Extensional Normalisation and Type-Directed Partial Evaluation for Typed Lambda Calculus with Sums

Vincent Balat
PPS - Université Paris 7

Roberto Di Cosmo
PPS - Université Paris 7
and
INRIA-Roquencourt

Marcelo Fiore[*]
Computer Laboratory
University of Cambridge

Abstract

We present a notion of η-long β-normal term for the typed lambda calculus with sums and prove, using Grothendieck logical relations, that every term is equivalent to one in normal form. Based on this development we give the first type-directed partial evaluator that constructs normal forms of terms in this calculus.

Categories and Subject Descriptors: F.3.2 [Logics and Meanings of Programs]: Semantics of Programming Languages—*partial evaluation*; F.4.1 [Mathematical Logic and Formal Languages]: Mathematical Logic—*lambda calculus and related systems*; D.3.1 [Programming Languages]: Formal Definitions and Theory—*semantics*

General Terms: Languages, Theory, Algorithms.

Keywords: Typed lambda calculus, Strong sums, Grothendieck logical relations, Normalisation, Type-Directed Partial Evaluation.

1 Introduction

Sum types and their associated case expressions are an essential feature of any programming language. Taking into account the full range of commuting conversions in performing program optimisations and partial evaluation in their presence is a difficult, but important, task. For example, consider the *Objective Caml* sum type

```
type ('a,'b) sum = Left of 'a | Right of 'b
```

and the program

```
fun f -> fun g -> fun z -> fun x ->
  match ( match x with
            Left  x1 -> Left z
          | Right x2 -> Right ( g z ) )      (1)
  with
    Left  y1 -> f( g y1 )
  | Right y2 -> f y2
```

[*]Research supported by an EPSRC Advanced Research Fellowship.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
POPL'04 January 14–16, 2004, Venice, Italy.
Copyright 2004 ACM 1-58113-729-X/04/0001 ...$5.00

of type

```
('a -> 'b) -> ('c -> 'a) -> 'c -> ('d, 'e) sum -> 'b
```

This is a typical, hard to read, example of what automatically generated code looks like. Using commuting conversions, it can be transformed into the program

```
fun f -> fun g -> fun z -> fun x ->
  match x with
    Left  x1 -> ( match (Left z) with
                    Left  y1 -> f( g y1 )
                  | Right y2 -> f( y2 ) )
  | Right x2 -> ( match Right( g z ) with
                    Left  y1 -> f( g y1 )
                  | Right y2 -> f( y2 ) )
```

which can be then optimised into the program

```
fun f -> fun g -> fun z -> fun x ->
  match x with
    Left  x1 -> f( g z )
  | Right x2 -> f( g z )
```

that, by extensionality, can be transformed into the more readable and efficient

```
  fun f -> fun g -> fun z -> fun x ->
    f( g z )                                  (2)
```

The commuting conversions associated to case expressions are derivable from the (strong) *sum extensionality* axiom; which identifies the programs

$$\begin{array}{l}\texttt{match e with}\\ \quad\texttt{Left x1 -> t}\left[{}^{\texttt{Left x1}}/_{\texttt{x}}\right]\\ \texttt{| Right x2 -> t}\left[{}^{\texttt{Right x2}}/_{\texttt{x}}\right]\end{array} \quad \text{and} \quad \texttt{t}\left[{}^{\texttt{e}}/_{\texttt{x}}\right]$$

Sum types satisfying this axiom are sometimes referred to as *strong* or *categorical* sums.

In this paper we consider sum types in the most basic foundational type theory for functional programming: the typed lambda calculus with sums. In particular, we tackle the problem of defining and computing normal forms in it; so that, for instance, the passage from (1) to (2) can be done automatically. Besides the interest in the typed lambda calculus with sums from the programming-language viewpoint, there is also a type theoretic one, and the study of sum types in this setting has proved challenging; see [17, 13, 16, 1].

The theory of *weak sums*, either without the extensionality axiom (see [11]) or with the extensionality axiom restricted to the case $\mathtt{t} = \mathtt{x}$ (see [10]), is well understood. However, there is as yet no known confluent and strongly normalising reduction system for strong sums. Thus, we consider below normalisation within the whole calculus in the spirit of *Normalisation by Evaluation* (NBE) and *Type-Directed Partial Evaluation* (TDPE).

NBE is a normalisation technique introduced by Berger and Schwichtenberg [4] for the simply typed lambda calculus as an inverse to the evaluation function, mapping a semantic value into a syntactic one in normal form. Since then, NBE has been the subject of investigation in many domains: logic, type theory, category theory, partial evaluation (see, *e.g.*, [7]).

Partial evaluation is a program transformation technique used to specialise functions. TDPE is a partial evaluator for functional languages invented by Danvy [5]. It is based on the same principle as NBE; it constructs code of compiled programs, acting as a decompiler.

An extension of NBE to the typed lambda calculus with binary sums has been proposed by Altenkirch, Dybjer, Hofmann, and Scott [1]. However, normalising calculi with strong sums in the style of TDPE was an open problem; to which this paper offers a solution.

For the typed lambda calculus, Fiore [14] showed that we can extract the NBE algorithm as an intentional version of an extensional-normalisation result (stating that every term equals one in normal form). Here, in the context of the lambda calculus with sums, we start by following this analysis and present a notion of normal form with respect to which we establish an extensional-normalisation result. Afterwards, we proceed in a different direction and draw insight from the proof of this result to develop a partial evaluator for the typed lambda calculus with binary sums that constructs normal forms; the extension to incorporate the empty type does not present much difficulty. The partial evaluator, written in *Objective Caml*, can be downloaded from the web.

Organisation of the paper. In Section 2, we recall the syntax and semantics of the typed lambda calculus with sums. In Section 3, after recalling the construction of bicartesian closed categories of Grothendieck relations, we present a basic lemma that provides both guidelines for defining the notion of normal term given in Section 4, and the proof-skeleton for establishing the extensional-normalisation result of Section 5. In Section 6, we present the solution to normalisation via TDPE for the simply typed lambda calculus with binary sums. Concluding remarks are offered in Section 7.

2 Typed lambda calculus with sums

We recall the syntax and categorical semantics of the simply typed lambda calculus with (empty and binary) products and (empty and binary) sums. For details see [20].

2.1 Syntax

The set of types has a (countable) set of base types and two type constants $\mathtt{1}$ and $\mathtt{0}$, the unit and empty type, and is closed under the formation of product, function, and sum type constructors. Formally, types are defined by the following grammar:

$$\begin{array}{rcll}
\tau & ::= & \theta & \text{(Base types)} \\
 & | & \mathtt{1} & \text{(Unit type)} \\
 & | & \tau_1 \times \tau_2 & \text{(Product types)} \\
 & | & \tau_1 \to \tau_2 & \text{(Function types)} \\
 & | & \mathtt{0} & \text{(Empty type)} \\
 & | & \tau_1 + \tau_2 & \text{(Sum types)}
\end{array}$$

The raw terms of the calculus are defined by the following grammar:

$$\begin{array}{rcll}
t & ::= & x & \text{(Variables)} \\
 & | & \langle\rangle & \text{(Unit)} \\
 & | & \langle t_1, t_2 \rangle & \text{(Pairing)} \\
 & | & \pi_1(t) & \text{(First projection)} \\
 & | & \pi_2(t) & \text{(Second projection)} \\
 & | & \lambda x : \tau.\, t & \text{(Abstraction)} \\
 & | & t_1(t_2) & \text{(Application)} \\
 & | & \bot_\tau & \text{(Absurd)} \\
 & | & \iota_1^{\tau_1,\tau_2}(t) & \text{(First injection)} \\
 & | & \iota_2^{\tau_1,\tau_2}(t) & \text{(Second injection)} \\
 & | & \delta(t, x_1.\, t_1, x_2.\, t_2) & \text{(Discriminator)}
\end{array}$$

where x ranges over (a countable set of) variables.

The unit, pairing, and abstraction are respectively the term constructors for the unit, product, and function types; whilst the projections and application are respectively the term destructors for the product and function types.

The term constructors for sum types are given by the injections; whilst the absurd and discriminator are respectively the term destructors for empty and sum types. In particular, discriminator terms permit definitions by cases.

The abstraction and discriminator are binding operators; $\lambda x : \tau.\, t$ binds the free occurrences of x in t, and $\delta(t, x_1.\, t_1, x_2.\, t_2)$ binds the free occurrences of x_i in t_i ($i = 1,2$). The notions of free and bound variables are standard, and terms are identified up to alpha conversion

Notice that we have adopted a non-standard (proof irrelevant) version of absurd terms as $\bot_\tau$, rather than the standard one of the form $\bot_\tau(t)$. This is important in the treatment of normal forms.

As usual we consider typing contexts as lists of type declarations for distinct variables, and say that a term t has type τ in the context Γ if the judgement $\Gamma \vdash t : \tau$ is derivable from the rules of Figure 1.

Finally, we impose the standard notion of equality on terms, including the sum extensionality axiom, as detailed in Figure 2.

2.2 Semantics

Bicartesian closed categories (BiCCCs) are categories with finite products (1, $\times$), exponentials ($\Rightarrow$), and finite coproducts (0, $+$).

The typed lambda calculus with sums is the internal language of BiCCCs and as such has sound and complete interpretations in them. With respect to an interpretation I of base types in a BiCCC S, we write $I[\![\tau]\!]$ for the interpretation of the type τ induced by the bicartesian closed structure. That is,

$$\begin{array}{rcll}
I[\![\theta]\!] & = & I(\theta) & (\theta \text{ a base type}) \\
I[\![\mathtt{1}]\!] & = & 1 & \\
I[\![\tau \times \tau']\!] & = & I[\![\tau]\!] \times I[\![\tau']\!] & \\
I[\![\tau \to \tau']\!] & = & I[\![\tau]\!] \Rightarrow I[\![\tau']\!] &
\end{array}$$

$$\frac{}{\Gamma,\ x:\tau,\ \Gamma'\vdash x:\tau}$$

$$\frac{}{\Gamma\vdash\langle\rangle:1}\qquad\frac{\Gamma\vdash t_i:\tau_i\quad(i=1,2)}{\Gamma\vdash\langle t_1,t_2\rangle:\tau_1\times\tau_2}\qquad\frac{\Gamma\vdash t:\tau_1\times\tau_2}{\Gamma\vdash\pi_i(t):\tau_i}\ (i=1,2)$$

$$\frac{\Gamma,\ x:\tau_1\vdash t:\tau}{\Gamma\vdash\lambda x:\tau_1.\,t:\tau_1\to\tau}\qquad\frac{\Gamma\vdash t:\tau_1\to\tau\qquad\Gamma\vdash t_1:\tau_1}{\Gamma\vdash t(t_1):\tau}$$

$$\frac{\Gamma\vdash t:0}{\Gamma\vdash\bot_\tau:\tau}\qquad\frac{\Gamma\vdash t:\tau_i}{\Gamma\vdash\iota_i^{\tau_1,\tau_2}(t):\tau_1+\tau_2}\ (i=1,2)\qquad\frac{\Gamma\vdash t:\tau_1+\tau_2\qquad\Gamma,\ x_i:\tau_i\vdash t_i:\tau\quad(i=1,2)}{\Gamma\vdash\delta(t,x_1.\,t_1,x_2.\,t_2):\tau}$$

Figure 1. Typing rules.

$$\frac{\Gamma\vdash t:\tau}{\Gamma\vdash t=t:\tau}\qquad\frac{\Gamma\vdash t=t':\tau}{\Gamma\vdash t'=t:\tau}\qquad\frac{\Gamma\vdash t_1=t_2:\tau\qquad\Gamma\vdash t_2=t_3:\tau}{\Gamma\vdash t_1=t_3:\tau}$$

$$\frac{\Gamma\vdash t:1}{\Gamma\vdash t=\langle\rangle:1}$$

$$\frac{\Gamma\vdash t_1:\tau_1\qquad\Gamma\vdash t_2:\tau_2}{\Gamma\vdash\pi_i\langle t_1,t_2\rangle=t_i:\tau_i}\ (i=1,2)\qquad\frac{\Gamma\vdash t:\tau_1\times\tau_2}{\Gamma\vdash t=\langle\pi_1(t),\pi_2(t)\rangle:\tau_1\times\tau_2}$$

$$\frac{\Gamma,x:\tau_1\vdash t:\tau\qquad\Gamma\vdash t_1:\tau_1}{\Gamma\vdash(\lambda x:\tau_1.t)(t_1)=t[{}^{t_1}/_{x}]:\tau}\qquad\frac{\Gamma\vdash t:\tau_1\to\tau}{\Gamma\vdash t=\lambda x:\tau_1.t(x):\tau_1\to\tau}\ (x\notin\mathrm{FV}(t))$$

$$\frac{\Gamma\vdash t:\tau_1\to\tau\qquad\Gamma\vdash t_1=t_1':\tau_1}{\Gamma\vdash t(t_1)=t(t_1'):\tau}\qquad\frac{\Gamma,x:\tau_1\vdash t=t':\tau}{\Gamma\vdash\lambda x:\tau_1.t=\lambda x:\tau_1.t':\tau_1\to\tau}$$

$$\frac{\Gamma\vdash\bot_0:0\qquad\Gamma\vdash t:\tau}{\Gamma\vdash\bot_\tau=t:\tau}$$

$$\frac{\Gamma\vdash t:\tau_j\qquad\Gamma,x_i:\tau_i\vdash t_i:\tau\quad(i=1,2)}{\Gamma\vdash\delta(\iota_j(t),x_1.t_1,x_2.t_2)=t_j[{}^{t}/_{x_j}]:\tau}\ (j=1,2)\qquad\frac{\Gamma\vdash t:\tau_1+\tau_2\qquad\Gamma,x:\tau_1+\tau_2\vdash t':\tau}{\Gamma\vdash\delta(t,x_1.t'[{}^{\iota_1(x_1)}/_{x}],x_2.t'[{}^{\iota_2(x_2)}/_{x}])=t'[{}^{t}/_{x}]:\tau}$$

Figure 2. Equational theory of the typed lambda calculus with sums.

$$I[\![0]\!]\ =\ 0$$
$$I[\![\tau+\tau']\!]\ =\ I[\![\tau]\!]+I[\![\tau']\!]$$

This interpretation extends to contexts in the usual manner:

$$I[\![x_1:\tau_1,\dots,x_n:\tau_n]\!]=I[\![\tau_1]\!]\times\dots\times I[\![\tau_n]\!]$$

We write $I[\![\Gamma\vdash t:\tau]\!]$ for the morphism $I[\![\Gamma]\!]\to I[\![\tau]\!]$ in S interpreting the judgement $\Gamma\vdash t:\tau$.

Syntactic BiCCC. The syntactic BiCCC induced by the type theory has objects given by types and morphisms $\tau_1\to\tau_2$ given by equivalence classes $[x:\tau_1\vdash t:\tau_2]$ of derivable judgements under the equivalence identifying $(x:\tau_1\vdash t:\tau_2)$ and $(x':\tau_1\vdash t':\tau_2)$ iff the judgement $x:\tau_1\vdash t=t'[{}^{x}/_{x'}]:\tau_2$ is derivable in the equational theory. Composition is by substitution

$$[x':\tau_2\vdash t':\tau_3]\circ[x:\tau_1\vdash t:\tau_2]\ =\ [x:\tau_1\vdash t'[{}^{t}/_{x'}]:\tau_3]$$

with identities given by $[x:\tau\vdash x:\tau]$.

3 BiCCCs of Grothendieck relations

The class of categorical models of the typed lambda calculus with sums needed for establishing the extensional normalisation result is that given by BiCCCs of Grothendieck relations [16]. These are categories defined over a site, a small category $\mathbb{C}$ with a Grothendieck topology K, equipped with an arity functor $s:\mathbb{C}\to S$ into a BiCCC. They consist of objects (A,R) where A is an object of S and R is a Grothendieck relation of arity s, and have morphisms $(A,R)\to(A',R')$ given by morphisms $A\to A'$ in S that preserve the relations.

We recall the formal definitions.

DEFINITION 3.1. *Given a small category $\mathbb{C}$, a (basis for) a* Grothendieck topology K *on $\mathbb{C}$, is given by associating to each object a of $\mathbb{C}$ a collection $K(a)$ of covers of a (a family of morphisms in $\mathbb{C}$ with codomain a), satisfying the following conditions:*

(*Identity*) *For every $a\in|\mathbb{C}|$, $K(a)$ contains the family consisting of the identity morphism on a.*

(*Stability*) *For every* $\{\phi_i : a_i \to a\}_{i\in I} \in K(a)$, *and morphism* $\psi : b \to a$, *there exists a family* $\{\phi'_j : b_i \to b\}_{j\in J} \in K(b)$ *such that every* $\psi \circ \phi'_j : b_j \to a$ *factors through some* ϕ_i (i.e., *for every* $j \in J$ *there exist* $i \in I$ *and* $\gamma_{ij} : b_j \to a_i$ *such that* $\psi \circ \phi'_j = \phi_i \circ \gamma_{ij}$).

(*Transitivity*) *For every* $\{\phi_i : a_i \to a\}_{i\in I} \in K(a)$, *and for every* $\{\gamma_{ij}\}_{j\in J_i} \in K(a_i)$ $(i \in I)$, *the family* $\{\phi_i \circ \gamma_{ij}\}_{i\in I, j\in J_i} \in K(a)$.

A small category together with a Grothendieck topology on it is called a site.

DEFINITION 3.2. *For a site* $(\mathbb{C}, K)$ *and a functor* $s : \mathbb{C} \to S$, *a* $(\mathbb{C}, K)$-Grothendieck relation R *of arity* s *over* $A \in |S|$ *is a family* $\{ R(c) \subseteq S(s(c), A) \}_{c\in|\mathbb{C}|}$ *with the following two properties.*

(*Monotonicity*) *For every* $\psi : c' \to c$ *in* $\mathbb{C}$, *and every* $x : s(c) \to A$ *in* S, *if* $x \in R(c)$, *then* $x \circ s(\psi) \in R(c')$.

(*Local character*) *For every cover* $\{ \phi_i : c_i \to c \}_{i\in I} \in K(c)$ *and for every* $x : s(c) \to A$ *in* S, *if* $x \circ s(\phi_i) \in R(c_i)$ *for all* $i \in I$, *then* $x \in R(c)$.

DEFINITION 3.3. *For a site* $(\mathbb{C}, K)$ *and a functor* $s : \mathbb{C} \to S$, *the* category of Grothendieck relations $\underline{G}(\mathbb{C}, K, s)$ *is defined as follows: objects are pairs* (A, R) *consisting of an object* A *in* S *and a* $(\mathbb{C}, K)$*-Grothendieck relation* R *of arity* s *over* A; *morphisms* $(A, R) \to (A', R')$ *are morphisms* $f : A \to A'$ *in* S *such that for all objects* c *in* $\mathbb{C}$ *and morphisms* $x : s(c) \to A$ *in* $R(c)$, *the composite* $f \circ x$ *is in* $R'(c)$.

We have the following important result; see [16] for details.

PROPOSITION 3.4. *For a site* $(\mathbb{C}, K)$ *and a functor* $s : \mathbb{C} \to S$ *into a bicartesian closed category, the category of Grothendieck relations* $\underline{G}(\mathbb{C}, K, s)$ *is bicartesian closed and the forgetful functor* $\underline{G}(\mathbb{C}, K, s) \to S$ *preserves the bicartesian closed structure.*

The FUNDAMENTAL LEMMA OF GROTHENDIECK LOGICAL RELATIONS follows as a corollary.

LEMMA 3.5 (FUNDAMENTAL LEMMA). *For a family of Grothendieck relations* $\langle (I[\![\theta]\!], R_\theta) \rangle_\theta$ *in* $\underline{G}(\mathbb{C}, K, s : \mathbb{C} \to S)$ *indexed by base types, let* $\langle (I[\![\tau]\!], R_\tau) \rangle_\tau$ (*resp.* $\langle (I[\![\Gamma]\!], R_\Gamma) \rangle_\Gamma$) *be the family of Grothendieck relations indexed by types* (*resp. contexts*) *induced by the bicartesian closed structure of* $\underline{G}(\mathbb{C}, K, s)$. *Then, the interpretation of terms* $I[\![\Gamma \vdash t : \tau]\!] : I[\![\Gamma]\!] \to I[\![\tau]\!]$ *in* S *are morphisms* $(I[\![\Gamma]\!], R_\Gamma) \to (I[\![\tau]\!], R_\tau)$ *in* $\underline{G}(\mathbb{C}, K, s)$.

3.1 Basic lemma

Following the analysis of [14] we give a BASIC LEMMA that provides the proof-skeleton for both the definability result of [16] and the extensional normalisation result (Theorem 5.1) of this paper.

LEMMA 3.6 (BASIC LEMMA). *Consider a site* $(\mathbb{C}, K)$, *a functor* $s : \mathbb{C} \to S$ *into a BiCCC, and an interpretation* I *of base types in* S.

Let $\langle (I[\![\tau]\!], L_\tau) \rangle_\tau$ *and* $\langle (I[\![\tau]\!], U_\tau) \rangle_\tau$ *be two families of Grothendieck relations in* $\underline{G}(\mathbb{C}, K, s)$ *indexed by types such that*

$$\begin{array}{ll} L_0 = \bot & U_1 = \top \\ L_{\sigma\times\tau} \subseteq L_\sigma \wedge L_\tau & U_\sigma \wedge U_\tau \subseteq U_{\sigma\times\tau} \\ L_{\sigma+\tau} \subseteq L_\sigma \vee L_\tau & U_\sigma \vee U_\tau \subseteq U_{\sigma+\tau} \\ L_{\sigma\to\tau} \subseteq U_\sigma \supset L_\tau & L_\sigma \supset U_\tau \subseteq U_{\sigma\to\tau} \end{array}$$

For a family of Grothendieck relations $\langle (I[\![\theta]\!], R_\theta) \rangle_\theta$ *in* $\underline{G}(\mathbb{C}, K, s)$ *indexed by base types, let* $\langle (I[\![\tau]\!], R_\tau) \rangle_\tau$ *be the family of Grothendieck relations indexed by types induced by the bicartesian closed structure of* $\underline{G}(\mathbb{C}, K, s)$.

If $L_\theta \subseteq R_\theta \subseteq U_\theta$ *for all base types* θ, *then*

1. $L_\tau \subseteq R_\tau \subseteq U_\tau$ *for all types* τ, *and thus*

2. *for all terms* $\Gamma \vdash t : \tau$ $(\Gamma = x_1 : \tau_1, \ldots, x_n : \tau_n)$ *and tuples* $a_i : s(c) \to I[\![\tau_i]\!]$ *in* $L_{\tau_i}(c)$ $(1 \le i \le n, c \in |\mathbb{C}|)$, *we have that* $I[\![\Gamma \vdash t : \tau]\!] \circ \langle a_1, \ldots, a_n \rangle : s(c) \to I[\![\tau]\!]$ *is in* $U_\tau(c)$.

The first part of the lemma follows by induction on types using the closure properties of the hypothesis and the functoriality of the categorical type constructors; the second part is a consequence of the first and the FUNDAMENTAL LEMMA.

4 Normal forms

We present a notion of η-long β-normal form for the typed lambda calculus with sums. The overall definition, which is given in Figure 3, depends on four mutual inductively defined entailment systems: $\vdash_{M_0}$ (pure neutral terms), $\vdash_M$ (neutral terms), $\vdash_{N_0}$ (pure normal terms), and $\vdash_N$ (normal terms). The pure neutral terms are essentially given as in the typed lambda calculus; whilst the neutral terms are obtained from these by closing under discriminators. The pure normal terms are essentially given as in the typed lambda calculus with the addition of the sum injections, and normal terms are obtained by closing under discriminators with respect to pure neutral terms. The unique neutral and normal term in an inconsistent context (*viz.*, a context Γ in which the judgement $\Gamma \vdash \bot_0 : 0$ is derivable) of type τ is $\bot_\tau$.

The definition of normal forms has been designed guided by that of [1] (of which the ones here are syntactic counterparts) and by making sure that the interpretations of neutral and normal terms provide Grothendieck relations satisfying the hypothesis of the BASIC LEMMA (see Section 3).

Note that there are syntactically different, but semantically equivalent normal forms; like the following ones:

$$\begin{array}{l}\lambda g : \theta \to \theta_1 + \theta_2 . \lambda h : \theta \to \theta_3 + \theta_4 . \lambda x : \theta . \\ \quad \delta(\ gx\ ,\ x_1.\iota_1\langle\rangle\ ,\ x_2.\delta(hx, y_1.\iota_2\langle\rangle, y_2.\iota_1\langle\rangle)\ )\end{array} \tag{3}$$

and

$$\begin{array}{l}\lambda g : \theta \to \theta_1 + \theta_2 . \lambda h : \theta \to \theta_3 + \theta_4 . \lambda x : \theta . \\ \quad \delta(\ hx\ ,\ y_1.\delta(gx, x_1.\iota_1\langle\rangle, x_2.\iota_2\langle\rangle)\ ,\ y_2.\iota_1\langle\rangle\ )\end{array} \tag{4}$$

which differ only in the order in which the case analysis is performed. This situation is formalised by the relation $\approx$ in the definition below; on which the side conditions $(\mathbf{B})$ and $(\mathbf{C})$ of Figure 3, allowing the closure under discriminators of normal terms, depend.

DEFINITION 4.1. *We let* $\approx$ *be the least congruence such that*

$$\begin{array}{l}\delta(M, x.\delta(M_1, x_1.N_1, x_2.N_2), y.N) \\ \quad \approx \delta(M_1, x_1.\delta(M, x.N_1, y.N), x_2.\delta(M, x.N_2, y.N))\end{array}$$

$$\begin{array}{l}\delta(M, y.N, x.\delta(M_1, x_1.N_1, x_2.N_2)) \\ \quad \approx \delta(M_1, x_1.\delta(M, y.N, x.N_1), x_2.\delta(M, y.N, x.N_2))\end{array}$$

where $x \notin FV(M_1)$ *and* $x_i \notin FV(M)$ $(i = 1, 2)$, *and*

$$\frac{}{\Gamma, x:\tau, \Gamma' \vdash_{M_0} x:\tau}$$

$$\frac{\Gamma \vdash_{M_0} M:\tau_1 \times \tau_2}{\Gamma \vdash_{M_0} \pi_i(M):\tau_i}\ (i=1,2)$$

$$\frac{\Gamma \vdash_{M_0} M:\tau_1 \to \tau \qquad \Gamma \vdash_{N_0} N:\tau_1}{\Gamma \vdash_{M_0} M(N):\tau}$$

$$\frac{\Gamma \vdash_{M_0} M:\tau}{\Gamma \vdash_{M} M:\tau}$$

$$\frac{}{\Gamma \vdash_{M} \bot_\tau:\tau}\ (\Gamma \text{ inconsistent})$$

$$\frac{\Gamma \vdash_{M_0} M:\tau_1+\tau_2 \qquad \Gamma, x_i:\tau_i \vdash_{M} M_i:\tau\ (i=1,2)}{\Gamma \vdash_{M} \delta(M, x_1.\,M_1, x_2.\,M_2):\tau}$$

$$\frac{\Gamma \vdash_{M_0} M:\theta}{\Gamma \vdash_{N_0} M:\theta}\ (\theta \text{ a base type})$$

$$\frac{}{\Gamma \vdash_{N_0} \langle\rangle:1} \qquad \frac{\Gamma \vdash_{N_0} N_i:\tau_i\ (i=1,2)}{\Gamma \vdash_{N_0} \langle N_1, N_2\rangle:\tau_1 \times \tau_2}$$

$$\frac{\Gamma \vdash_{N_0} N:\tau_i}{\Gamma \vdash_{N_0} \iota_i^{\tau_1,\tau_2}(N):\tau_1+\tau_2}\ (i=1,2)$$

$$\frac{\Gamma \vdash_{N_0} N:\tau}{\Gamma \vdash_{N} N:\tau}$$

$$\frac{\Gamma, x:\tau_1 \vdash_{N} N:\tau}{\Gamma \vdash_{N_0} \lambda x:\tau_1.\,N:\tau_1 \to \tau}\ \left(\ x \in FV(C) \text{ for all } C \in \text{Guards}(N)\ \right) \qquad (\mathbf{A})$$

$$\frac{}{\Gamma \vdash_{N} \bot_\tau:\tau}\ (\Gamma \text{ inconsistent})$$

$$\frac{\Gamma \vdash_{M_0} M:\tau_1+\tau_2 \qquad \Gamma, x_i:\tau_i \vdash_{N} N_i:\tau\ (i=1,2)}{\Gamma \vdash_{N} \delta(M, x_1.\,N_1, x_2.\,N_2):\tau}\ \left(\begin{array}{l} M \not\approx C \text{ for all } C \in \bigcup_{i=1,2} \text{Guards}(x_i.\,N_i) \\ N_1 \not\approx N_2 \text{ whenever } x_1 \notin FV(N_1) \text{ and } x_2 \notin FV(N_2) \end{array}\right) \qquad \begin{array}{r}(\mathbf{B})\\(\mathbf{C})\end{array}$$

$$\text{Guards}(N) \overset{\text{def}}{=} \begin{cases} \{\,M\,\} \cup \bigcup_{i=1,2} \text{Guards}(x_i.\,N_i) & \text{, if } N = \delta(M, x_1.\,N_1, x_2.\,N_2) \\ \emptyset & \text{, otherwise} \end{cases}$$

$$\text{Guards}(x_i.\,N_i) \overset{\text{def}}{=} \left\{\ C \in \text{Guards}(N_i) \mid x_i \notin FV(C)\ \right\}$$

Figure 3. Neutral and normal terms.
(The context is assumed consistent unless stated otherwise.)

$$\frac{N \approx N'}{\delta(M, x.N, y.N') \approx N}$$

where $x \notin \mathrm{FV}(N)$ *and* $y \notin \mathrm{FV}(N')$.

If desired, unique representatives for normal terms can be chosen. Indeed, in [1] this is done by considering a generalised form of discriminator construct allowing simultaneous case analysis. Alternatively, one could proceed by both fixing a canonical notation for binders and a linear order on pure neutral terms to be respected in nested discriminators. This, we believe, yields unique normal forms. For instance, adopting the canonical notation for binders provided by de Bruijn levels, the normal form for the terms (3) and (4) under the linear order in which $\ell_0(\ell_2)$ precedes $\ell_1(\ell_2)$ is

$$\begin{array}{l}
\lambda \ell_0 : \theta \to \theta_1 + \theta_2 . \lambda \ell_1 : \theta \to \theta_3 + \theta_4 . \lambda \ell_2 : \theta . \\
\quad \delta(\ \ell_0(\ell_2)\ ,\ \ell_3 . \iota_1\langle\rangle\ ,\ \ell_3 . \delta(\ell_1(\ell_2), \ell_4 . \iota_2\langle\rangle, \ell_4 . \iota_1\langle\rangle)\ )
\end{array}$$

Examples. We conclude the section with examples of terms and their normal forms that will help to elucidate the notion.

To grasp the role of the side conditions in Figure 3 note that: condition (**A**) fixes the relative position of abstractions and discriminators; condition (**B**) forbids dead branches (that is, when the same case analysis is performed more than once, and hence becomes redundant); and condition (**C**) forbids the two branches of a discriminator to be the same (as in such case the discriminator is redundant).

EXAMPLE 4.2. 1. *The identity term* $\lambda x : \theta . x$ *of type* $\theta \to \theta$ *is a normal term.*

2. *The identity term* $\lambda x : \theta_1 + \theta_2 . x$ *of type*

$$(\theta_1 + \theta_2) \to (\theta_1 + \theta_2)$$

is not a normal term; its normal form is

$$\lambda x : \theta_1 + \theta_2 . \delta(\ x\ ,\ x_1 . \iota_1(x_1)\ ,\ x_2 . \iota_2(x_2)\ ).$$

3. *The identity term* $\lambda x : (\theta_1 + \theta_2) \times (\theta'_1 + \theta'_2) . x$ *of type* $(\theta_1 + \theta_2) \times (\theta'_1 + \theta'_2) \to (\theta_1 + \theta_2) \times (\theta'_1 + \theta'_2)$ *has two equivalent normal terms:*

$$\begin{array}{l}
\lambda x : (\theta_1 + \theta_2) \times (\theta'_1 + \theta'_2) . \\
\quad \delta(\ \pi_1(x)\ , \\
\quad\quad x_1 . \delta(\ \pi_2(x)\ , \\
\quad\quad\quad\quad x'_1 . \langle \iota_1(x_1), \iota_1(x'_1) \rangle\ , \\
\quad\quad\quad\quad x'_2 . \langle \iota_1(x_1), \iota_2(x'_2) \rangle\ ) \\
\quad\quad x_2 . \delta(\ \pi_2(x)\ , \\
\quad\quad\quad\quad x'_1 . \langle \iota_2(x_2), \iota_1(x'_1) \rangle\ , \\
\quad\quad\quad\quad x'_2 . \langle \iota_2(x_2), \iota_2(x'_2) \rangle\ )
\end{array}$$

and

$$\begin{array}{l}
\lambda x : (\theta_1 + \theta_2) \times (\theta'_1 + \theta'_2) . \\
\quad \delta(\ \pi_2(x)\ , \\
\quad\quad x'_1 . \delta(\ \pi_1(x)\ , \\
\quad\quad\quad\quad x_1 . \langle \iota_1(x_1), \iota_1(x'_1) \rangle\ , \\
\quad\quad\quad\quad x_2 . \langle \iota_2(x_2), \iota_1(x'_1) \rangle\ ) \\
\quad\quad x'_2 . \delta(\ \pi_1(x)\ , \\
\quad\quad\quad\quad x_1 . \langle \iota_1(x_1), \iota_2(x'_2) \rangle\ , \\
\quad\quad\quad\quad x_2 . \langle \iota_2(x_2), \iota_2(x'_2) \rangle\ )
\end{array}$$

4. *The curried identity term* $\lambda x : \theta_1 + \theta_2 . \lambda y : \theta'_1 + \theta'_2 . \langle x, y \rangle$ *of type* $(\theta_1 + \theta_2) \to (\theta'_1 + \theta'_2) \to (\theta_1 + \theta_2) \times (\theta'_1 + \theta'_2)$ *has as unique normal form the term*

$$\begin{array}{l}
\lambda x : \theta_1 + \theta_2 . \\
\quad \delta(\ x\ , \\
\quad\quad x_1 .\ \lambda y : \theta'_1 + \theta'_2 . \\
\quad\quad\quad\quad \delta(\ y\ , \\
\quad\quad\quad\quad\quad y_1 . \langle \iota_1(x_1), \iota_1(y_1) \rangle\ , \\
\quad\quad\quad\quad\quad y_2 . \langle \iota_1(x_1), \iota_2(y_2) \rangle\ ) \\
\quad\quad x_2 .\ \lambda y : \theta'_1 + \theta'_2 . \\
\quad\quad\quad\quad \delta(\ y\ , \\
\quad\quad\quad\quad\quad y_1 . \langle \iota_2(x_2), \iota_1(y_1) \rangle\ , \\
\quad\quad\quad\quad\quad y_2 . \langle \iota_2(x_2), \iota_2(y_2) \rangle\ )
\end{array}$$

EXAMPLE 4.3. *Let* $A = \iota_1(\iota_1\langle\rangle)$, $B = \iota_1(\iota_2\langle\rangle)$, $C = \iota_2(\iota_1\langle\rangle)$, *and* $D = \iota_2(\iota_2\langle\rangle)$ *of type* $(1+1) + (1+1)$.

1. *The term*

$$\lambda x : \theta_1 + \theta_2 . \delta(\ \delta(x, x_1 . A, x_2 . D)\ ,\ y_1 . \iota_2(y_1)\ ,\ y_2 . \iota_1(y_2)\ )$$

is not a normal term because

$$x : \theta_1 + \theta_2 \nvdash_{M_0} \delta(x, x_1 . A, x_2 . D) : (1+1) + (1+1)$$

Its normal form is the term

$$\lambda x : \theta_1 + \theta_2 . \delta(x, x_1 . C, x_2 . B)$$

2. *The term*

$$\begin{array}{l}
\lambda f : \theta \to (\theta_1 + \theta_2) . \lambda x : \theta . \lambda g : \theta \to (\theta'_1 + \theta'_2) . \lambda y : \theta . \\
\quad \delta(\ gy\ ,\ x_1 . \delta(fx, y_1 . A, y_2 . B)\ ,\ x_2 . \delta(gx, z_1 . C, z_2 . D)\ )
\end{array}$$

is not a normal term because condition (**A**) *is not satisfied. Its normal form is the term*

$$\begin{array}{l}
\lambda f : \theta \to (\theta_1 + \theta_2) . \lambda x : \theta . \\
\quad \delta(\ fx\ , \\
\quad\quad y_1 .\ \lambda g : \theta \to (\theta'_1 + \theta'_2) . \\
\quad\quad\quad\quad \delta(\ gx\ , \\
\quad\quad\quad\quad\quad z_1 . \lambda y : \theta . \delta(gy, x_1 . A, x_2 . C)\ , \\
\quad\quad\quad\quad\quad z_2 . \lambda y : \theta . \delta(gy, x_1 . A, x_2 . D)\ )\ , \\
\quad\quad y_2 .\ \lambda g : \theta \to (\theta'_1 + \theta'_2) . \\
\quad\quad\quad\quad \delta(\ gx\ , \\
\quad\quad\quad\quad\quad z_1 . \lambda y : \theta . \delta(gy, x_1 . B, x_2 . C)\ , \\
\quad\quad\quad\quad\quad z_2 . \lambda y : \theta . \delta(gy, x_1 . B, x_2 . D)\ )\ )
\end{array}$$

3. *The term*

$$\begin{array}{l}
\lambda f : ((\theta \to \theta_1 + \theta_2) \to (\theta \to \theta_3 + \theta_4) \to \theta \to (1+1)) \\
\quad\quad\quad\quad \to \theta_5 + \theta_6 . \\
\quad \delta(\ f\big(\lambda g : \theta \to \theta_1 + \theta_2 . \lambda h : \theta \to \theta_3 + \theta_4 . \lambda x : \theta . \\
\quad\quad\quad \delta(\ gx\ , \\
\quad\quad\quad\quad x_1 . \iota_1\langle\rangle\ , \\
\quad\quad\quad\quad x_2 . \delta(\ hx\ ,\ y_1 . \iota_2\langle\rangle\ ,\ y_2 . \iota_1\langle\rangle\ )\ )\big)\ , \\
\quad\quad z_1 . A\ , \\
\quad\quad z_2 . \delta(\ f\big(\lambda g : \theta \to \theta_1 + \theta_2 . \lambda h : \theta \to \theta_3 + \theta_4 . \lambda x : \theta . \\
\quad\quad\quad\quad\quad \delta(\ hx\ , \\
\quad\quad\quad\quad\quad\quad y_1 . \delta(\ gx\ ,\ x_1 . \iota_1\langle\rangle\ ,\ x_2 . \iota_2\langle\rangle\ ), \\
\quad\quad\quad\quad\quad\quad y_2 . \iota_1\langle\rangle\ )\big)\ , \\
\quad\quad\quad z_1 . B\ , \\
\quad\quad\quad z_2 . C\ )\ )
\end{array}$$

does not satisfies condition (**B**) *and so is not a normal term.*

Its normal forms are

$$\begin{array}{l}\lambda f:((\theta\to\theta_1+\theta_2)\to(\theta\to\theta_3+\theta_4)\to\theta\to(1+1))\\ \qquad\to\theta_5+\theta_6.\\ \quad\delta(\ f\big(\lambda g:\theta\to\theta_1+\theta_2.\lambda h:\theta\to\theta_3+\theta_4.\lambda x:\theta.\\ \qquad\quad\delta(\ gx\ ,\\ \qquad\qquad x_1.\iota_1\langle\rangle,\\ \qquad\qquad x_2.\delta(\ hx\ ,\ y_1.\iota_2\langle\rangle\ ,\ y_2.\iota_1\langle\rangle\ )\ )\big)\ ,\\ \qquad z_1.\mathsf{A}\ ,\\ \qquad z_2.\mathsf{C}\ )\end{array}$$

and

$$\begin{array}{l}\lambda f:((\theta\to\theta_1+\theta_2)\to(\theta\to\theta_3+\theta_4)\to\theta\to(1+1))\\ \qquad\to\theta_5+\theta_6.\\ \quad\delta(\ f\big(\lambda g:\theta\to\theta_1+\theta_2.\lambda h:\theta\to\theta_3+\theta_4.\lambda x:\theta.\\ \qquad\quad\delta(\ hx\ ,\\ \qquad\qquad y_1.\delta(\ gx\ ,\ x_1.\iota_1\langle\rangle\ ,\ x_2.\iota_2\langle\rangle\ ),\\ \qquad\qquad y_2.\iota_1\langle\rangle\ )\big)\ ,\\ \qquad z_1.\mathsf{A}\ ,\\ \qquad z_2.\mathsf{C}\ )\end{array}$$

4. *The term*

$$\lambda f:\theta\to 0.\lambda x:\theta+0.\delta(\ x\ ,\ x_1.x_1\ ,\ x_2.\bot_\theta\ )$$

is not normal, as

$$f:\theta\to 0, x:\theta+0, x_1:\theta\not\vdash_N x_1:\theta$$

because the context is inconsistent. The equivalent term

$$\lambda f:\theta\to 0.\lambda x:\theta+0.\delta(\ x\ ,\ x_1.\bot_\theta\ ,\ x_2.\bot_\theta\ )$$

is not normal either because condition (**C**) *is not satisfied. The normal form of these two terms is*

$$\lambda f:\theta\to 0.\lambda x:\theta+0.\bot_\theta$$

EXAMPLE 4.4. *The normal form of the term*

$$\begin{array}{l}\lambda f:\theta_1\to 0.\lambda x:\theta_1+\theta_2.\lambda g:\theta_2\to\theta_1.\lambda y:\theta.\\ \quad\delta(\ x\ ,\ x_1.\bot_{\theta_1}\ ,\ x_2.g(x_2)\ )\end{array}$$

is

$$\begin{array}{l}\lambda f:\theta_1\to 0.\lambda x:\theta_1+\theta_2.\\ \quad\delta(\ x\ ,\ x_1.\bot_{(\theta_2\to\theta_1)\to\theta\to\theta_1}\ ,\ x_2.\lambda g:\theta_2\to\theta_1.\bot_{\theta\to\theta_1}\ )\end{array}$$

5 Extensional normalisation

Following [14], we establish the following extensional-normalisation result.

THEOREM 5.1 (EXTENSIONAL NORMALISATION). *For every term of the typed lambda calculus with sums $\Gamma\vdash t:\tau$ there exists a normal term $\Gamma\vdash_N N:\tau$ such that $\Gamma\vdash t=N:\tau$ is provable in the equational theory of the calculus.*

The proof is along the following lines.

- We define an appropriate syntactic site $(\mathbf{C},\mathsf{K})$ together with an arity functor $I:\mathbf{C}\to S$ into a BiCCC canonically induced by a *stable* interpretation I of base types. (See Section 5.1.)
- We establish that the interpretation of neutral and normal terms define Grothendieck relations in $\underline{G}(\mathbf{C},\mathsf{K},I)$ satisfying the hypothesis of the BASIC LEMMA. (See Section 5.2.)
- As a direct consequence we have the semantic result that for every term $\Gamma\vdash t:\tau$ there exists a normal term $\Gamma\vdash_N N:\tau$ such that $I[\![\Gamma\vdash t:\tau]\!]=I[\![\Gamma\vdash N:\tau]\!]:I[\![\Gamma]\!]\to I[\![\tau]\!]$ in S.
- The syntactic result of Theorem 5.1 follows from the semantic one by embedding the syntactic BiCCC induced by the type theory into a BiCCC in which the sums become stable.

5.1 The syntactic site and its arity functor

The syntactic site. Following [16] in the light of [1], we will use a site of constrained contexts $\Gamma|\Xi$; the intuition is that we consider the context Γ under the constraints Ξ.

DEFINITION 5.2. Constrained contexts *are defined by the following rules*

$$\frac{}{\langle\rangle|\langle\rangle}\qquad\qquad\frac{\Gamma|\Xi}{\Gamma,x:\tau|\Xi,x=_\tau x}$$

$$\frac{\Gamma|\Xi\qquad\Gamma\vdash_{M_0}M:\tau_1+\tau_2}{\Gamma,x:\tau_i|\Xi,\iota_i(x)=_{\tau_1+\tau_2}M}\ (i=1,2)$$

$$\frac{\Gamma|\Xi\qquad\Gamma\vdash t:\tau_1+\tau_2}{\Gamma,x:\tau_i|\Xi,\iota_i(x)=_{\tau_1+\tau_2}t}\ (\Gamma\ \textit{inconsistent})\quad(\textit{i=1,2})$$

where $x\notin\mathrm{dom}(\Gamma)$.

DEFINITION 5.3. *The category* **C** *has objects given by constrained contexts and morphisms* $\Gamma'|\Xi'\to\Gamma|\Xi$ *given by injective renamings* $\rho:\mathrm{dom}(\Gamma)\rightarrowtail\mathrm{dom}(\Gamma')$ *that preserve typing* (*i.e. if* $(x:\tau)\in\Gamma$, *then* $(\rho(x):\tau)\in\Gamma'$) *and constraints* (*i.e. if* $t=_\tau t'\in\Xi$, *then* $t[\rho]=_\tau t'[\rho]\in\Xi'$).

DEFINITION 5.4. *The family of covers* $\mathsf{K}(\Gamma|\Xi)$ *of a constrained context* $\Gamma|\Xi$ *is defined by the following rules:*

$$\frac{}{\emptyset\in\mathsf{K}(\Gamma|\Xi)}\ (\Gamma\ \textit{inconsistent})$$

$$\frac{}{\{\ \mathrm{id}_{\mathrm{dom}(\Gamma)}\ \}\in\mathsf{K}(\Gamma|\Xi)}$$

$$\frac{\{\ \rho_j\ \}_{j\in J}\cup\{\ \rho:\Gamma'|\Xi'\to\Gamma|\Xi\ \}\in\mathsf{K}(\Gamma|\Xi)}{\{\ \rho_j\ \}_{j\in J}\cup\{\ \rho\circ\rho_i:\Gamma'_i|\Xi'_i\to\Gamma|\Xi\ \}_{i=1,2}}$$

where, for $i=1,2$, *the constrained contexts* $\Gamma'_i|\Xi'_i$ *are of the form* $\langle\Gamma',x'_i:\tau_i|\Xi',\iota_i(x'_i)=_{\tau_1+\tau_2}t\rangle$ *and the renamings* ρ_i *are the inclusions* $\Gamma'_i|\Xi'_i\rightarrowtail\Gamma'|\Xi'$.

PROPOSITION 5.5. *The pair* $(\mathbf{C},\mathsf{K})$ *is a site.*

The arity functor. We restrict attention to *stable interpretations* of types; *i.e.*, interpretations I of base types in a BiCCC such that, for all pair of types τ_1 and τ_2, the coproduct $I[\![\tau_1]\!]+I[\![\tau_2]\!]$ is stable under pullbacks.

For a stable interpretation, we define the semantic interpretation of the constrained context $\Gamma|\Xi$ as a subobject of the semantic interpretation of the context Γ.

DEFINITION 5.6. *With respect to a stable interpretation I of base types in a BiCCC, we associate to every constrained context $\Gamma|\Xi$ its interpretation $I[\![\Gamma|\Xi]\!]$ given by the domain of a monomorphism* $m_{\Gamma|\Xi}:I[\![\Gamma|\Xi]\!]\rightarrowtail I[\![\Gamma]\!]$ *inductively defined as follows.*

- $m_{\langle\rangle|\langle\rangle} : 1 \rightarrowtail 1$ *is defined as* id_1.

- $m_{\Gamma,x:\tau|\Xi,x=_\tau x} : I[\![\Gamma|\Xi]\!] \times I[\![\tau]\!] \rightarrowtail I[\![\Gamma]\!] \times I[\![\tau]\!]$ *is defined as* $m_{\Gamma|\Xi} \times \mathrm{id}_{I[\![\tau]\!]}$.

- $m_{\Gamma,x:\tau_i|\Xi,\iota_i(x)=_{\tau_1+\tau_2}t}$:
  $I[\![\Gamma, x:\tau_i|\Xi, \iota_i(x) =_{\tau_1+\tau_2} t]\!] \rightarrowtail I[\![\Gamma]\!] \times I[\![\tau_i]\!]$
  is defined as $\langle m_{\Gamma|\Xi} \circ p_i, q_i\rangle$ *where the following square*

$$\begin{tikzcd}
I[\![\Gamma, x:\tau_i|\Xi, \iota_i(x) =_{\tau_1+\tau_2} t]\!] \arrow[r, "p_i"] \arrow[dd, "q_i"'] & I[\![\Gamma|\Xi]\!] \arrow[d, tail, "m_{\Gamma|\Xi}"] \\
 & I[\![\Gamma]\!] \arrow[d, "I[\![\Gamma\vdash t:\tau_1+\tau_2]\!]"] \\
I[\![\tau_i]\!] \arrow[r, "\coprod_i"'] & I[\![\tau_1]\!] + I[\![\tau_2]\!]
\end{tikzcd}$$

is a pullback.

By stability, the family

$$\{\, I[\![\Gamma, x_i:\tau_i|\Xi, \iota_i(x_i) =_{\tau_1+\tau_2} t]\!] \xrightarrow{p_i} I[\![\Gamma|\Xi]\!] \,\}_{i=1,2}$$

is a coproduct, and for every

$$\Gamma|\Xi = \langle x_1:\tau_1,\ldots,x_n:\tau_n | t_1 =_{\tau_1'} t_1',\ldots,t_n =_{\tau_n'} t_n'\rangle$$

we have an equaliser diagram

$$\begin{tikzcd}
I[\![\Gamma|\Xi]\!] \arrow[r, tail, "m_{\Gamma|\Xi}"] & I[\![\Gamma]\!] \arrow[r, shift left, "\langle I[\![\Gamma\vdash t_i:\tau_i']\!]\rangle_{1\le i\le n}"] \arrow[r, shift right, "\langle I[\![\Gamma\vdash t_i':\tau_i']\!]\rangle_{1\le i\le n}"'] & I[\![\tau_1']\!] \times\ldots\times I[\![\tau_n']\!]
\end{tikzcd}$$

The definition of the arity functor induced by a stable interpretation follows.

DEFINITION 5.7. *With respect to a stable interpretation I of base types in a BiCCC S, the arity functor $I : \mathbf{C} \to S$ is defined as follows.*

On objects: $I(\Gamma|\Xi) \overset{\mathrm{def}}{=} I[\![\Gamma|\Xi]\!]$.

On morphisms: for $\rho : \Gamma'|\Xi' \to \Gamma|\Xi$, we define $I(\rho)$ as the unique map $I[\![\Gamma'|\Xi']\!] \to I[\![\Gamma|\Xi]\!]$ such that

$$\begin{tikzcd}
I[\![\Gamma|\Xi]\!] \arrow[r, tail, "m_{\Gamma|\Xi}"] & I[\![\Gamma]\!] = \Pi_{x\in\mathrm{dom}(\Gamma)} I[\![\Gamma(x)]\!] \\
I[\![\Gamma'|\Xi']\!] \arrow[u, "I(\rho)"] \arrow[r, tail, "m_{\Gamma'|\Xi'}"'] & I[\![\Gamma']\!] = \Pi_{x'\in\mathrm{dom}(\Gamma')} I[\![\Gamma'(x')]\!] \arrow[u, "\langle\pi_{\rho(x)}\rangle_{x\in\mathrm{dom}(\Gamma)}"']
\end{tikzcd}$$

5.2 Extensional-normalisation result

For a stable interpretation I of base types in a BiCCC S the definitions

$$\begin{aligned}
M_\tau(\Gamma|\Xi) &= \{\, I[\![\Gamma\vdash M:\tau]\!] \circ m_{\Gamma|\Xi} \mid \Gamma\vdash_M M:\tau \,\} \\
N_\tau(\Gamma|\Xi) &= \{\, I[\![\Gamma\vdash N:\tau]\!] \circ m_{\Gamma|\Xi} \mid \Gamma\vdash_N N:\tau \,\}
\end{aligned}$$

respectively identify the sets of neutral and normal morphisms in $S(I[\![\Gamma|\Xi]\!], I[\![\tau]\!])$.

PROPOSITION 5.8. *Let I be a stable interpretation of base types in a BiCCC. For all types τ, $(I[\![\tau]\!], M_\tau)$ and $(I[\![\tau]\!], N_\tau)$ are Grothendieck relations in $\underline{\mathrm{G}}(\mathbf{C}, \mathsf{K}, I)$.*

THEOREM 5.9. *The Grothendieck relations of neutral and normal morphisms satisfy the following closure properties.*

$$\begin{array}{llll}
M_0 = \bot & & N_1 = \top & \\
M_{\sigma\times\tau} \subseteq M_\sigma \wedge M_\tau & & N_\sigma \wedge N_\tau \subseteq N_{\sigma\times\tau} & \\
M_{\sigma+\tau} \subseteq M_\sigma \vee M_\tau & & N_\sigma \vee N_\tau \subseteq N_{\sigma+\tau} & \\
M_{\sigma\to\tau} \subseteq N_\sigma \supset M_\tau & & M_\sigma \supset N_\tau \subseteq N_{\sigma\to\tau} &
\end{array}$$

and

$$M_\theta \subseteq N_\theta \qquad (\theta \text{ a base type})$$

The proof of the theorem relies on the next two lemmas; whose proofs embody the algorithmic idea underlying the normalisation program of Section 6.

LEMMA 5.10. *1. For every neutral term $\Gamma\vdash_M M:\tau_1\times\tau_2$ there exist neutral terms $\Gamma\vdash_M M_1:\tau_1$ and $\Gamma\vdash_M M_2:\tau_2$ such that $\Gamma\vdash\pi_i(M) = M_i:\tau_i$ $(i=1,2)$.*

2. For every neutral term $\Gamma\vdash_M M:\tau_1\to\tau$ and normal term $\Gamma\vdash_N N:\tau_1$, there exists a neutral term $\Gamma\vdash_M M':\tau$ such that $\Gamma\vdash M(N) = M':\tau$.

LEMMA 5.11. *1. For every term $\Gamma\vdash_{N_1} C:\tau$ derivable according to the following rules*

$$\frac{\Gamma\vdash_{N_0} N:\tau}{\Gamma\vdash_{N_1} N:\tau}\ (\Gamma \textit{ consistent})$$

$$\frac{\Gamma\vdash_{M_0} M:\tau_1+\tau_2 \qquad \Gamma, x_i:\tau_i\vdash_{N_1} C_i:\tau \quad (i=1,2)}{\Gamma\vdash_{N_1} \delta(M, x_1.C_1, x_2.C_2):\tau}\ (\Gamma \textit{ consistent})$$

$$\frac{}{\Gamma\vdash_{N_1} \bot_\tau:\tau}\ (\Gamma \textit{ inconsistent})$$

there exists a normal term $\Gamma\vdash_N N:\tau$ such that $\Gamma\vdash C = N:\tau$.

2. For every pair of normal terms $\Gamma\vdash_N N_i:\tau_i$ $(i=1,2)$, there exists a normal term $\Gamma\vdash_N N:\tau_1\times\tau_2$ such that $\Gamma\vdash\langle N_1, N_2\rangle = N:\tau_1\times\tau_2$.

3. For every normal term $\Gamma\vdash_N N:\tau_i$ $(i\in\{1,2\})$, there exists a normal term $\Gamma\vdash_N N':\tau_1+\tau_2$ such that $\Gamma\vdash\iota_i(N) = N':\tau_1+\tau_2$.

4. For every normal term $\Gamma, x:\tau_1\vdash_N N_1:\tau$, there exists a normal term $\Gamma\vdash_N N:\tau_1\to\tau$ such that $\Gamma\vdash\lambda x:\tau_1.\,N_1 = N:\tau_1\to\tau$.

Since for $\Gamma = \langle x_1:\tau_1,\ldots,x_n:\tau_n\rangle$ we have that the projection $I[\![\Gamma\vdash x_i:\tau_i]\!]$ $(1\le i\le n)$, is a neutral morphism $I[\![\Gamma]\!]\to I[\![\tau_i]\!]$ in $M_{\tau_i}(\Gamma|\Delta_\Gamma)$ where $\Delta_\Gamma = \langle x_1 =_{\tau_1} x_1,\ldots,x_n =_{\tau_n} x_n\rangle$, it follows from Theorem 5.9 and the BASIC LEMMA that the interpretation

$$I[\![\Gamma\vdash t:\tau]\!] = I[\![\Gamma\vdash t:\tau]\!] \circ \langle I[\![\Gamma\vdash x_1:\tau_1]\!],\ldots,I[\![\Gamma\vdash x_n:\tau_n]\!]\rangle$$

of the term $\Gamma\vdash t:\tau$ is a normal morphism $I[\![\Gamma]\!]\to I[\![\tau]\!]$ in $N_\tau(\Gamma|\Delta_\Gamma)$. Thus we have the following corollary.

COROLLARY 5.12. *Let I be a stable interpretation of base types in a BiCCC S. For every term $\Gamma \vdash t : \tau$, there exists a normal term $\Gamma \vdash_N N : \tau$ such that $I[\![\Gamma \vdash t : \tau]\!] = I[\![\Gamma \vdash N : \tau]\!] : I[\![\Gamma]\!] \to I[\![\tau]\!]$ in S.*

Theorem 5.2 is obtained from this corollary by producing a BiCCC embedding $T \hookrightarrow \widetilde{T}$, of the syntactic BiCCC T into a BiCCC $\widetilde{T}$, mapping sums to stable sums, and considering the canonical interpretation of types in $\widetilde{T}$.

6 Type-Directed Partial Evaluation with sums

We show how to build a normalisation algorithm based on Type-Directed Partial Evaluation that puts terms in the normal form of Section 4. In fact, we use a version of TDPE written for the language *Objective Caml* (see [2]) slightly modified to allow the use of certain powerful control operators.

An interesting point of this work is that the optimisations we introduce will be usable in some other cases of partial evaluation. Here, however, we are only concerned in normalising functional programs corresponding to terms in the typed lambda calculus with binary sums with respect to the equational theory of the calculus. In particular, note that the normalisation of a program may have a different observational semantics (within the programming language that is) than the original program; as, for instance, the evaluation order may not be preserved.

6.1 The original TDPE

We recall the basic elements of the original TDPE algorithm. For details see [5, 6].

NBE is based on an η-expansion of the term using a two-level language, which in our case is defined as follows:

$$
\begin{array}{rcll}
t & ::= & s & \text{(Static terms)} \\
 & | & d & \text{(Dynamic terms)} \\[1ex]
s & ::= & x & \\
 & | & \langle\rangle \mid \mathsf{pair}\langle t_1, t_2\rangle \mid \pi_1(t) \mid \pi_2(t) & \\
 & | & \lambda x.\, t \mid t_1 @ t_2 & \\
 & | & \iota_1(t) \mid \iota_2(t) \mid \delta(t, x_1.\, t_1, x_2.\, t_2) & \\[1ex]
d & ::= & \underline{x} & \\
 & & \underline{\langle\rangle} \mid \underline{\mathsf{pair}}\langle t_1, t_2\rangle \mid \underline{\pi_1}(t) \mid \underline{\pi_2}(t) & \\
 & | & \underline{\lambda}\underline{x}.\, t \mid t_1 \underline{@} t_2 & \\
 & | & \underline{\iota_1}(t) \mid \underline{\iota_2}(t) \mid \underline{\delta}(t, x_1.\, t_1, x_2.\, t_2) &
\end{array}
$$

where x (resp. $\underline{x}$) ranges over (a countable set of) *static* (resp. *dynamic*) variables. The s-terms are said to be *static* and the d-terms to be *dynamic*. In implementations, dynamic terms are often represented by data structures, whereas static terms are values of the language itself.

The TDPE algorithm without let insertion is presented in Figure 4. It inductively defines two functions for each type. One, written $\downarrow$, is called `reify` and the other one, written $\uparrow$, is called `reflect`. The functions $\downarrow$ and $\uparrow$ are basically two-level η-expansions.

To normalise a static value V of type τ, first apply the function $\downarrow^\tau$ to V, and then reduce the static part, obtaining a fully dynamic term in normal form. The reduction of static parts is performed automatically by the abstract machine of the programming language. The control operators shift and reset are used to place δ in the right place in the final result.

Shift and reset. We briefly explain the way in which shift and reset work with an example. For details see [8, 9].

The operator reset is used to delimit a context of evaluation, and shift abstracts this context in a function. Thus the term

$$1 + \mathsf{reset}\ (\mathbf{2} + \mathsf{shift}\ \mathsf{c}.\ (3\ +\ (\mathbf{c}\ 4)\ +\ (\mathbf{c}\ 5)))$$

reduces to $1\ +\ 3\ +\ (\mathbf{2} + 4) + (\mathbf{2} + 5)$. Indeed, the operator reset delimits the context $2\ +\ \square$, which is abstracted into the function c; the values 4 and 5 are successively inserted in this context and the resulting expression is evaluated.

6.2 Producing normal terms

The original TDPE algorithm without let insertion produces terms following the inference system of Figure 3 without taking into account the side conditions $(\mathbf{A})$, $(\mathbf{B})$, $(\mathbf{C})$ there in.

For example, the evaluation of the term

$$\lambda z.\lambda x.\lambda f.\delta(\ (f @ x)\ ,\ x_1.\ (\lambda y.\iota_1(y))\ ,\ x_2.\ (\lambda y.f @ z)\ )$$

of type

$$\theta \to \theta \to (\theta \to \theta_1 + \theta_2) \to \theta_1 \to (\theta_1 + \theta_2)$$

yields the term

$$
\begin{array}{l}
\underline{\lambda}\underline{z}.\underline{\lambda}\underline{x}.\underline{\lambda}\underline{f}. \\
\quad \underline{\delta}(\ (\underline{f}\ \underline{@}\ \underline{x})\ , \\
\qquad \underline{x}_1.\ (\underline{\lambda}\underline{y}.\underline{\iota_1}(\underline{y}))\ , \\
\qquad \underline{x}_2.\ (\underline{\lambda}\underline{y}.\underline{\delta}\Big((\underline{f}\ \underline{@}\ \underline{z}),\ \underline{y}_1.\ \underline{\iota_1}(\underline{y}_1),\ \underline{y}_2.\ \underline{\iota_2}(\underline{y}_2)\Big))\ )
\end{array}
\tag{5}
$$

which does not satisfy condition $(\mathbf{A})$ since $\underline{f}\ \underline{@}\ \underline{z}$ does not contain the variable $\underline{y}$.

In the following, we propose three modifications of TDPE to take the conditions $(\mathbf{A})$, $(\mathbf{B})$, $(\mathbf{C})$ into account.

6.2.1 *Remove dead branches*

To ensure the condition $(\mathbf{B})$ we will use the following derivable equations:

$$\delta(\mathbf{t},\ x.\ \boldsymbol{\delta}(\mathbf{t},\ \mathbf{x_1}.\ \mathbf{t_1},\ \mathbf{x_2}.\ \mathbf{t_2}),\ y.\ t_0)\ =\ \delta(\mathbf{t},\ x.\ t_1[{}^{x}/{x_1}],\ y.\ t_0)$$

$$\delta(\mathbf{t},\ x.\ t_0,\ y.\ \boldsymbol{\delta}(\mathbf{t},\ \mathbf{x_1}.\ \mathbf{t_1},\ \mathbf{x_2}.\ \mathbf{t_2}))\ =\ \delta(\mathbf{t},\ x.\ t_0,\ y.\ t_2[{}^{y}/{x_2}])$$

To apply these transformations, notice that the residual program is an abstract syntax tree built in depth-first manner, from left to right, the evaluation being done in call by value. The idea consists in maintaining a global table accounting for the conditional branches in the path from the root of the residual program to the current point of construction. This table associates a flag (L or R) and a variable

$$\begin{array}{rcl}
\downarrow^{\theta} V & = & V \qquad (\theta \text{ a base type}) \\
\downarrow^{1} V & = & \underline{\langle\rangle} \\
\downarrow^{\sigma\to\tau} V & = & \mathsf{let}\ \underline{x} \text{ be a fresh variable } \mathsf{in}\ \underline{\lambda}\underline{x}.\,\mathsf{reset}(\downarrow^{\tau} (V\ @\uparrow^{\sigma} \underline{x})) \\
\downarrow^{\tau_1\times\tau_2} V & = & \underline{\mathrm{pair}}\langle\downarrow^{\tau_1} (\pi_1(V)), \downarrow^{\tau_2} (\pi_2(V))\rangle \\
\downarrow^{\tau_1+\tau_2} V & = & \delta\left(V, x_1.\,\underline{\iota_1}(\downarrow^{\tau_1} x_1), x_2.\,\underline{\iota_2}(\downarrow^{\tau_2} x_2)\right) \\
 & & \\
\uparrow^{\theta} M & = & M \qquad (\theta \text{ a base type}) \\
\uparrow^{1} M & = & \langle\rangle \\
\uparrow^{\tau\to\sigma} M & = & \lambda x.\uparrow^{\sigma} (M\ \underline{@}\downarrow^{\tau} x) \\
\uparrow^{\sigma_1\times\sigma_2} M & = & \mathrm{pair}\langle\uparrow^{\sigma_1} (\underline{\pi_1}(M)), \uparrow^{\sigma_2} (\underline{\pi_2}(M))\rangle \\
\uparrow^{\sigma_1+\sigma_2} M & = & \mathsf{let}\ \underline{x}_1 \text{ and } \underline{x}_2 \text{ be fresh variables} \\
 & & \mathsf{in\ shift}\ c.\ \underline{\delta}(\ M\ ,\ \underline{x}_1.\,\mathsf{reset}(c\ @\ \iota_1(\uparrow^{\sigma_1} \underline{x}_1))\ ,\ \underline{x}_2.\,\mathsf{reset}(c\ @\ \iota_2(\uparrow^{\sigma_2} \underline{x}_2))\ )
\end{array}$$

Figure 4. Type-directed partial evaluation without let insertion.

to an expression in the following way:

```
$\uparrow^{\sigma_1+\sigma_2} M$ =
    if $M$ is globally associated to (L, $\underline{z}$) modulo $\approx$
    then $\iota_1(\uparrow^{\sigma_1} \underline{z})$
    else if $M$ is globally associated to (R, $\underline{z}$) modulo $\approx$
        then $\iota_2(\uparrow^{\sigma_2} \underline{z})$
        else shift $c$.
                let $\underline{x}_1$ and $\underline{x}_2$ be fresh variables,
                    associate $M$ to $(\mathrm{L}, \underline{x}_1)$ while computing
                      $n_1 = \mathsf{reset}(\iota_1(\uparrow^{\sigma_1} \underline{x}_1))$,
                    associate $M$ to $(\mathrm{R}, \underline{x}_2)$ while computing
                      $n_2 = \mathsf{reset}(\iota_2(\uparrow^{\sigma_2} \underline{x}_2))$,
                in $\underline{\delta}(\ M\ ,\ \underline{x}_1.\,n_1,\ \underline{x}_2.\,n_2)$
```

(Note that the test of global association is done modulo $\approx$; this is explained in the next section.)

This optimisation, associated with let insertion and other memoization techniques, has been used for building a fully lazy partial evaluator from TDPE; see [3].

6.2.2 *Forbid redundant discriminators*

To enforce the condition $(\mathbf{C})$, we write a test of membership of free variables and implement a test of the congruence $\approx$ of two normal terms. There are different ways in which to implement this latter test. One method is to define, in a mutually recursive fashion, three tests $\approx_{M_0}$, $\approx_{N_0}$, and $\approx_N$ that respectively test the equivalence between pure neutral terms, pure normal terms, and normal terms along the following lines.

- The test $\approx_{M_0}$ is done by structural recursion, using the test $\approx_N$ in the case of applications.
- The test $\approx_{N_0}$ is done by structural recursion, using the test $\approx_N$ in the case of abstractions.
- The test $N \approx_N N'$ inspects the set of paths p given by all possible branchings in discriminators containing the guards of N, and collects the sequence of guards together with the end pure normal form N_p. For each of these paths p, it proceeds according to the following sub-test: if N' is a pure normal term then check whether $N_p \approx_{N_0} N'$, otherwise, for N' of the form $\delta(M', x.N'_1, y.N'_2)$, there are three possibilities: if M' is in the path p up to $\approx_{M_0}$ and the path branches left (resp. right) the sub-test is repeated for N'_1 (resp. N'_2) instead of N', however, if M' is not in the path p up to $\approx_{M_0}$, the sub-test is repeated for both N'_1 and N'_2 instead of N', succeeding if both of these sub-tests do.

Note that condition $(\mathbf{C})$ does not need to be checked recursively within the branches of the discriminator; since, as TDPE builds the normal form in depth-first manner, it is known that each branch satisfies it.

6.2.3 *Fix the relative positions of abstractions and discriminators*

To obtain terms in normal form, we must also check the condition $(\mathbf{A})$ concerning the guards of abstractions.

For that, let us look at the example in (5). We want to introduce the $\underline{\delta}(\underline{f}\ \underline{@}\ \underline{z}, \ldots)$ above $\underline{\lambda y}\ \ldots$ However a shift always returns to the preceding reset. Thus, it would be necessary to be able to name each reset and to choose the best one at the time of introducing the $\underline{\delta}$. This is what the control operators cupto/set, introduced in [19], allow us to do.

Set and cupto. The control operators set and cupto are very powerful, and generalise exceptions and continuations. Here we give the idea of how they work on an example. For details see [19, 18].

The operators set/cupto rely on the concept of *prompt*, that allows marking the occurrences of set. New prompts can be created upon request. For two prompts p_1 and p_2, one can write an expression like the following one

$$1 + \mathsf{set}\ p_1\ \mathsf{in}\ \mathbf{2} + \mathsf{set}\ p_2\ \mathsf{in}\ \mathbf{3} + \mathsf{cupto}\ p_1\ \mathsf{as}\ c\ \mathsf{in}\ (4\ +\ (\mathbf{c}\ 5))$$

which evaluates to $1\ +\ 4\ +\ (\mathbf{2}\ +\ \mathbf{3} + 5)$.

Application to TDPE. To use set/cupto to address the problem of fixing the relative position of abstractions and discriminators, we must create a new prompt with each created dynamic $\underline{\lambda}$. Further, we maintain a global list associating to each prompt a set of variables. To introduce a new δ, we look for all the free variables of its condition, and look in this list for the last prompt introduced to which one of these variables is associated. Since the term is built in depth first manner and from left to right, one obtains a closed term.

We thus modify the algorithm of TDPE in the following way:

```
$\downarrow^{\sigma\to\tau} V =$   let $\underline{x}$ be a fresh variable and $p$ be a new prompt
               in $\underline{\lambda}\underline{x}.$ set $p$ in $\downarrow^{\tau} (V\ @\uparrow^{\sigma} \underline{x})$

$\uparrow^{\sigma_1+\sigma_2} M =$   let $m$ be the best prompt for $M$
                 in cupto $m$ as c
                    in let $\underline{x}_1$ and $\underline{x}_2$ be fresh variables,
                           $n_1 =$ set $m$ in $(c\ @\ \iota_1(\uparrow^{\sigma_1} \underline{x}_1))$,
                           $n_2 =$ set $m$ in $(c\ @\ \iota_2(\uparrow^{\sigma_2} \underline{x}_2))$,
                       in $\underline{\delta}(M, \underline{x}_1.n_1, \underline{x}_2.n_2)$
```

The complete algorithm is presented in Figure 5.

6.2.4 *Two examples*

1. We show the application of the optimised partial evaluator to the example of the introduction.

```
# let example f g z x =
  match (match x with
           Left x1 -> Left z
         | Right x2 -> Right (g z))
  with
    Left y1 -> (f (g y1))
  | Right y2 -> (f y2);;

val example :
  ('a -> 'b) -> ('c -> 'a) ->
    'c -> ('d, 'e) sum -> 'b = <fun>
```

To use a type directed partial evaluator, one has to pass to the evaluator a representation of the type of the term to be evaluated. There are different approaches to representing types. Here we use the approach pioneered by Filinski, who represents types via combinators, so that

```
('a -> 'b) -> ('c -> 'a) -> 'c -> ('d, 'e) sum -> 'b
```

becomes

```
  ((base **-> base) **->
    ((base **-> base) **->
      (base **-> ((sum (base, base)) **-> base))))
```

which we abbreviate below as `combinatortype`.

The application of the partial evaluator based on shift/reset yields:

```
# tdpesr combinatortype example;;
- : Shiftreset.ans =
        (fun v0 v1 v2 v3 ->
           (match v3 with
            | Left  v4 -> (v0 (v1 v2))
            | Right  v4 -> (v0 (v1 v2))))
```

whilst the partial evaluator based on cupto produces the desired result:

```
# tdpecupto combinatortype example;;
- : Normal.normal =
     (fun f g z x -> (f (g z)))
```

2. We now test the partial evaluator on an example suggested to us by Filinski.

For every endofunction f on a two-element set, the identity $f^3 = f$ holds. We give a proof of this fact in the equational theory of the typed lambda calculus with sums by establishing the identity

$$\begin{aligned}&\lambda f:(1+1)\to(1+1).\lambda x:1+1.f(f(fx))\\ &\quad = \lambda f:(1+1)\to(1+1).f\end{aligned}$$

in the equational theory using the partial evaluator.

Defining

```
 # let fff f x = f (f (f x));;
val fff : ('a -> 'a) -> 'a -> 'a = <fun>
```

and

```
 # let bool = sum (unit,unit);;
```

we want that the normalisation of `fff` of type

```
          (bool -> bool) -> bool -> bool
```

is (the normal form of) the identity.

Normalising `fff` by the TDPE with shift/reset gives the following (uninformative) result.

```
 # tdpesr
     ((bool **-> bool) **-> (bool **-> bool)) fff;;
- : Shiftreset.ans = (fun v0 v1 ->
(match v1 with
 | Left  v2 ->
   (match (v0 (Left  ())) with
    | Left  v10 ->
      (match (v0 (Left  ())) with
       | Left  v14 -> (match (v0 (Left  ())) with
                       | Left  v16 -> (Left  ())
                       | Right v16 -> (Right ()))
       | Right v14 -> (match (v0 (Right ())) with
                       | Left  v15 -> (Left  ())
                       | Right v15 -> (Right ())))
    | Right v10 ->
      (match (v0 (Right ())) with
       | Left  v11 -> (match (v0 (Left  ())) with
                       | Left  v13 -> (Left  ())
                       | Right v13 -> (Right ()))
       | Right v11 -> (match (v0 (Right ())) with
                       | Left  v12 -> (Left  ())
                       | Right v12 -> (Right ()))))
 | Right v2 ->
   (match (v0 (Right ())) with
    | Left  v3 ->
      (match (v0 (Left  ())) with
       | Left  v7 -> (match (v0 (Left  ())) with
                      | Left  v9 -> (Left  ())
                      | Right v9 -> (Right ()))
       | Right v7 -> (match (v0 (Right ())) with
                      | Left  v8 -> (Left  ())
```

$$
\begin{aligned}
\downarrow^{\theta} V &= V \\
\downarrow^{1} V &= \underline{\langle\rangle} \\
\downarrow^{\sigma\to\tau} V &= \text{let } \underline{x} \text{ be a fresh variable and } p \text{ a new prompt in } \underline{\lambda}\underline{x}.\,\text{set } p \text{ in} \downarrow^{\tau} (V \,@\, \uparrow^{\sigma} \underline{x}) \\
\downarrow^{\tau_1\times\tau_2} V &= \underline{\text{pair}}\langle \downarrow^{\tau_1} (\pi_1(V)), \downarrow^{\tau_2} (\pi_2(V))\rangle \\
\downarrow^{\tau_1+\tau_2} V &= \delta\big(V, x_1.\,\underline{\iota_1}(\downarrow^{\tau_1} x_1), x_2.\,\underline{\iota_2}(\downarrow^{\tau_2} x_2)\big) \\
\uparrow^{\theta} M &= M \\
\uparrow^{1} M &= \langle\rangle \\
\uparrow^{\tau\to\sigma} M &= \lambda x.\uparrow^{\sigma} (M \,\underline{@} \downarrow^{\tau} x) \\
\uparrow^{\sigma_1\times\sigma_2} M &= \text{pair}\langle \uparrow^{\sigma_1} (\underline{\pi_1}(M)), \uparrow^{\sigma_2} (\underline{\pi_2}(M))\rangle
\end{aligned}
$$

```
↑^{σ1+σ2} M  =  if $M$ is globally associated to $(L, \underline{z})$ modulo $\approx$
                then  $\iota_1(\uparrow^{\sigma_1} \underline{z})$
                else  if $M$ is globally associated to $(R, \underline{z})$ modulo $\approx$
                      then  $\iota_2(\uparrow^{\sigma_2} \underline{z})$
                      else  let $m$ be the best prompt for $M$
                            in  cupto $m$ as $c$
                                in  let  $\underline{x_1}$ and $\underline{x_2}$ be fresh variables
                                         associate $M$ to $(L, \underline{x_1})$ while computing $n_1 =$ set $m$ in $(c \,@\, \iota_1(\uparrow^{\sigma_1} \underline{x_1}))$
                                         associate $M$ to $(R, \underline{x_2})$ while computing $n_2 =$ set $m$ in $(c \,@\, \iota_2(\uparrow^{\sigma_2} \underline{x_2}))$
                                    in  if  $x_1 \notin FV(n_1)$, $x_2 \notin FV(n_2)$, and $n_1 \approx n_2$
                                        then  $n_1$
                                        else  $\underline{\delta}(M, \underline{x_1}.\,n_1, \underline{x_2}.\,n_2)$
```

Figure 5. Optimised type-directed normalisation.

```
                    | Right v8 -> (Right ()))))
    | Right v3 ->
      (match (v0 (Right ())) with
       | Left  v4 -> (match (v0 (Left  ())) with
                    | Left  v6 -> (Left  ())
                    | Right v6 -> (Right ()))
       | Right v4 -> (match (v0 (Right ())) with
                    | Left  v5 -> (Left  ())
                    | Right v5 -> (Right ())))))))
```

The result of normalising `fff` with the partial evaluator based on cupto is the residualisation of the identity:

```
 # tdpecupto
     ((bool **-> bool) **-> (bool **-> bool)) fff;;
- : Normal.normal =
(fun v0 ->
 (match (v0 (Left  ())) with
  | Left  v4 ->
    (match (v0 (Right ())) with
     | Left  v6 -> (fun v1 -> (Left  ()))
     | Right v7 ->
          (fun v1 -> (match v1 with
                      | Left  v2 -> (Left  ())
                      | Right v3 -> (Right ()))))
   | Right v5 ->
     (match (v0 (Right ())) with
      | Left  v10 ->
          (fun v1 -> (match v1 with
                      | Left  v2 -> (Right ())
                      | Right v3 -> (Left  ())))
      | Right v11 -> (fun v1 -> (Right ()))))))
```

7 Concluding remarks

We have presented a notion of normal term for the typed lambda calculus with sums and proved that every term of the calculus is equivalent to one in normal form. Further, we have used this theoretical development as the basis to implement a partial evaluator that provides a reductionless normalisation procedure for the typed lambda calculus with binary sums.

Our partial evaluator is in the style of TDPE. Thus, it can be grafted on any suitable interpreter, and does not need to examine the structure of the compiled code during normalisation. Its main originality is the use of the control operators set/cupto to fix the relative position of abstractions and discriminators. This is the first non-trivial exploitation of the extra expressive power of set/cupto over shift/reset. The effectiveness of the partial evaluator has been tested on the very sophisticated terms that come from the study of isomorphisms in the typed lambda calculus with sums [15], that make previously existing partial evaluators explode.

The new algorithm does not use all the power of the operators set/cupto. In particular we do not use their ability to code exceptions. One could thus use only a restricted version of these operators. There is, for example, a hierarchical version of shift/reset [8], that allows several, but fixed, levels of control. An implementation with shift/reset (hierarchical or not) is not obvious.

Acknowledgements. Thanks are due to Xavier Leroy for the `call/cc` for *Objective Caml*, and to Olivier Danvy, Andrzej Filinski, and Didier Rémy for interesting discussions about control operators.

8 References

[1] T. Altenkirch, P. Dybjer, M. Hofmann, and P. Scott. Normalization by evaluation for typed lambda calculus with coproducts. In *Sixteenth Annual IEEE Symposium on Logic in Computer Science*, pages 203–210. IEEE Computer Society Press, 2001.

[2] V. Balat and O. Danvy. Strong normalization by type-directed partial evaluation and run-time code generation. In *Second International Workshop on Types in Compilation*, number 1473 in Lecture Notes in Computer Science, pages 240–252. Springer-Verlag, 1998.

[3] V. Balat and O. Danvy. Memoization in type-directed partial evaluation. In *ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering (GCSE/SAIG)*, number 2487 in Lecture Notes in Computer Science, 2002.

[4] U. Berger and H. Schwichtenberg. An inverse of the evaluation functional for typed λ-calculus. In *Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 203–211. IEEE Computer Society Press, 1991.

[5] O. Danvy. Type-directed partial evaluation. In *Twenty-Third Annual ACM Symposium on Principles of Programming Languages*, pages 242–257. ACM Press, 1996.

[6] O. Danvy. Type-directed partial evaluation. In *Partial Evaluation – Practice and Theory; Proceedings of the 1998 DIKU Summer School*, number 1706 in Lecture Notes in Computer Science, pages 367–411. Springer-Verlag, 1998.

[7] O. Danvy and P. Dybjer, editors. *Preliminary Proceedings of the APPSEM Workshop on Normalization by Evaluation*, BRICS Note NS-98-1. Department of Computer Science, University of Aarhus, 1998.

[8] O. Danvy and A. Filinski. Abstracting control. In *ACM Conference on Lisp and Functional Programming*, pages 151–160. ACM Press, 1990.

[9] O. Danvy and A. Filinski. Representing control, a study of the CPS transformation. *MSCS*, 4(2):361–191, December 1992.

[10] R. Di Cosmo and D. Kesner. Simulating expansions without expansions. *Mathematical Structures in Computer Science*, 4:1–48, 1994.

[11] D. Dougherty. Some lambda calculi with categorical sums and products. In 5^{th} *International Conference on Rewriting Techniques and Applications (RTA-93)*, volume 690 of *Lecture Notes in Computer Science*, pages 137–151. Springer-Verlag, 1993.

[12] D. Dougherty and R. Subrahmanyam. Equality between functionals in the presence of coproducts. In *Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 282–291. IEEE Computer Society Press, 1995.

[13] D. Dougherty and R. Subrahmanyam. Equality between functionals in the presence of coproducts. *Information and Computation*, 157:52–83, 2000. (An earlier version appeared as [12]).

[14] M. Fiore. Semantic analysis of normalisation by evaluation for typed lambda calculus. In 4^{th} *International Conference on Principles and Practice of Declarative Programming (PPDP 2002)*. ACM Press, 2002.

[15] M. Fiore, R. Di Cosmo, and V. Balat. Remarks on isomorphisms in typed lambda calculi with empty and sum types. In *Seventeenth Annual IEEE Symposium on Logic in Computer Science*, pages 147–156. IEEE Computer Society Press, 2002.

[16] M. Fiore and A. Simpson. Lambda-definability with sums via Grothendieck logical relations. In *Typed Lambda Calculus and Applications*, number 1581 in Lecture Notes in Computer Science, pages 147–161. Springer-Verlag, 1999.

[17] N. Ghani. $\beta\eta$-equality for coproducts. In *Typed Lambda Calculus and Applications*, number 902 in Lecture Notes in Computer Science, pages 171–185. Springer-Verlag, 1995.

[18] C. A. Gunter, D. Rémy, and J. G. Riecke. Return types for functional continuations. 1998. (An earlier version appeared as [19]).

[19] C. A. Gunter, D. Rémy, and J. G. Riecke. A generalization of exceptions and control in ML. In *ACM Conference on Functional Programming and Computer Architecture*, 1995.

[20] J. Lambek and P. Scott. *Introduction to higher order categorical logic*, volume 7 of *Cambridge studies in advanced mathematics*. Cambridge University Press, 1986.