F. Barral, D. Chemouil, S. Soloviev

NON-STANDARD REDUCTIONS AND CATEGORICAL MODELS IN TYPED LAMBDA-CALCULUS

Abstract. We consider the problem of incorporation of new computational rules in lambda calculus with inductive types and recursion. We consider the extensions of standard reduction systems by certain new reductions preserving strong normalization and Church-Rosser property with possible applications to proof assistants and computer algebra systems.

1.Introduction.

Computational power of untyped lambda-calculus is sufficient to represent any partial recursive function. One of obvious drawbacks is that some basic questions (like termination) are undecidable. Nowadays more and more attention is paid to various systems of typed lambda-calculi since typing provides greater safety. In "non-pathological" systems, computation represented by well-typed term always terminates.

Another positive aspect of typed calculi in comparison with untyped case is due to so called "proofs as programs paradigm". The type of a term can be considered as logical formula and the term represents its proof. At the same time it can be considered as a program. This explains why typed lambda-calculi are often used in modern proof-assistants. In perspective, this is one of possible ways to unification of proof and computation.

Of course, typing doesn't resolve all the difficulties. One of them is that the representation of real computations in lambda-calculus including only the fundamental term and type constructors (application and abstraction for terms, functional arrow for types) is very indirect, it is in fact complex coding, satisfactory for theoretical results but lacking directness and transparency required for efficient applications. Extensions of typed systems with "real-life" inductive types like natural numbers, lists, trees and corresponding functional constants and recursion operators are helpful but not sufficient. Mathematics computations are seldom represented in the form of recursive functions even if they are fully constructive.

Symbolic computations, for example, often include the transformations of symbolic expressions that were never studied from the point of view of properties of corresponding rewriting system. The importance of the problem of certified computation, symbolic or numerical (i.e., computation that is completed with the proof of its correctness) was emphasized several years ago in [4] but it was studied since in very limited number of cases.

The possibility usually provided by proof-assistants based on type theory is to obtain a proof-term that represents the proof of equality of two terms representing computations. This term should be carried everywhere the equality should be used, and this turns out to be very heavy and inefficient.

One of the reasons is that the system of reductions of terms incorporated in the underlying typed lambda-calculus is very restrictive. Because of this even very simple equalities used routinely very often require the proof-term corresponding to this equality to be carried around. It may require quite complex manipulations if the equality is used within another computation.

The approach we are studying in this paper is based on extension of the systems of reductions preserving good properties of the reduction system as a whole. Such properties as Strong Normalization (SN) and Church-Rosser property, or confluence (CR) need to be proved only once. Afterwards the use of the lambda-calculus may follow similar schema: some equalities are proved by reduction (this is much more efficient) and for some others we need to find a proof-term, but the classes of these “intensional” and “extensional” equalities are different, we have more “intensional” equalities. As result the transparency and efficiency of a system may be improved.

In this paper we consider several model cases of extensions of reduction systems in the calculus that doesn’t contain the type Prop and terms representing proofs, i.e., we concentrate on the computational part. This permits to simplify the technical side of the presentation. The calculus under consideration is simply typed lambda-calculus with inductive types.

Three cases are considered:

- the notion of a copy of inductive type and the reductions necessary to make it an isomorphism (it is not an isomorphism w.r.t. standard system of reductions and this complicates a lot the handling of copies of inductive types);
- the reductions to incorporate into lambda-calculus certain algebraic structures, such as groups of permutations;

- functoriality of the schema of inductive type (a schema of parameterized inductive type, like List(A), does not represent a functor w.r.t. standard reductions).

The aim of this paper is to present an approach that would help to bring closer proof and computation. The results concerning functoriality are completely new, the results concerning copy and groups of permutations were partly published (see [5], [9]).

2. Simply-typed lambda-calculus with inductive types.

We will consider infinite sets of constructor name (Const), term variables (Var) and type variables (Tvar), with $\mathrm{Const}\cap\mathrm{Var} = \mathrm{Const}\cap\mathrm{Tvar} = \mathrm{Var}\cap\mathrm{Tvar} = \varnothing$. We will reserve the letters x, y and z for term variables, α and β for type variables, r, s, t and u for arbitrary terms, ρ and τ for arbitrary types, and κ for constructor schemas. The letters i, j, k, l will only be used for indexes and, respectively, n, m, p, q for their upper bound. Finally, constructor names will be denoted either by c_1, c_2, …, c'_1, c'_2… or by the generic name *in*. Definitions will be introduced by the symbol =def , as in id =def $\lambda x^\tau .x$. Terms and types will be considered up to α-congruence (that is, the names of bound variables are meaningless) and this last relation will be denoted $\equiv$, thus one has $\lambda x^\tau.x \equiv \lambda y{:}\tau .y$. Sequences of types or terms $(t_i)_{i=1,n}$ will be written as $t_{1\div n}$. Using this notation we will sometimes write $\boldsymbol{\rho}_{1\div n}\to\tau$ to mean $\rho_1\to\ldots\to\rho_n\to\tau$, associated to the right. Furthermore, $s\in \mathbf{t}_{1\div n}$ will mean that there is an i such that $s\equiv\mathbf{t}_i$, and $\mathbf{t}_{i\div n}\in S$ will mean that all the $\mathbf{t}_i$'s belong to the set S. Finally, if some indexes depend on other ones, we shall write j(i), and $\mathbf{t}_{j(1\div n)}$ will stand for $t_{j(1)},\ldots, t_{j(n)}$. We will also need the notion of "curried" composition: for given lambda-terms f: $\boldsymbol{\rho}_{1\div n}\to\tau$ and $g{:}\tau\to\upsilon$, gof will be defined as $\lambda \mathbf{z}_{1\div n}{:}\boldsymbol{\rho}_{1\div n}.g(f\mathbf{z}_{1\div n})$, with $z_{1\div n}\notin FV(g)$ and $z_{1\div n}\notin FV(f)$. We shall also use the following notation, provided of course that f and g are of suitable types: $g\bullet f\equiv$ gof if f and g are composable, gf if they are not, but g can be applied to f.

Definition 1. (Prototypes.) The grammar of prototypes is defined as follows:
$\tau::=\alpha|\tau\to\tau|\mu\alpha(c_{1\div n}{:}\tau_{1\div n})$, with $\alpha\in$ Tvar.

Definition 2. (Types). We define simultaneously:

The set Ty of types:

$$\frac{\upsilon \in \mathrm{Tvar}}{\upsilon \in \mathrm{Ty}} \qquad \frac{\rho, \tau \in \mathrm{Ty}}{\rho \rightarrow \tau \in \mathrm{Ty}} \qquad \frac{c_{1\div n} \in \mathrm{Const}; \boldsymbol{\alpha} \in \mathrm{Tvar}; \boldsymbol{\kappa}_{\mathbf{1\div n}} \in \mathrm{Sch}(\alpha)}{\mu\alpha(c_{1\div n} \mathbf{:} \boldsymbol{\kappa}_{1\div n}) \in \mathrm{Ty}}$$

and the set Sch(α)of constructor schemas over type variable α :

$$\frac{\rho_{1\div m}, \sigma_{1,1\div j(1)}, \ldots, \sigma_{n\div j(n)} \in \mathrm{Ty}}{\boldsymbol{\rho}_{\mathbf{1\div m}} \rightarrow (\boldsymbol{\sigma}_{1,1\div j(1)} \rightarrow \alpha) \rightarrow \ldots \rightarrow (\boldsymbol{\sigma}_{n,1\div j(n)} \rightarrow \alpha) \rightarrow \alpha \in \mathrm{Sch}(\alpha)}$$

As usual, constructor names can only belong to one inductive type. Thus, an inductive type is also defined by names of its constructors.

Remarks:

An inductive type is a recursive type built from a sequence of (constructor) schemas.

Every schema κ_k over α is of the form $\boldsymbol{\rho}_{\mathbf{1\div m}} \rightarrow (\boldsymbol{\sigma}_{1,1\div j(1)} \rightarrow \alpha) \rightarrow \ldots \rightarrow (\boldsymbol{\sigma}_{n,1\div j(n)} \rightarrow \alpha) \rightarrow \alpha$ and each premise is called an operator over α. The number of operators in a schema is denoted ar(κ_k) (arity). We write $nb^P(\kappa_k)$=m for the number of ρ's and $nb^R(\kappa_k)$=n for the number of operators ($\boldsymbol{\sigma}_{i,1\div j(i)} \rightarrow \alpha$), thus we have $ar(\kappa_k) = nb^P(\kappa_k) + nb^R(\kappa_k) = n+m$.

The ρ's and σ' s are in Ty, which implies they don't contain any free type variable. They are called parameter types. The occurrences belonging to $\boldsymbol{\rho}_{\mathbf{1\div m}}$ are called covariant and to $\boldsymbol{\sigma}_{1,1\div j(1)}$,..., $\boldsymbol{\sigma}_{n,1\div j(n)}$ contravariant. The fact that they don't contain any free type variable implies also that the only occurrences of α are those explicitly shown and α occurs only strictly positively in the operators of the schema. The operators containing α are recursive (correspond to "recursive calls"). If the list $\boldsymbol{\sigma}_{i,1\div j(i)}$ is empty , such operator is called 0-recursive otherwise 1-recursive (by analogy with the functionals of types 0 and 1 in Godel's system T). By definition of schemas, parameter types can only occur at the beginning of the schema: this restriction is useful for technical reasons, most notably for the typing of recursors and the definition of their computation rules. It will be clear to the reader that this is a minor restriction which does not impair the system at all.

Example 1. With the rules for inductive types described above, it is possible to define the types of natural numbers, of Brouwer's ordinals and of lists of natural numbers:

Nat $=_{def} \mu\alpha[0:\alpha, \mathrm{succ}:\alpha \rightarrow \alpha]$

Ord $=_{def}$ $\mu\alpha[0_{ord}{:}\alpha,\ succ_{ord}{:}\alpha\to\alpha,\ lim{:}\ (Nat\to\alpha)\to\alpha]$

List(Nat) $=_{def}$ $\mu\alpha[nil{:}\alpha,\ \ cons{:}\ Nat\to\alpha\to\alpha]$

Note that every inductive type τ generates a recursor (or structural-recursion operator) to any type μ. This is explained below.

Definition 3. (Terms). The set of terms is generated by the following grammar (with $x\in$ Var, $k\in N\backslash\{0\}$ and $\tau, \mu\in Ty$):

$t ::= x \mid \lambda x\tau\, t \mid (t\ t) \mid in_k^{\mu} \mid (\mid t_{1\div n} \mid)^{\mu,\tau}$

Here in_k^{μ} is the k-th constructor of the inductive type μ (in practice, we actually have constructor names $c\in$ Const) and $(\mid t_{1\div n} \mid)^{\mu,\tau}$ is a recursor (or structural recursion operator) from μ to another type τ.

Definition 4. (Step type.) Given inductive type(s) $\mu \equiv \mu\alpha(c_{1\div n}\boldsymbol{:}\boldsymbol{\kappa}_{1\div n})$ and a result type τ, we define for every

$\kappa_k \equiv \boldsymbol{\rho}_{\mathbf{1\div m}}\to(\boldsymbol{\sigma}_{1,1\div j(1)}\to\alpha)\to\ldots\to(\boldsymbol{\sigma}_{n,1\div j(n)}\to\alpha)\to\alpha$ in Sch(α)

the step type

$\delta^{\mu,\tau}$

$\equiv \boldsymbol{\rho}_{\mathbf{1\div m}}\to(\boldsymbol{\sigma}_{1,1\div j(1)}\to\mu)\to\ldots\to(\boldsymbol{\sigma}_{n,1\div j(n)}\to\mu)\to(\boldsymbol{\sigma}_{1,1\div j(1)}\to\tau)\to\ldots\to(\boldsymbol{\sigma}_{n,1\div j(n)}\to\tau)\to\tau$

Definition 5. (Typing) We define the following typing rules of the calculus:

$$\frac{}{\Gamma, x{:}\tau \vdash x{:}\tau}\ (Var)$$

$$\frac{\Gamma, x{:}\rho \vdash t{:}\tau}{\Gamma \vdash \lambda x{:}\rho.t{:}\rho\to\tau}\ (Lambda) \qquad \frac{\Gamma \vdash t{:}\rho\to\tau \quad \Gamma\vdash u{:}\rho}{\Gamma \vdash (t\ u){:}\tau}\ (App)$$

$$\frac{c\in Const}{\Gamma \vdash c_k : \kappa_k[\mu]}\ (In) \qquad \frac{\Gamma \vdash t_{1\div n} : \delta_{1\div n}{}^{\mu,\tau}}{\Gamma \vdash (\mid t_{1\div n} \mid)^{\mu,\tau}{:}\mu\to\tau}\ (Rec)$$

Sometimes for typographical reasons we shall write types of variables as superscripts

Reduction. We take most of our terminology and notation in [2]. Given a binary relation R on a set A, we will denote the induced rewrite relation $\rightarrow_R$, but shall sometimes write R for $\rightarrow_R$ and vice-versa. We will respectively write $\rightarrow^*_R$, $\rightarrow^+_R$, and $=_R$ for its transitive, reflexive-transitive and reflexive-symmetric-transitive closures. Sometimes we may write R^*, R^+ and $R^=$. We say that a term t rewrites to u if there is a term u such that $t\rightarrow_R u$ and it reduces to u if there is a derivation $t\rightarrow_R^+ u$. The union $R\cup S$ of binary relations on the same set will be denoted RS. We also write R;S for the set $\{(r,s) \mid \exists t.\ rRt\wedge tSs\}$. A term is in normal form if it is not rewriteable. A rewrite relation R is strongly normalizing (terminating) is there is no infinite derivation $t_1\rightarrow_R t_2 \rightarrow_R\ldots$, for any term t_1.

Given two rewrite relations R and S: R commutes with S if $^*\!\leftarrow_S;\rightarrow^*_R\subseteq\ \rightarrow^*_R;\ ^*\!\leftarrow_S$. R commutes strictly locally over S if $\leftarrow_S;\rightarrow_R\subseteq\ \rightarrow_R;\ \leftarrow_S$.

This definition is made in [8], and by R. Di Cosmo in [9] to state Akama- Di Cosmo's lemma under the name of (DPG) condition (see lemma 1 below).

A relation R is confluent (resp. locally confluent) if it commutes (resp. commutes locally) with itself. A strongly normalizing and confluent relation is said convergent. We will also write R/S to represent the quotient of a relation R by the reflexive-symmetric-transitive closure of S.

The usual notion of substitution is written $t\{u/x\}$ to mean that u replaces every free occurrence of x in t, avoiding capture. Finally, as usual in this kind of work, we will consider contexts, written C[], that is, terms with a "hole" inside which can be filled (giving for example $C[(\lambda x^\tau.p)q]$.

Definition 6. (β-conversion) We define the relation of β-conversion by the following rule: $(\beta).(\lambda x^\tau.t)u\rightarrow\beta\ t\{u/x\}$.

Definition 7. (η-conversion). We define η-conversion by the following rule (η) $t\rightarrow\eta\ \lambda x^{\tau.}tx$ if t of type $\tau\rightarrow\upsilon$ is not in applicative position, does not begin with λ and $x\notin FV(t)$.

(This rule is also called η-expansion and is known to be more convenient for categorical applications than η–reduction oriented in opposite way.)

Definition 8. (ι-conversion). Let $\mu\equiv\mu\alpha(c_{1\div n}\mathbf{:}\boldsymbol{\kappa}_{1\div n})$, and

$\kappa_k \equiv \boldsymbol{\rho}_{1\div m} \to (\boldsymbol{\sigma}_{1,1\div j(1)} \to \alpha) \to \ldots \to (\boldsymbol{\sigma}_{n,1\div j(n)} \to \alpha) \to \alpha$ over α in μ. Let $v_{1\div m}{:}\boldsymbol{\rho}_{1\div m}$ and $u_{1\div n}$ with $u_i^R{:}\ \boldsymbol{\sigma}_{i,1\div j(i)} \to \mu$ for any $1 \le i \le n$. Then, we define ι-reduction by the rule

(ι) $(|\, \mathbf{t}\, |)^{\mu,\tau} in^{\mu}_k\, (v_{1\div m}\, ,\, u_{1\div n}) \to_\iota t_k\, (v_{1\div m},\, u_{1\div n}\, ,\, ((|\, t_{1\div p}\, |)^{\mu,\tau} \bullet u_{1\div n}))$.

Remark 1. Recall that $g \bullet f$ is just an abbreviation. Hence, we may describe ι-reductions as $(|t_{1\div p}\, |)^{\mu,\tau} in^{\mu}_k\, (v_{1\div m}\, ,\, u_{1\div n}) \to_\iota t_k\, (v_{1\div m},\, u_{1\div n}\, ,\, \boldsymbol{\Delta}_{1\div n}\, (u_{1\div n}))$ where

$\Delta_i(u_i) \equiv (|t_{1\div p}\, |)^{\mu,\tau}\, u_i$ if $u_i^R{:}\mu$ (i.e., u_i^R is 0-recursive), and $\Delta_i(u_i) \equiv (|t_{1\div p}\, |)^{\mu,\tau} \circ u_i$ if $u_i{:}\ \boldsymbol{\sigma}_{i,1\div j(i)} \to \mu$ (i.e., u_i is 1-recursive).

Example 2. If we take the type of Brouwer's ordinals, $Ord =_{def} \mu\alpha[0_{ord}{:}\alpha,\ succ_{ord}{:}\alpha \to \alpha,\ lim{:}\ (Nat \to \alpha) \to \alpha]$, then, given some type τ (the type of result), the step types corresponding to 0_{ord}, $succ_{ord}$ and lim will be respectively τ, $Ord \to \tau \to \tau$ and $(Nat \to Ord) \to (Nat \to \tau) \to \tau$, the recursor will be of the form $(|t_1,t_2,t_3|)$ with $t_1{:}\tau$, $t_2{:}Ord \to \tau \to \tau$, $t_3{:}(Nat \to Ord) \to (Nat \to \tau) \to \tau$ and the ι-reduction will take the following forms:

$(|t_1,t_2,t_3|)\ 0_{ord} \to_\iota t_1$, $(|t_1,t_2,t_3|)\ succ_{ord}(u_1) \to_\iota (t_2\ u_1)((|t_1,t_2,t_3|)u_1)$,

$(|t_1,t_2,t_3|)\ lim\ (u_2) \to_\iota (t_3\ u_2)\ ((|t_1,t_2,t_3|) \circ u_2) \equiv (t_3\ u_2)\ (\lambda x^{Nat}.((|t_1,t_2,t_3|)(u_2 x^{Nat}))$

(here $u_1{:}Ord$, $u_2{:}Nat \to Ord$).

The λ-calculus thus defined, together with βηι-conversion, is called βηι.

In the rest of this paper, we will often omit type indications, except for abstracted variables, to lighten the notation.

3. Main results

3.1. Copy

Let us consider the type $\mu \equiv \mu\alpha[c_1{:}\sigma_{1,1\div j(1)} \to \alpha,\ \ldots\ c_p{:}\ \sigma_{p,1\div j(p)} \to \alpha]$

An exact copy of this type differs only by names of introduction operators, e.g., $\mu\,'$ with introduction operators $c'_{1\div p}$. It is faithful or isomorphic copy of μ if some of the parameters π are replaced by isomorphic types π'.

Remark 2. The types π and π' are isomorphic if there exist $f: \pi \to \pi'$ and $f': \pi' \to \pi$ such that f of' and f'of can be reduced to id_{π}, $id_{\pi'}$ respectively. In this case we write $f:\pi \leftrightarrow \pi': f'$

In general we call "copy" of μ any type μ' that differs by names of introduction operators and some parameters π are replaced by π' with $f: \pi \to \pi'$ and $f': \pi' \to \pi$, but it is no more required that f, f' were mutually inverse.

Let one occurrence of π into $\mu' \equiv \mu\alpha[\ c_1:\sigma_{1,1\div j(1)} \to \alpha, \dots\ c_p: \sigma_{p,1\div j(p)} \to \alpha]$ be fixed and μ' be a copy of μ such that the occurrence of π is replaced by π' (other changes concern only the names of introduction operators). We shall consider only the case when π occurs as a parameter type. Assume that it is given $f: \pi \to \pi'$ when the occurrence is covariant and $f: \pi' \to \pi$ it is contravaiant. The function $Cp(f): \mu \to \mu'$ is defined as $(| t_{1\div p} |)$ where the terms $t_1, \dots, t_p$ are defined in the following way.

We shall note by $\underline{f}\,r$ the application fr if $r:\pi$ or $r:\pi'$ corresponds to an occurrence to be replaced and r otherwise. Similarly, we shall write $g\ \underline{o\ f}$ for g o f if g has π or π' as its domain and for g otherwise.

Let us consider the introduction operator $c_i :\sigma_{i,i\div j(i)} \to \mu$. We may assume that $\sigma_{i,i\div j(i)} \to \mu \equiv \pi_{1\div k} \to \mu \to \dots \to \mu \to (\pi_{1,1\div n(1)} \to \mu) \to \dots \to (\pi_{m,1\div n(m)} \to \mu) \to \mu$ (with l premises of the form μ).

We define:

$t_i \equiv \lambda x_{1\div k}\colon \pi_{1\div k}.\lambda y_{1\div l}\colon\mu.\lambda z_{1\div m}\colon \pi_{1\div m,1\div n(1\div m)} \to \mu.\lambda u_{1\div l}\colon\mu'.\ \lambda v_{1\div m}\colon \pi_{1\div m,1\div n(1\div m)} \to \tau.c_i'(\underline{f}\ x)r(s\ \underline{o\ f'})$.

Example 3. Take again the type of Brouwer's ordinals and let $f: Nat' \to Nat$. We have $Cp(f) \equiv (| t_{1\div 3} |)$, $t_1 \equiv 0'_{ord}$, $t_2 \equiv \lambda y^{Ord}.\ \lambda u^{Ord'}.succ'_{ord}(u)$, $t_3 \equiv \lambda z^{Nat\to Ord}.\lambda v^{Nat\to Ord'}.lim'(v\ o\ f)$.

When f is an isomorphism with inverse f^{-1}, the function Cp(f) is an extensional isomorphism in the sense that for every canonical element e of type μ (constant term containing only $c_1, \dots, c_p$) $Cp(f^{-1})(Cp(f)\ e) \to_{\beta\eta\iota} e$. One of main motivations to study non-standard reductions and their properties was for us the fact that within standard system of reductions $\beta\eta\iota$ many equalities are only extensional, for example $Cp(f^{-1})\ o\ Cp(f)$ does not reduce to id_{μ} or, equivalently, $Cp(f^{-1})(Cp(f)\ x)$ does not reduce to x if x is a variable, so, in practice, we have either to carry everywhere proof-term or we cannot verify the equalities before some constant value is called.

This will be the case even if we take f to be identity. For example, the composition of Cp(id) for Ord $\equiv \mu\alpha[0_{ord}:\alpha, succ_{ord}:\alpha\rightarrow\alpha, lim: (Nat\rightarrow\alpha)\rightarrow\alpha]$ and Ord'$\equiv\mu\alpha[0'_{ord}:\alpha, succ'_{ord}:\alpha\rightarrow\alpha, lim': (Nat\rightarrow\alpha)\rightarrow\alpha]$ will not be reducible to $id_{Ord}\equiv \lambda x^{Ord}.x$. In fact the term $(| t'_{1\div3}|) ((| t_{1\div3}|)x)$ will be $\beta\eta\iota$–normal.

Meanwhile, as the results below show, reductions can be added to $\beta\eta\iota$ to make the resulting system SN and CR.

Definition 9. (χ-reduction). Let f be an isomorphism and f^{-1} its inverse. The χ-rewriting rule is defined by

(χ_1) $Cp(f^{-1})(Cp(f)r)\rightarrow_{\chi} r$

(χ_2) $Cp(f^{-1})(Cp(f)r)\rightarrow_{\chi} r$,

where it is supposed that f and f^{-1} act at the same occurrence of parameter of some inductive type μ and its faithful copy μ', r is arbitrary term.The χ-reduction is its contextual closure.

For lambda-calculus with inductive types considered in this paper the following theorem holds:

Theorem 1. The $\beta\eta\iota\chi$ reduction is SN and CR.

The detailed proof of this theorem may be found in [5]. Here we rather would like to discuss in more concrete way than before the specifics of the proofs of SN and CR for the extensions of reduction systems of the type we consider in this paper, both its technical and conceptual aspect.

The proof of SN for $\beta\eta\iota\chi$ reduction uses some standard lemmas, first of all, the Akama-Di Cosmo Lemma:

Lemma 1. Let R and S be two convergent relations, such that R preserves S-normal forms. Then RS is convergent if R commutes strictly locally over S. [1, 8].

Standard techniques are sufficient to prove convergence of $\beta\eta\iota$ part. To prove SN property in theorem 1 we need some more definitions and lemmas.

Definition 10. (Adjournment.) Given two binary relations R and S, S is adjournable w.r.t. R if S; R $\subseteq$ R, (RS)*.

Lemma 2. (Adjournment lemma.) Given two strongly normalizing relations R and S, RS is strongly normalizing if S is adjournable w.r.t. R.

Proofs of (variants of) this lemma can be found in literature [1, 3, 7, 8]. A subtle point in the proof of SN for $\beta\eta\iota\chi$ is that there are cases when the adjournment lemma can be used only on condition that certain 1-recursive arguments of ι-redex are η-expanded. The idea is therefore to insert suitable η-expansions in a term before χ-conversion, so that the adjournment remains possible.

Definition 11. (Conditional adjournment.) Let R and S be some reduction relations and P be a predicate on terms. Then S is adjournable w.r.t. R under condition P if $\forall t \forall t' \forall t''.\ P(t) \wedge t \rightarrow_S t' \wedge t' \rightarrow_R t'' \Rightarrow \exists u.\ t \rightarrow_R u \rightarrow^*_{RS} t''$.

Definition 12. (Realization.) Let T be a reduction relation, P be a predicate on terms and t some term. Then T realize P for t if $\exists t'.\ t \rightarrow^*_T t' \wedge P(t')$. It will be said that T realizes P if this is true for every term t.

Definition 13. (Insertability.) Let U, T be two reduction relations and Q a binary relation on terms. Then T is insertable in U w.r.t. S if:

- $T \subset U$,
- If t_1 Q t_2 and $t_1 \rightarrow_{\{U \setminus T\}} t_1'$ then there exists t_2' such that $t_2 \rightarrow^+_U t_2'$ and t_1' Q t_2',
- If t_1 Q t_2 and $t_1 \rightarrow_T t_1'$ then there exists t_2' such that $t_2 \rightarrow^*_U t_2'$ and t_1' Q t_2'

Lemma 3. (Insertion). Let U, T be two reduction relations and Q a binary relation such that:

- T is insertable in U w.r.t. Q,
- T is SN,

Then for every infinite sequence of U-reductions u beginning at the term t, and every T-reduction $t \rightarrow_T t'$ such that t' Q t there exists an infinite sequence of U-reductions that has as its first step $t \rightarrow_T t'$.

The principal idea of insertion and insertion lemma is that we can add necessary reductions and preserve infinite sequences of reductions if they exist. This is useful for the proofs of strong normalization "ad absurdum". In the proof of theorem 1 this lemma is used with η-expansions as T, the whole $\beta\eta\iota\chi$ as U, and the relation of η-reduction inverse to η-expansion as Q.

Lemma 4. (Pre-adjusted adjournment.) Let R, S, T be reduction relations, Q a binary relation and P a predicate on terms such that:

- T⊂ R,
- R is SN,
- S is SN,
- T realizes P,
- S is adjournable w.r.t. R under condition P,
- T is insertable into RS w.r.t. Q.

Then RS is SN.

This lemma is used in the setting similar to lemma 3, with P(t) meaning that the term t is in η-expanded form. T corresponds to η-expansion.

These lemmas are sufficient to prove SN property.

The proof of CR property (confluence) is based on routine check of possible critical pairs.

The following example shows how the notion of copy may be used to define easily interesting data structure.

Example 4. In type theory the (easily defined) embeddings of Nat into Nat are used to define even and odd numbers. In the definition of even numbers 0 is mapped to 0, 1 to 2 etc., and in case of odd numbers 0 is mapped to 1, 1 to 3... In fact, to make this definition "clean" the copies of Nat should be used. Let us note these copies by Nat', Nat''. Let E: Nat'→Nat and O: Nat''→ Nat be corresponding embeddings. We have also Cp': Nat→Nat' and Cp'': Nat→Nat'' (there is no change of parameter, so there is no parameter f in Cp', Cp''). Combining E, O, Cp' and Cp'' we can now iterate the whole construction:

$$\begin{tikzcd}
 & & \text{Nat} & \text{Nat}' \arrow[l, "E"'] & \text{Nat} \arrow[l, "Cp'"'] & \dots \arrow[l, "E"'] \\
 \arrow[r] & \text{Nat} \arrow[r, "Cp''"] & \text{Nat}'' \arrow[u, "O"] & \text{Nat} \arrow[u, "Cp'"] & \arrow[l] \arrow[u, "O"] & \\
 & \arrow[u] & \arrow[r] & \text{Nat} \arrow[ul, "Cp''"'] & \arrow[ul] & \\
 & & & \arrow[u] & &
\end{tikzcd}$$

and within this structure define all subtypes of Nat defined via divisibility by 2^n.

3.2 Algebraic Structures.

In this part we consider the extensions of reduction systems used to provide good representation of algebraic structures on finite types.

Finite set $|n| = \{1,\dots, n\}$ will be represented by the type $\underline{n} =_{def} \mu\alpha(cn_1{:}\alpha,\dots, cn_n{:}\alpha)$ (of course many representations that differ only by the names of the constructors are possible).

To every function $f{:}\ |n|\rightarrow |m|$ corresponds a term $\underline{f}{:}\underline{n}\rightarrow \underline{m}$ of the form $(|cm_{f(1)},\dots, cm_{f(n)}\ |)$ where $\underline{m}$ is $\mu\alpha(cm_1{:}\alpha,\dots, cm_m{:}\alpha)$. Note that the terms $\underline{f}$ are normal.

We considered two problems concerning finite types and terms $\underline{f}$: (a) What categorical structure can be introduced on this calculus and (b) what ca be done to represent symmetric group using finite types and corresponding representation of permutations.

The difficulty in type theory as usual is that w.r.t. standard reductions one doesn't for example have $\underline{g}(\underline{f}\ r) =_{\beta\eta\iota} (\underline{g \circ f})\ r$ for arbitrary term r.

Definition 14. We define υ-rewriting by

$\underline{g}(\underline{f}\ r) \rightarrow_{\upsilon} (\underline{g \circ f})\ r$

and υ-reduction as its contextual closure.

The υ-reduction is thus defined for all recursors in normal form representing the applications $f{:}\ |n|\rightarrow |m|$, $g{:}\ |m|\rightarrow |p|$. It is supposed, of course, that "externally" the functions f and g are known.

Theorem 2. The $\beta\eta\iota\upsilon$ reduction is SN and CR.

The proof of this theorem uses essentially the same lemmas as in case of copy (the details may be found in [5], cf. also [6, 9]).

Categorical structure. As soon as the υ-reduction is integrated in the calculus, it becomes possible to define categorical structure on this calculus, in the following way:

- the objects are the types representing $|n|$ for all $n\in N$ (let us recall that there is infinitely many of them because the names of the constructors may be different);

- the arrows between n and m are the equivalence classes modulo $\beta\eta\iota\upsilon$ of the recursors $\underline{f} : \underline{n}\rightarrow\underline{m}$ for every $f{:}\ |n|\rightarrow|m|$.

One may consider $\underline{id_{\underline{n}}},=_{def} (|cn_1,\dots, cn_n\ |)$, one of many representations (associated with $\underline{n}$) of the identity map on $|n|$. Let us recall that one has also $id_{\underline{n}} \equiv \lambda x^{\underline{n}}.x$. This term doesn't belong obviously to the categorical structure described above. It may be noticed that the reduction of $\underline{id_{\underline{n}}}$, to $id_{\underline{n}}$ is not necessary for the categorical construction described above because it would lead us outside this categorical structure and, moreover, one already has $\underline{f} \circ \underline{id}_{\underline{n}} \leftrightarrow^* \underline{f} \leftrightarrow^* \underline{id}_{\underline{n}} \circ \underline{f}$. This is a case when in certain categorical structures within λ-calculus the term chosen to represent identity is not necessarily of the form λx.x .

Interaction with the copies. The results concerning υ-reduction presented above didn't take into account copies and χ-reduction. In fact when the χ-reduction is added, the identification of $\underline{id}_{\underline{n}}$ and $id_{\underline{n}} \equiv \lambda x^{\underline{n}}.x$ may be necessary, since with χ-reduction the following critical pair appears. Let us take Cp: $\underline{n} \to \underline{n}'$ and Cp':$\underline{n}' \to \underline{n}$ (the copy map without change of parameter). We'll have:

$$\underline{id}_{\underline{n}}x \ {}_{\upsilon}\!\leftarrow Cp'(Cp\ x) \to_{\chi} x$$

To avoid non-confluence one can add the following new reduction rule.

Definition 14. (ω-reduction) The ω–rewriting relation is defined by:

$\underline{id}_{\underline{n}}r \to_{\omega} r$ for every $n \in N$ and term r

and ω–reduction relation is its contextual closure.

Theorem 3. The βηιυχ reduction is SN and CR.

Group structure. Now we consider only the case of f: $|n| \to |n|$ associated term representation $\underline{f}$: $\underline{n} \to \underline{n}$. The set of (equivalence classes of) these terms, in presence of υ-reduction, may be considered as a representation of symmetric group, i.e., the group of permutations of the set {1,..., n}. But the groups are often defined in mathematics using generators and relations, and it is natural to ask, if there is any connection between this definition and the notion of normal form used in lambda-calculus. The normal forms w.r.t. βηιυ–reduction have little to do with generators-and-relations representation of symmetric group. But instead of υ-reduction we may consider reductions going in opposite direction, i.e., "splitting" $\underline{f}$ into composition.

It is well known that every permutation f: $|n| \to |n|$ can be represented as a product of disjoint cycles.

More precisely, f is called cycle if there exists some subset $\{i_1,..., i_k\} \in \{1,..., n\}$ such that $f(i_1) = i_2,..., f(i_{k-1}) = i_k$, $f(i_k) = i_1$ and $f(i) = i$ if $i \notin \{i_1,..., i_k\}$.

Two cycles are disjoint if the corresponding sets $\{i_1,..., i_k\}$ and $\{j_1,..., j_l\}$ have no common elements.

Product in S_n is represented by functional composition of permutations.

If f: $|n| \rightarrow |n|$ then $f = f_1$ o ... f_m where $f_1,..., f_m$ are disjoints cycles and the cycles that appear in the product are unique.

Product (composition) of disjoint cycles is commutative but it is possible to order cycles (for example, lexicographically) and to have for every f unique decomposition $f = f_1$ o ... f_m with $f_1 \leq ... \leq f_m$. This suggests to study the conversion $\underline{f}\ r \rightarrow \underline{f}_1$ (... ($\underline{f}_m$ r)..) instead of $\rightarrow_\upsilon$ where f : $|n| \rightarrow |n|$ and $f_1,..., f_m$ are disjoints cycles of the unique decomposition of f.

Definition 15. The υ'-rewriting is defined by

$\underline{f}\ r \rightarrow \underline{f}_1$ (... ($\underline{f}_m$ r)..)

for every permutation f : $|n| \rightarrow |n|$, with $n \geq 2$, where f is decompose in $m \geq 2$ pairwise disjoint cycles. The υ'-reduction is defined as its contextual closure.

Theorem 4. $\beta\eta\iota\upsilon$ reduction is SN and CR.

(See [5], [9])

3.3 Functoriality of schemas.

In this part we consider most recent results obtained by Freiric Barral. These results concern more general categorical structures in lambda-calculus with inductive types.

When a schema of inductive type $\mu \equiv \mu\alpha[in_1:\sigma_{1,1\div j(1)} \rightarrow \alpha, ...\ in_p: \sigma_{p,1\div j(p)} \rightarrow \alpha]$ is given, to everybody familiar with category theory it suggests the question of functoriality of this schema w. r. t. its parameters. Assume that for every choice of parameters the names of introduction operators are fixed. The choice of parameters may be limited in advance by some set of possible values.

In fact, since many categorical structures (with types as objects) were considered on the fragments of lambda-calculus, it may be the set of objects of one of such syntactic categories.

For example, it could be certain set of types of the form $\underline{n}$.

It may be also that only some functions f: $\pi \rightarrow \pi'$ are admitted (are considered as morphisms of the underlying category.

If we want to define a functor using the schema of inductive type, it is natural to take as its values on objects the types corresponding to the values of parameter. For simplicity we shall assume that only one occurrence of parameter is modified.

For example, we may consider

$\text{List}(\pi) =_{def} \mu\alpha[nil_\pi:\alpha,\ cons_\pi: \pi \rightarrow \alpha \rightarrow \alpha]$

(we added the index π to show that the names of introduction operators are different for different values of parameter).

Or we may consider

Ord $=_{def}$ $\mu\alpha[0_{ord(\nu)}:\alpha$, $succ_{ord(\nu)}:\alpha\rightarrow\alpha$, lim_{ν}: $(\nu\rightarrow\alpha)\rightarrow\alpha]$
where ν is taking only copies of Nat as values.

Notice that in the first case we have covariant occurrence of the parameter and in another contravariant.

The function Cp(f) may be suggested now as the value on morphism f: $\pi\rightarrow\pi$'. Indeed, if we shall denote by $\mu(\pi)$, $\mu(\pi')$ the types corresponding to the values π, π' of parameter, we shall have Cp(f): $\mu(\pi) \rightarrow\mu(\pi')$ for covariant occurrence and Cp(f): $\mu(\pi') \rightarrow\mu(\pi)$ for the contravariant.

The problem will be that the equalities required in category theory: Cp(f) o Cp(g) = Cp(f o g) for covariant occurrence and Cp(f) o Cp(g) = Cp(g o f) for contravariant, and Cp(id_{π}) = $id_{\mu(\pi)}$ will not hold. It turns out that this problem can be solved by appropriate extension of the system of reductions.

Definition 16. The θ-rewriting is defined by

Cp(g)(Cp(f) r)$\rightarrow\theta$ Cp((f o g)*) r
in case of covariant occurrence of a parameter, and

Cp(g)(Cp(f) r)$\rightarrow\theta$ Cp((g o f)*) r
in case of contravariant one. Here it is assumed that f, g act on the same occurrence of a parameter, and (f o g)* denotes the $\beta\eta\iota$-normal form of (f o g). The θ-reduction is defined as its contextual closure.

The main reason to consider this reduction is to obtain new categorical structures from already defined ones together with a functor given by the schema of inductive type. It should be noted that in general one may have difficulties with the proof of SN and CR for the calculus extended by θ-reduction but since the underlying categorical structure doesn't necessarily include all the functions f: $\pi\rightarrow\pi$' definable in our calculus it is natural to consider certain restrictions on the structure of term representing f.

Theorem 5. Let in the definition of θ-reduction the following additional constraint be satisfied: f and g should be of the form $\lambda x^{\pi}.x$, of the form (| $t_{1\div n}$ |) (or expansions of such terms) where $t_{1\div n}$ do not contain free variables. Then the $\beta\eta\iota\theta$ reduction is SN and CR.

The proof of this theorem is more complex than in previous cases, especially in the part SN. The proof uses again conditional adjournment (in principal case $\rightarrow_{\theta}$ of followed by $\rightarrow_{\iota}$), but in addition we need to prove that in a special case β-reduction is inserable. (In general of course it is not, because, for example, when the term contains a redex of the form the term s can disappear because of β-reduction, and original infinite sequence may originate from s.) The inserability proof uses parallel construction of several partly defined insertion operators and the proof that at least one will indeed produce an infinite sequence of reductions if input sequence was infinite. This is used to obtain a contradiction with SN for $\beta\eta\iota$ reduction.

The constraints we had to impose on the structure of f and g in θ-reduction were necessary for the proof of confluence.

It should be noted that this variant of constraint is not the only possible constraint that will provide the "good behavior" of extended system of reductions. The fact that there are other possibilities is demonstrated by the following example.

We may take as the only object of underlying category the object Nat and as morphisms the functions succ, succ o succ, ... , succ o succ ... o succ: Nat$\rightarrow$ Nat. If we shall restrict θ-reduction to the case when f and g are of this form only, the $\beta\eta\iota\theta$ reduction will be SN and CR.

At the moment we work on more general description of possible constraints to be imposed on θ–reduction.

4. Conclusion.

Probably one of the main reasons why the "Types" community didn't yet study actively the extensions of standard reduction systems is that very little success and a lot of technical difficulties was expected. There are some exceptions [10], and hopefully more and more. Another reason is that there is still too much separation between groups working on theoretical aspects of formal methods and their applications, and between different approaches. One may mention two European research projects: "Types" and "Calculemus". While the people working on theoretical analysis of formal systems possess necessary methods and could prove useful innovative results, they are often satisfied with much less innovative solutions of standard problems. Within the class of problems we consider here it might be standard $\beta\eta$ normalizability for slightly modified calculus.

The groups working on practical aspects and implementation of theorem provers, proof assistants and alike, leave fundamental questions unanswered, or provide makeshift answers that, in long term, cannot satisfy competent user. One example cited above is the problem of extensional versus intensional equality. It is difficult to imagine a user, if only this user does not consider the answer provided by "scientifically approved" proof assistant as an oracle, who would accept that, for example, that the multiplication by 2 followed by (integer) division by 2 does not define identity on Nat. But with respect to intentional equality is not identity function.

Our methods permit to introduce simple extension of reduction system where it will be identity function.

In general the properties of extensions of reduction systems are not always easily proved, and there is many cases when they do not have good properties with respect to reduction at all.

One example "close at hand" would be the isomorphism between Nat and Nat × Nat. It holds extensionally because Nat × Nat can be enumerated, but the attempt to add "supporting" reductions, following χ-reduction as a model, fails (one doesn't obtain SN and CR system).

The point is that in many cases extensions with good properties can be successfully obtained. Moreover, this is true for some cases that are conceptually important, as with copies.

One may note that the Cp(f) permits to obtain new isomorphisms from already existing. The isomorphisms already have important role in applications, for example, for invertible transformations of data, so called "middleware", data search etc.

The same potential to generate new categorical structures from already existing within lambda-calculus has the theorem about functoriality of the schemas of inductive types w.r.t their parameters in an extension of standard reduction system that still has good properties, i.e., is convergent.

REFERENCES.

1.*Y. Akama.* On Mints' reductions for ccc-calculus. – Lecture Notes in Computer Science, vol. 664, p.1-12 (1993).

2.*F. Baader and T. Nipkow.* Term rewriting and all that. - Cambridge University Press, N. Y. 1998.

3.*L. Bachmair and N. Dershowitz.* Commutation, transformation and termination. - Lecture Notes in Computer Science, vol. 230, p. 5-20 (1986).

4.*H. Barendreg and E. Barendsen.* Autarkic computations in formal proofs. J. Autom. Reasoning 28(3), 321-336 (2002).

5.*D.Chemouil.* "Types inductifs, isomorphismes et recriture extensionnelle", ph.d. thesis, Universite Toulouse 3 (2004).

6.*D. Chemouil and S. Soloviev.* Remarks on Isomorphisms of Simple Inductive Types. – In H. Geuvers, F. Kamareddine eds., Electronic Notes in Theoretical Computer Science, vol. 85, Elsevier (2003).

7.*R. Di Cosmo.* On the power of simple diagrams. – In Harald Ganzinger, ed. Proceedings of the 7-th International Conference on Rewriting

Techniques and Applications (RTA- 96), Lecture Notes in Computer Science, vol. 1103, p. 200- 214,Springer (1996)

8.*A. Gezer.* “Relative termination”. – Ph.D. thesis, University of Passau, Germany (1990).

9.*S. Soloviev and D. Chemouil.* Some algebraic structures in lambda-calculus with inductive types. – In Stefano Berardi, Mario Coppo and Ferruccio Damiani eds., Proc. Types’03, Lecture Notes in Computer Science, vol. 3085, Springer, 2004.

10.*D. Walukiewicz-Chrzaszcz.* Termination of rewriting in the calculus of constructions. – J. of Functional Programming, vol.13(2), p.339-414 (2003).