & \quad \forall \bar{a} \text{ basic and } \forall g_i \in \mathbf{C}_{\bar{a}\to\alpha_i}, \\
& & \quad \lambda\bar{\mathsf{a}}.f(g_1\bar{\mathsf{a}})\ldots(g_n\bar{\mathsf{a}}) \in \mathbf{C}_{\bar{a}\to b}\}, \\[2ex]
\mathbf{C}_{\alpha_1\ldots\alpha_n\to\gamma+\delta} & \rightleftharpoons & \{f \in \mathbf{C}_{\alpha_1} \to \ldots \to \mathbf{C}_{\alpha_n} \to \mathbf{C}_{\gamma+\delta} \mid \forall b \text{ basic}, \\
& & \quad \lambda\bar{x} : \bar{\alpha}.f\bar{x}b \in \mathbf{C}_{\bar{\alpha}\to(\gamma\to b)\to(\delta\to b)\to b}\}.
\end{array}$$

Note, that "or" in the definition of $\mathbf{C}_{\gamma+\delta}$ means that this domain is a union. In fact, we could equivalently define $\mathbf{C}_{\gamma+\delta}$ as the disjoint union of $\mathbf{C}_\gamma$ and $\mathbf{C}_\delta$ *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. $x, y \in \mathbf{C}_\gamma$ then $x \sim y$ iff $gx = gy$ holds for all basic b and $g \in \mathbf{C}_{\gamma\to b}$. If $x \in \mathbf{C}_\gamma$, $y \in \mathbf{C}_\delta$ then $x \sim y$ iff $gx = hy$ for all b and $g \in \mathbf{C}_{\gamma\to b}$, $h \in \mathbf{C}_{\delta\to b}$.

It is easy to show that $\mathbf{C}_{\gamma+\delta}$ is canonically isomorphic to this factorized union of $\mathbf{C}_\gamma$ and $\mathbf{C}_\delta$. Moreover, if we additionally assume that *some basic domain* $\mathbf{D}_b$ *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 $\mathbf{C}_{\gamma+\delta}$ proves to satisfy all formal properties of co-product.

Correctness of Definition 5.1 (whose well-foundedness is obtained in *two* steps, for the last clause), the formulation and proof of *combinatory completeness* of domains $\{\mathbf{C}_\tau\}$ 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:

$$\mathrm{inl} \rightleftharpoons \lambda x \Lambda b \lambda g \lambda h. gx (\in \mathbf{C}_{\alpha\to\alpha+\beta}),$$

$$\mathrm{inr} \rightleftharpoons \lambda x \Lambda b \lambda g \lambda h. hx (\in \mathbf{C}_{\beta\to\alpha+\beta}),$$

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

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 5.2 *For all* $g \in \mathbf{C}_{\alpha\to\delta}$, $h \in \mathbf{C}_{\beta\to\delta}$ *and* $F \in \mathbf{C}_{\alpha+\beta\to\delta}$ *we have*

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

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

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 $\mathbf{C}_{\alpha+\beta} = \{\mathrm{inl}\, x \mid x \in \mathbf{C}_\alpha\} \bigcup \{\mathrm{inr}\, y \mid y \in \mathbf{C}_\beta\}$ 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 5.1:

$$\mathbf{C}_{\gamma\times\delta} \rightleftharpoons \{\langle x, y\rangle \mid x \in \mathbf{C}_\gamma, \ y \in \mathbf{C}_\delta\},$$

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

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 $\{\mathbf{C}_\tau\}$ comprise an extensional model for typed λ-β-η-calculus with additional primitive constructs $[-,-]$, inl, inr satisfying the three identities of Proposition 5.2 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

Buss S. [1986], The polynomial hierarchy and intuitionistic bounded arithmetic, *in* 'Structure in Complexity Theory', Vol. 223 of *Lecture Notes in Computer Science*, pp. 77–103.

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

Cook S. and Urquhart A. [1989], Functional interpretations of feasibly constructive arithmetic, *in* 'Proc. 21st ACM STOC', pp. 107–112.

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

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

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

Mitchell J. [1990], Type systems for programming languages, *in* J. van Leeuwen, ed., 'Handbook of Theoretical Computer Science', Elsevier Science, chapter 8, pp. 367–458.

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

Rogers H. [1967], *Theory of recursive functions and effective computability*, McGraw Hill.

Sazonov V. [1990], An equivalence between polynomial constructivity of Markov's principle and the equality P=NP, *in* P. Petkov, ed., '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', Plenum Press, NY, pp. 351–360.

Sazonov V. and Sviridenko D. [1986], Denotational semantics of the language of σ-expressions (in Russian), *in* 'Logical Problems in the Theory of Data Types', Vol. 114 of *Vychislitelnye sistemy*, Novosibirsk, pp. 16–34.

Sazonov V. Y. [1989], An equivalence between polynomial constructivity of markov's principle and the equality P=NP, (in Russian), *in* 'Mathematical Logic and Algorithmical Problems', Vol. 12 of *Trudy Instituta Matematiki*, Nauka, Novosibirsk, pp. 138–165.

Statman R. [1985], Logical relations and the typed λ-calculus, Vol. 65, pp. 85–97.

VORONKOV A. [1990], Towards the theory of programming in constructive logic, *in* N. Jones, ed., 'Proc. of the European Symp. on Programming', Vol. 432 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 421–435.

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