LFCS

Laboratory for Foundations of Computer Science
Department of Computer Science - The University of Edinburgh

Modelling Reduction in Confluent Categories

by

C. Barry Jay

LFCS Report Series
ECS-LFCS-91-187

LFCS
December 1991

Department of Computer Science
University of Edinburgh
The King's Buildings
Edinburgh EH9 3JZ

Copyright © 1991, LFCS

Copyright © 1991, Laboratory for Foundations of Computer Science,
University of Edinburgh. All rights reserved.

Reproduction of all or part of this work
is permitted for educational or research use
on condition that this copyright notice is
included in any copy.

Modelling Reduction in Confluent Categories *

C. Barry Jay †

Abstract

Rewriting systems can be modelled categorically by representing terms as morphisms and reduction by an order on the homsets. Confluence of the system, and hence of the homset orders, yields a confluent category. Collapsing these hom-orders, so that a term is identified with their reducts, and in particular any normal form it might have, then yields a denotational semantics. Confluent functors, natural transformations and adjunctions can also be defined, so that on passing to the denotations they yield the usual notions.

Two general ways of constructing such models are given. In the first objects are types and morphisms are terms, with substitution represented by equality. The second has contexts for objects and declarations for morphisms, with the call mechanism represented by a reduction. When applied to the simply-typed λ-calculus these constructions yield confluently cartesian closed categories. A third such model, in which the call rule is used to interpret β-reduction, is also given.

1. Introduction

One of the cornerstones of programming language semantics is the correspondence between simply-typed λ-calculi and cartesian closed categories established by Lambek and Scott [LS86]. It provides a clean algebraic semantics for λ-terms, which helps us to reason effectively about the values of programs. Of equal interest to the implementer, however, is the reduction process which computes these values, i.e. the operational semantics. One of the most satisfying approaches to date is Plotkin's structural operational semantics [Plo75, Plo81] which uses a sequent calculus to describe the reduction process and provide a rigorous specification for computation.

*to appear, without appendix, in *Proceedings of the Durham Symposium on Applications of Categories in Computer Science.*

†Laboratory for Foundations of Computer Science, University of Edinburgh, The King's Buildings, Mayfield Road, Edinburgh, U.K. EH9-3JZ. Research supported by BP/Royal Society of Edinburgh Research Fellowship.

In spite of their separate virtues, it is difficult to relate the logic (of operational semantics) to the categorical algebra (of denotational semantics); the usual approach, using computational adequacy, is rather unwieldy.

The proposal here is to give a *categorical* semantics of reduction with a simple functorial relationship between the operational and denotational semantics. This is achieved by extending the usual mapping of types to objects and terms to morphisms, so that reductions correspond to an order relation on the morphisms. We will impose two conditions on this relation. First, it must be a rewriting system, so that reduction is modular, and the composition of the category preserves the order. Second, the reduction order must be confluent, so that any normal form that exists is unique. This leads to *confluent categories* and the corresponding functors, natural transformations and adjoints.

Confluent functors map objects to objects, and morphisms to morphisms, as usual, and preserve the order on the homsets. Composition and identities, however, are not preserved exactly but only up to compatibility, i.e. having a common reduct. For example, if $g \circ f$ is a composite morphism in $\mathcal{C}$ and $F : \mathcal{C} \to \mathcal{D}$ is a confluent functor then $F(g \circ f)$ and $Fg \circ Ff$ are compatible. Similarly, a *confluent natural transformation* $\alpha : F \Rightarrow G : \mathcal{C} \to \mathcal{D}$ between confluent functors is given by a family of morphisms $\alpha_A : FA \to GA$ such that, for each $f : A \to B$ in $\mathcal{C}$ there is some $g : FA \to GB$ such that

$$\begin{array}{ccc} FA & \xrightarrow{\alpha_A} & GA \\ {\scriptstyle Ff}\downarrow & \overset{g}{\swarrow} & \downarrow{\scriptstyle Gf} \\ FB & \xrightarrow[\alpha_B]{} & GB \end{array} \qquad (1)$$

The intuition is that since both sides of the square reduce to a common form they have the same denotational semantics. Technically, confluence of the order is necessary to ensure that the horizontal composition of natural transformations and the application of functors to natural transformations are defined. It follows that the usual Godemont rules for functors and natural transformations hold, provided that some of the equalities are relaxed to compatibilities. Confluent adjunctions are defined similarly, and determine the *confluently cartesian closed categories.*

The ease with which the usual calculus of functors and natural transformations is generalised to confluent categories is in stark contrast to the situation

for general ordered categories. Without confluence, one is forced to make composition and identities preserved up to a reduction, which may be in one of two possible directions, yielding *lax* and *colax functors* (unlike the symmetric concept of compatibility). Both types of functor arise in practice, but they do not compose. There are similar incompatible choices for natural transformations (namely, the *lax* and *oplax*). And finally, applying lax functors to lax natural transformations does not always yield a lax (or oplax) natural transformation. Since there doesn't appear to be a general framework, one is forced to seek out additional constraints that are satisfied in interesting classes of examples. For example, one may characterise categories of relations in terms of their sub-category of *maps* (internal left adjoints) [CKS84, CKW89] or categories of partial morphisms in terms of their *total morphisms* [Jay90a, Jay90e]. Here confluence of the orders will be used.

Once a reduction system has been modelled in a confluent category $\mathcal{C}$ then the corresponding denotational semantics arises by collapsing the order on the homsets, to obtain a set-based category whose homsets are equivalence classes of morphisms of $\mathcal{C}$. Confluent functors, natural transformations and adjunctions are mapped to their more usual counterparts so that, in particular, confluently cartesian closed categories are mapped to their ordinary counterparts.

Some authors [See87, RS87, HR91] seeking categorical models of reduction have chosen to use 2-categories or bicategories, that is, to label their reductions and establish equations and inequations between them. However, without prejudging future developments, there are many interesting phenomena which are obscured by the large conceptual overhead arising from the need to choose, state and verify many equations on 2-cells, and establish the desired coherence theorems. Similar choices are available in modelling refinement, where ordered categories have been used [HHM89].

The construction of a confluently cartesian closed category from a confluent rewriting system can be done in many ways, depending on which features are to be emphasised. Two methods are demonstrated here. The first is chosen for maximum simplicity (objects are types, morphisms are terms, and substitution is an equality), so that the basic ideas can be understood. The second takes contexts as objects and declarations as morphisms, with the call rule (substitution) modelled by a reduction.

Finally, these techniques are used to model the simply-typed λ-calculus in confluently cartesian closed categories. As well as the two canonical models derived from the previous section, there is a third, in which the usual β-reduction is broken into two steps, the creation of a new declaration (corresponding to its interpretation by a `let`), and its call to the body of the

λ-term.

The paper is divided into the following sections: Introduction; Confluent Categories; Confluent Adjunctions; Modelling Reduction; The λ-Calculus; Further Work.

2. Confluent Categories

Let $(P, \Rightarrow)$ be a pre-ordered set in which the pre-order is called *reduction*. Elements x, y of P are *equivalent*, denoted $x \Leftrightarrow y$ if each reduces to the other, in which case the reductions $x \Rightarrow y$ and $y \Rightarrow x$ are *reversible*. They are *compatible*, denoted $x \sim y$, if they have a common reduct, i.e. there is a z such that $x \Rightarrow z$ and $y \Rightarrow z$. Define P to be *confluently ordered* if the set of reducts of any given element are all pairwise compatible (that is, $\sim$ is an equivalence relation on P).

The ordered category **Conf** is defined to have confluent orders as objects and order-preserving functions as morphisms. The order on the morphisms $f, g : P \to Q$ between confluent orders is the equivalence relation given by pointwise compatibility, i.e. $f \sim g$ iff for all $x \in P$ we have $f(x) \sim g(x)$. Now P and Q are *equivalent* if there are confluent functions $f : P \to Q$ and $g : Q \to P$ such that $g \circ f \sim \mathrm{id}_P$ and $f \circ g \sim \mathrm{id}_Q$.

The interplay between preserving reduction and preserving compatibility is quite delicate. Morphisms in **Conf** will be used to combine reduction systems as modules of a larger system, and so should preserve the actual computation steps, and not merely their denotations, which is all that compatibility ensures. However, in comparing different ways of combining modules, it suffices to compare their denotations, so that compatibility suffices. Note that if the morphisms of **Conf** were functions that merely preserve compatibility then they could not distinguish between $\Rightarrow$ and $\sim$ and the category would be equivalent to its subcategory of equivalence relations.

Recall [Jay90e] that an *ordered category* is a category $\mathcal{C}$ whose homsets have a given pre-order such that composition is order-preserving, i.e. $\mathcal{C}$ is pre-order enriched.

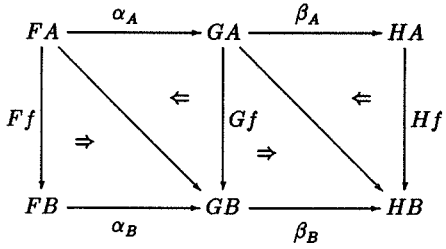
Definition 2.1 *A* confluently ordered category *is an ordered category whose homsets are confluently ordered. A* confluent functor $F : \mathcal{C} \to \mathcal{D}$ *between confluent categories $\mathcal{C}$ and $\mathcal{D}$ is given by a function $A \mapsto FA$ from the objects of $\mathcal{C}$ to those of $\mathcal{D}$ and, for each pair of objects A, B of $\mathcal{C}$ an order-preserving function $F_{A,B} : \mathcal{C}(A,B) \to \mathcal{D}(FA, FB)$ which together preserve composition and identities, up to compatibility. That is, if $g \circ f : A \to C$ is a composite*

morphism in $\mathcal{C}$ then

$$\begin{aligned} F(g \circ f) &\sim Fg \circ Ff \\ F(\mathrm{id}_A) &\sim \mathrm{id}_{FA} \end{aligned}$$

A confluent (natural) transformation $\alpha : F \Rightarrow G : \mathcal{C} \to \mathcal{D}$ *between confluent functors is given by a family of components* $\alpha_A : FA \to GA$ *such that for all* $f : A \to B$ *in* $\mathcal{C}$ *we have* $Gf \circ \alpha_A \sim \alpha_B \circ Ff$ *(see Diagram 1 above).*

Vertical composition of confluent natural transformations is defined in the usual way: if $\alpha : F \to G : \mathcal{C} \to \mathcal{D}$ and $\beta : G \to H$ then $\beta \circ \alpha : F \to H$ has components $\beta_A \circ \alpha_A$



Confluence ensures that the two different reductions of $\beta_B \circ Gf \circ \alpha_A$ have a common reduct, which makes $\beta \circ \alpha$ a confluent natural transformation.

Define confluent natural transformations to be *compatible* if their components at each object are. Also an *equivalence* of the confluent functors $F, G; \mathcal{C} \to \mathcal{D}$ is given by a pair of confluent natural transformations $\alpha : F \to G$ and $\beta : G \to F$ such that $\beta \circ \alpha$ and $\alpha \circ \beta$ are both equivalent to identity natural transformations.

A diagram in a confluent category is *compatible* if any pair of paths through it are so. Then confluence of the order ensures that a diagram is compatible if all of its cells are. In future, a diagram cell with compatible sides will be labelled with the compatibility symbol $\sim$.

Further, confluent functors preserve compatible diagrams, from which follows several facts. First, composites of confluent functors are confluent. Second, if α is as above and $\alpha' : F' \to G' : \mathcal{D} \to \mathcal{E}$ is another confluent natural transformation then we can define confluent natural transformations $F'\alpha : F' \circ F \to F' \circ G$ and $\alpha' F : F' \circ F \to G' \circ F$ with components at A given by $F'\alpha_A : F'FA \to F'GA$ and α'_{FA} respectively. Thus we can define the horizontal composition of α and $\alpha' : F' \to G' : \mathcal{D} \to \mathcal{E}$ to be

$$(\alpha' * \alpha) = \alpha' G \circ F'\alpha$$

A distinct, but compatible, alternative would have the horizontal composite be $G'\alpha \circ \alpha' F$.

Now the usual Godemont rules for functors and natural transformations hold, provided that equality is replaced by compatibility. For example, the interchange law for α, α', β as above and $\beta' : G' \to H'$ asserts

$$(\beta' \circ \beta) * (\alpha' \circ \alpha) \sim (\beta' * \alpha') \circ (\beta * \alpha)$$

which follows from the confluence of G'' and of β' with respect to components of α.

Consequently, key notions such as adjoints, which are defined up to isomorphism in a 2-category, are here defined only up to compatibility. If one must have a 2-category then this can be obtained in at least two different ways, which will not be pursued here. First, one could replace the confluent natural transformations by their equivalence classes under compatibility; this would damage the correspondence with the syntax in the examples. Second, and perhaps more natural, would be to require that the components of confluent natural transformations be normal forms. Of course this will only make sense if composites of normal forms are normalisable.

In the absence of a 2-category, and without wishing to enter ever-higher dimensions, let us consider **ConfCat** to be the 2-graph of confluent categories, functors and natural transformations.

Proposition 2.2 *There is a 2-graph homomorphism* $(-)_r : \mathbf{ConfCat} \to \mathbf{Cat}$ *which preserves all horizontal and vertical composites, and adjunctions, and identifies compatible natural transformations.*

Proof Define the *residual category* $\mathcal{C}_r$ of a confluent category $\mathcal{C}$ to be the quotient of $\mathcal{C}$ by the congruence given by compatibility. The rest is straightforward. □

Thus by interpreting reduction in a confluent category and then taking the residual category one obtains a denotational semantics of the reduction system. Of course, the problem of interpreting the equivalence classes remains, but the conceptual difficulties of linking the operational and denotational semantics have been removed.

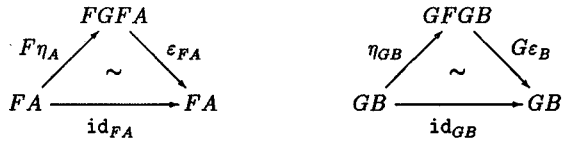
If $\mathcal{C}$ is a confluent category then $\mathcal{C}^{op}$ is also confluent. The opposite of confluent functors and natural transformations are also defined in the usual way. However, the other dual $\mathcal{C}^{co}$ in which only the order on the morphisms is reversed, is not generally confluent. The product of confluent categories is also confluent with the pointwise order.

Conf is a confluent category since the order on its morphisms is actually an equivalence relation. Further, each object A in a confluent category $\mathcal{C}$ yields confluent *hom-functors* $\mathcal{C}(A,-) : \mathcal{C}\to\mathbf{Conf}$ and $\mathcal{C}(-,A) : \mathcal{C}^{op}\to\mathbf{Conf}$, and each morphism $A\to B$ yields a corresponding pair of confluent natural transformations. Define a confluent functor $\mathcal{C}\to\mathbf{Conf}$ (respectively, $\mathcal{C}^{op}\to\mathbf{Conf}$) to be *representable* if it is equivalent to a representable functor.

3. Confluent Adjunctions

With the weaker theory that we have chosen it is inevitable, if not desirable, that compatibilities will replace equalities in the definition of adjoints. That said, all the usual ways of representing adjunctions, as natural isomorphisms (or equivalences), via representablility, universal maps or triangle laws are valid.

Definition 3.1 *A* confluent adjunction *is given by a pair of confluent functors $F : \mathcal{C}\to\mathcal{D}$ and $G : \mathcal{D}\to\mathcal{C}$ and a pair of confluent natural transformations, namely the* unit η : $\mathtt{id}\to GF$ *and the* counit ε : $FG\to\mathtt{id}$ *which satisfy the following triangle laws*



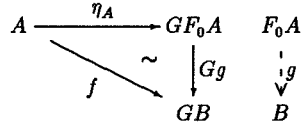
We may write $F \dashv G : \mathcal{D}\to\mathcal{C}$ when η and ε are understood.

Of course if $F \dashv G$ then $G^{op} \dashv F^{op}$ and the following results can be dualised.

Lemma 3.2 *Two confluent left adjoints to a confluent functor are equivalent.*

Proof Let $(F, G, \eta, \varepsilon) : \mathcal{D}\to\mathcal{C}$ and $(F', G, \eta', \varepsilon') : \mathcal{D}\to\mathcal{C}$ yield two left adjoints for G. Then the equivalence is given by $\varepsilon F' \circ F\eta' : F\to F'$ in the usual way. □

Let $G : \mathcal{D}\to\mathcal{C}$ be a confluent functor and let A be an object of $\mathcal{C}$. A *universal arrow* from A to G is given by an object F_0A in $\mathcal{D}$ and an arrow $\eta_A : A\to GF_0A$ such that for each morphism $f : A\to GB$ there is a chosen arrow $g : F_0A\to B$ such that (i) the diagram



is compatible; (ii) if $h : F_0A \to B$ is such that $Gh \circ \eta_A \sim f$ then $h \sim g$, and; (iii) if $f \Rightarrow f'$ then $g \Rightarrow g'$ where $g' : F_0A \to B$ is the morphism determined by f'.

Theorem 3.3 *The following statements about a confluent functor $G : \mathcal{D} \to \mathcal{C}$ are equivalent:*

(i) G has a left adjoint.

*(ii) There is a confluent functor $F : \mathcal{C} \to \mathcal{D}$ such that the confluent functors $\mathcal{D}(F-,?)$ and $\mathcal{C}(-,G?) : \mathcal{C}^{op} \times \mathcal{D} \to$ **Conf** are equivalent, i.e. there are equivalences*

$$\mathcal{D}(FA,B) \underset{\psi_{A,B}}{\overset{\phi_{A,B}}{\rightleftarrows}} \mathcal{C}(A,GB)$$

which are confluently natural in A and B.

*(iii) The functors $\mathcal{C}(A,G-) : \mathcal{C} \to$ **Conf** are all representable.*

(iv) For each object A in $\mathcal{C}$ there is a universal arrow from A to G.

Proof (i)$\Rightarrow$ (ii): define $\phi_{A,B}(g) = Gg \circ \eta_A$ and $\psi_{A,B}(f) = \varepsilon_B \circ Ff$. The compatibility proofs are routine. (ii) $\Rightarrow$ (iii): immediate.

(iii) $\Rightarrow$ (iv): Let F_0A be the representing object for $\mathcal{C}(A,G-)$ and let $\alpha_B : \mathcal{D}(F_0A,B) \to \mathcal{C}(A,GB)$ and $\beta_B : \mathcal{C}(A,GB) \to \mathcal{D}(F_0A,B)$ be the natural equivalences. Define

$$\eta_A = \alpha_{F_0A}(\mathrm{id}) : A \to GF_0A$$

Now if there is a g making the triangle above compatible, then

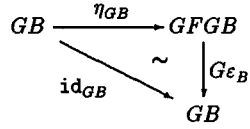
$$\beta_B(f) \sim \beta_B(Gg \circ \eta_A) \sim g \circ \beta_{F_0A}(\eta_A) = g \circ \beta_{F_0A}(\alpha_{F_0A}(\mathrm{id})) \sim g$$

by the naturality of β with respect to B. This establishes the uniqueness of g up to compatibility. The naturality of α with respect to B can be similarly used to prove that $\beta_B(f)$ does indeed make the triangle compatible, and the construction is order-preserving since β is.

(iv) $\Rightarrow$ (i): F_0 can be made the object part of a functor $\mathcal{C} \to \mathcal{D}$ by defining F on arrows as follows. If $f : A \to A'$ in $\mathcal{C}$ then Ff is the chosen map which makes the following square compatible

$$\begin{array}{ccc} A & \xrightarrow{\eta_A} & GFA \\ {\scriptstyle f}\downarrow & \sim & \downarrow{\scriptstyle GFf} \\ A' & \xrightarrow[\eta_{A'}]{} & GFA' \end{array}$$

The universal property shows that F preserves the order on morphisms, and preserves composition and identities up to compatibility. Thus it is a confluent functor. Also, η is a confluent natural transformation. Define $\varepsilon_B : FGB \to B$ to be the chosen morphism making the following compatible



ε_B is natural in B since given $g : B \to B'$ in $\mathcal{D}$ then $\varepsilon_{B'} \circ FGg$ and $g \circ \varepsilon_B : FGB \to B'$ are both equivalent to the induced morphism corresponding to $Gg : GB \to GB'$ and so are equivalent.

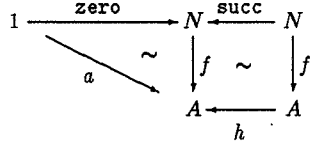
One of the triangle laws defines ε. The other follows upon verifying that $\varepsilon_{FA} \circ F\eta_A$ and $\mathtt{id}_{FA}$ are both equivalent to the morphism induced by η_A. □

The definitions of product and coproduct functors, and of cartesian closure flow directly from that of adjunction. They will satisfy the usual universal properties except that equalities are replaced by compatibility.

At this point we should also define a *confluent natural numbers object*. Without striving for maximum generality [Rom89, PR89, Coc90, Jay91a], we may define it to be an object N equipped with morphisms $\mathtt{zero} : 1 \to N$ and $\mathtt{succ} : N \to N$ such that for each diagram

$$1 \xrightarrow{a} A \xleftarrow{h} A$$

there is a chosen morphism $f : N \to A$ called the *iterator* of a and h such that (i) the diagram



is compatible; (ii) if f can be replaced by $g : N \to A$ in the diagram while preserving compatibility then $f \sim g$ and; (iii) if $a \Rightarrow a'$ and $h \Rightarrow h'$ then $f \Rightarrow f'$ where f' is the iterator of a' and h'. If the iterator is exists but without satisfying (ii) above then $(N, \mathtt{zero}, \mathtt{succ})$ is a confluent *weak* natural numbers object.

The temptation to replace the compatibilities in this definition with reductions must be resisted, since these changes would neither be in the spirit of the universal properties above nor be satisfied by the examples in Section 5.3.

4. Modelling Reduction

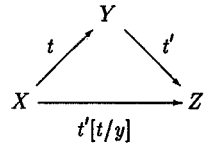
A rewriting system is a reduction system R in which arbitrary sub-terms may be reduced. That is, if $t_1 \Rightarrow t_2$ then $t[t_1/x] \Rightarrow t[t_2/x]$ (where $t[t_1/x]$ denotes the expression obtained by replacing each free occurrence of x in t by t_1). We will consider two constructions of ordered categories from R, which will be fixed for the rest of the section. The first will quickly establish the general approach, while the second will pay closer attention to contexts, declarations and substitution.

4.1. Substitution as Composition

Define an ordered category $\mathcal{C}(R)$ whose objects are the types and whose morphisms

$$X \xrightarrow{t} Y$$

are terms $t : Y$ which have at most one free variable, say x, which is of type X (as in [LS86]), that is $FV(t) \subseteq \{x\}$. The order on morphisms is generated by term reduction in R. Composition is given by substitution: if $FV(t') \subseteq \{y\}$ then



commutes.

Composition preserves the order since R is a rewriting system. Thus $\mathcal{C}(R)$ is an ordered category, which is confluent iff R is a confluent rewriting system.

$\mathcal{C}(R)$ suffers from two main operational defects. First, its restriction to terms with at most one free variable only makes sense in the presence of product types, which is perhaps too great a burden for them. Second, treating substitution as an equality ignores its fundamental importance, and, in particular, its irreversibility as a computation step.

4.2. Contexts and Declarations

The next model will have *contexts* as its objects and *declarations* as its morphisms. The presence of many variables in the context removes the first objection. For the second, composition of declarations will be given freely,

with the call mechanism (replacing substitution) presented as a form of reduction, or computation step. In this way, substitution or calling becomes more explicit, and thus can be controlled.

A *context* is a finite collection of typed variables of R, i.e. a set of variables equipped with type assignments. *Declarations* are lists of *atomic declarations* which in turn have the form $x = t$ where t is an term of the reduction system. Declarations should be thought of as updating a context or environment, or as a way of interpreting a `let` expression, i.e. interpret $y = \texttt{let}\ x = t_1\ \texttt{in}\ t_2$ by $x = t_1; y = t_2$. Thus atomic declarations bind everything to their right, for example $x = e$ binds d in $x = e; d$. Any declaration d and context Δ combine to form a *typed declaration* $d : \Delta$.

The ordered category $\mathcal{D}(R)$ has contexts for objects. Informally, a morphism $\Gamma \rightarrow \Delta$ between contexts is given by a declaration d such that Γ contains the variables required to produce Δ from d. More precisely, define the *minimum domain* $\mathrm{dom}(d : \Delta)$ of a *typed declaration* $d : \Delta$ by

$$\begin{aligned} \mathrm{dom}(nil : \Delta) &= \Delta \\ \mathrm{dom}(x = t; d : \Delta) &= FV(t) \cup (\mathrm{dom}(d, \Delta) - \{x\}) \end{aligned}$$

Now $d : \Gamma \rightarrow \Delta$ if $\Gamma \supseteq \mathrm{dom}(d : \Delta)$. Note that a declaration d will be a morphism between many different pairs of contexts, e.g. $\texttt{nil} : \Gamma \rightarrow \Delta$ iff $\Gamma \supseteq \Delta$. Composition of declarations is given by appending of lists and identities are given by `nil`.

Reduction of morphisms $d : \Gamma \rightarrow \Delta$ are specified by reductions of the typed declaration $d : \Delta$ (which will always preserve the type). Its *basic reductions* are given by the following list:

(E)	$x = t$	$>$	$x = t'$	if $t \Rightarrow t'$ in R.
(C)	$x = t_1; y = t_2$	$>$	$y = t_2[t_1/x]; x = t_1$	if $x \neq y$ and $y \notin FV(t_1)$.
(T)	$x = t_1; x = t_2$	$>$	$x = t_2[t_1/x]$	
(R)	$x = t$	$>$	$y = t; x = y$	if $y \notin \Delta$.
(W)	$x = t$	$>$	nil	if $x \notin \Delta$.
(N)	nil	$>$	$x = x$	if $x \in \Delta$.

The first of these is *expression reduction* and is the only rule which refers to the reduction system R. The rest are *declaration reductions* which are named, respectively, *call, thinning, renaming of identifiers, weakening and nil reduction.* In particular, (C) above is a *call* of $x = t_1$ to $y = t_2$.

The *1-step reductions* are given by a decomposition of d as $d = d_1; d_2; d_3$ and a basic reduction $d_2 > d_4 : \mathrm{dom}(d_3, \Delta)$ whence $d \Rightarrow d_1; d_4; d_3 : \Delta$. They inherit the titles of expression reductions, calls etc. according to the basic

reduction employed. General reduction is the reflexive, transitive closure of 1-step reduction.

Here is an example of reduction of a declaration of type $\{y : N, z : N\}$ where R is any suitable reduction system for the natural numbers N and addition $(+)$.

$$\begin{array}{lll} x = 1; y = x + 2; z = x + 3 & \Rightarrow & y = 3; x = 1; z = x + 3 \quad \text{(C,E)} \\ & \Rightarrow & y = 3; z = 4; x = 1 \quad \text{(C,E)} \\ & \Rightarrow & y = 3; z = 4 \quad \text{(W)} \end{array}$$

The fundamental declaration reduction is call. Note that the declaration $x = t_1$ in its definition is not lost, but placed so that it still binds declarations further to the right. When it no longer binds any expressions then it may be weakened out. The two restrictions on call prevent any identifiers from increasing their scope. When $x = y$ then thinning can be applied; the declaration $x = t_1$ disappears since any later occurrences of x are bound by t_2. When y is free in t_1 then renaming, which plays a role comparable to α-conversion, can be used to permit the call to procede. This is illustrated by reduction of $x = y + 1; y = x + 2 : \{y : N\}$. Call cannot be applied, and so without renaming this declaration would be essentially normal, despite the possibility of substituting for x. The difficulty arises since y appears in the minimum domain but is later rebound as an identifier of the declaration. The solution is to separate the two uses of y by introducing another identifier u, as follows:

$$\begin{array}{lll} x = y + 1; y = x + 2 & \Rightarrow & x = y + 1; u = x + 2; y = u \quad \text{(R)} \\ & \Rightarrow & u = y + 3; y = u \quad \text{(C,E,W)} \\ & \Rightarrow & y = y + 3 \quad \text{(C,W)} \end{array}$$

In this case the identifier u that was introduced could later be eliminated. However, a second variable in the codomain may prevent this as is shown by

$$\begin{array}{ll} & x = y + 1; y = x + z; z = x \\ \Rightarrow & u = y + 1 + z; z = y + 1; y = u : \{y : N, z : N\} \end{array} \qquad (2)$$

whose right-hand side is essentially normal. Here the rebinding $y = u$ must be performed after all uses of the original y are complete. There is, however, a convenient shorthand for this rebinding process.

A *product* of contexts $\Gamma = \{x_i : X_i \mid 1 \leq i \leq m\}$ and $\Delta = \{y_j : Y_j \mid 1 \leq j \leq n\}$ is given by a choice of disjoint union:

$$\Gamma \times \Delta = \{x_{i1} : X_i \mid 1 \leq i \leq m\} \cup \{y_{j2} : Y_j \mid 1 \leq j \leq n\}$$

The first projection $\pi_{\Gamma,\Delta}$ is then the translation

$$x_1 = x_{1,1}; x_2 = x_{2,1}; \ldots; x_m = x_{m,1} : \Gamma \times \Delta \rightarrow \Gamma$$

with the second projection $\pi'_{\Gamma,\Delta}$ defined similarly. The pairing of declarations $d : \Theta \rightarrow \Gamma$ and $d' : \Theta \rightarrow \Delta$ is given by $d; \theta_{\Gamma,\Delta}; d'; \theta'_{\Gamma,\Delta} : \Theta \rightarrow \Gamma \times \Delta$ where $\theta_{\Gamma,\Delta}$ is defined by $x_{1,1} = x_1; x_{2,1} = x_2; \ldots; x_{m,1} = x_m$ with $\theta'_{\Gamma,\Delta}$ defined similarly. Note that θ is not a morphism $\Gamma \rightarrow \Gamma \times \Delta$.

Now if Γ and Δ have no variables in common then we can define the *conjunction* of d and d' by

$$\Theta \xrightarrow{d \text{ and } d'} \Gamma \cup \Delta$$

by d and $d' = \langle d, d' \rangle; \pi; \pi'$. Then the result of (2) is equivalent to

$$(y = y + 1 + z) \text{ and } (z = y + 1) : \{y : N, z : N\}$$

Aside from any reduction loops that might arise in R there are some new loops built from declaration reductions. For example, in the definition of call above if $y \notin FV(t_1)$ and $x \notin FV(t_2)$ then the call is a *1-step permutation* which can be reversed by another permutation

$$x = t_1; y = t_2 \Leftrightarrow y = t_2; x = t_1$$

Sequences of 1-step permutations are, of course, *permutations*. Also, renamings are reversible since

$$x = t_1 \Rightarrow u = t_1; x = u \Rightarrow x = t_1; u = t_1 \Rightarrow x = t_1$$

by using call and weakening.

These reversible reductions, such as permutations and renamings seem quite harmless (like α-conversion) but force us to modify our notion of normal form accordingly.

A term t is *essentially normal* if any reduction of it is reversible. It is *weakly essentially normalisable* if it reduces to an essentially normal term, and *(strongly) essentially normalisable* if each reduction sequence

$$t = t_0 \Rightarrow t_1 \Rightarrow t_2 \Rightarrow \ldots \Rightarrow t_n \Rightarrow \ldots$$

has a finite number of irreversible steps, called its *essential length*. R is *weakly (respectively, strongly) essentially normalising* if every expression is weakly (respectively, strongly) essentially normalisable.

Let R_0 be the term algebra of R, i.e. the sub-system containing all the terms, but with no non-trivial reduction.

Theorem 4.1 *The system of declarations and declaration reductions is confluent and strongly essentially normalising. A declaration* $d : \Gamma \rightarrow \Delta = \{y_j : Y_j\}$ *(for* $j = 1, \ldots n$*) is essentially normal iff it is equivalent to a conjunction*

$$(y_1 = t_1) \text{ and } (y_2 = t_2) \text{ and } \ldots \text{ and } (y_n = t_n)$$

The corresponding confluent category $\mathcal{D}(R_0)$ *has confluent finite products.*

Proof The full proof is in [Jay91c, Appendix]. It proceeds by progressively stripping out the obstructions to call, and hence the other reduction rules, and then proving that call is essentially normalising by a rank argument. Then confluence follows from a strengthening of weak confluence to account for reversible reductions. □

Theorem 4.2 *Let* R *be a typed reduction system. Then* $\mathcal{D}(R)$ *is a confluent category (respectively, weakly essentially normalising) iff* R *is a confluent rewriting system (respectively, weakly essentially normalising).*

Proof The forward direction is trivial since the call rule forces R to be a rewriting system. In the reverse direction, it suffices to prove the confluence of the coarser system whose 1-step reductions are sequences either of expression reductions or of declaration reductions. Now it is straight-forward to show that critical pairs of 1-step reductions can be completed in one step, which yields confluence. For weak essential normalisation, it suffices to reduce to the essential normal form in the sub-system of declaration reductions, and then reduce the expressions to essential normal forms. □

5. The λ-Calculus

The models of reduction above will now be instantiated for the simply-typed λ-calculus with pairing, unit or terminal type and iterator. After addressing some issues concerning confluence of expression reduction, the models of the previous section can be built directly and their confluent cartesian closure established. Following this, we will see how to tailor expression reduction to exploit the presence of declarations.

5.1. The Reduction System

Fix R to be the following variant of the λ-calculus, discussed in detail as the *expansionary reduction system* in [Jay91b]. Its *types* are generated by the type N of natural numbers and the unit type 1 and closed under products $X \times Y$ and arrows $X \rightarrow Y$.

Each type X comes equipped with a collection of variables $x : X$. Variables will be denoted by the letters x, y, z and their annotated versions. Other lower case letters will denote arbitrary terms. The *terms* are inductively generated by

(i) variables
(ii) $\lambda x.b : A \to B$ if $x : A$ and $b : B$
(iii) $fa : B$ if $f : A \to B$ and $a : A$
(iv) $\langle a, b \rangle : A \times B$ if $a : A$ and $b : B$
(v) $\pi c : A$ if $c : A \times B$
(vi) $\pi' c : B$ if $c : A \times B$
(vii) $0 : N$
(viii) $Sn : N$ if $n : N$
(ix) $\mathtt{It}(a, h, n) : A$ if $a : A$ and $h : A \to A$ and $n : N$
(x) $* : 1$

Terms equivalent up to change of bound variable will be identified.

The *basic reductions* are:

$$\begin{array}{lrcl}
(\beta) & (\lambda x.b)a & > & b[a/x] \\
(\eta') & t & > & \lambda x.tx \\
(\pi) & \pi\langle a, b\rangle & > & a \\
(\pi') & \pi'\langle a, b\rangle & > & b \\
(sp) & c & > & \langle \pi c, \pi' c\rangle \\
(*) & a & > & * \\
(It0) & \mathtt{It}(a, h, 0) & > & a \\
(ItS) & \mathtt{It}(a, h, Sn) & > & h\mathtt{It}(a, h, n)
\end{array}$$

where $t : A \to B$ and $c : A \times B$ and $a : 1$ is not $*$. The (β) rule involves implicit α-conversion whenever substitution threatens to capture free variables of a. The *1-step* reductions of a term are obtained by applying a basic reduction to one of its sub-terms. The transitive closure of the 1-step reductions is the reduction relation, denoted $\Rightarrow$. Reductions built solely from $(\eta'), (sp)$ and $(*)$ are called *expansions*. Conversely, reductions built without any use of expansions are *contractions*.

It is more usual to represent η-conversion and surjective pairing by contractions. The difficulty, observed by Obtulowicz (see [LS86]) is that the presence of the unit type prevents confluence. With expansions, however, we have the following result [Jay91b]

Theorem 5.1 *The expansionary reduction system is a confluent and weakly essentially normalising rewriting system with essential normal forms given by the long $\beta\eta$-normal forms of Huet [Hue76].* □

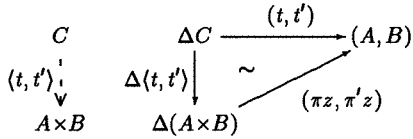
5.2. A Substitution Model

Theorem 5.2 *$\mathcal{C}(R)$ is a confluent cartesian closed category with confluent weak natural numbers object whose hom-orders are essentially normalising.*

Proof The product of types A and B in $\mathcal{C}(R)$ is simply $A{\times}B$. This yields a confluent right adjoint to the diagonal functor $\Delta : \mathcal{C}(R){\to}\mathcal{C}(R)^2$ whose universal arrow $\Delta(A{\times}B){\to}(A,B)$ in $\mathcal{C}(R)^2$ is given by

$$(A{\times}B, A{\times}B) \xrightarrow{(\pi z, \pi' z)} (A,B)$$

Now given $(t,t') : \Delta C{\to}(A,B)$ (and assuming that t and t' share any free variable they have) the induced morphism is $\langle t,t'\rangle : C{\to}A{\times}B$.



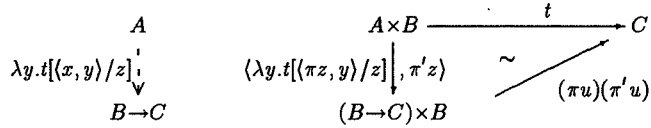
The compatibility arises because $\pi z \circ \langle t,t'\rangle = \pi\langle t,t'\rangle{\Rightarrow}t$ in $\mathcal{C}(R)$ and similarly for π'. The uniqueness of the induced morphism follows because if $f : C{\to}A{\times}B$ satisfies $\pi f \sim t$ and $\pi' f \sim t'$ then $f \sim \langle \pi f, \pi' f\rangle \sim \langle t,t'\rangle$. Finally if $t{\Rightarrow}s$ and $t'{\Rightarrow}s'$ then $\langle t,t'\rangle{\Rightarrow}\langle s,s'\rangle$.

The unit for the adjunction is thus $\langle x,x\rangle : A{\to}A{\times}A$ and the product of morphisms $t : A{\to}B$ and $t' : A'{\times}B'$ is given by

$$t{\times}t' = \langle t[\pi z/x], t'[\pi' z/x']\rangle$$

where x and x' are assumed to be the free variables of t and t' respectively.

The exponential C^B in $\mathcal{C}(R)$ is just the function type $B{\to}C$. The universal arrow is the evaluation $\mathrm{ev}_{B,C} = (\pi z)(\pi' z) : (B{\to}C){\times}B{\to}C$. Now given $t : A{\times}B{\to}C$ with free variable z the induced morphism is $\lambda y.t[\langle x,y\rangle/z] : A{\to}C^B$ with free variable $x : A$.



Compatibility follows since the lower composite is

$$\begin{array}{lcl}(\pi\langle\lambda y.t[\langle\pi z,y\rangle/z],\pi' z\rangle)(\pi'\langle\lambda y.t[\langle\pi z,y\rangle/z],\pi' z\rangle) & \Rightarrow & (\lambda y.t[\langle\pi z,y\rangle/z])(\pi' z)\\ & \Rightarrow & t[\langle\pi z,\pi' z\rangle/z]\\ & \Leftarrow & t\end{array}$$

Uniqueness of the induced morphism follows by direct computation.

The type N is a weak natural numbers object with $\mathtt{zero} = 0 : 1 \to N$ and $\mathtt{succ} = Sn : N \to N$ where $n : N$ is some variable. The iterator of morphisms $a : 1 \to A$ and $t : A \to A$ is given by $\mathtt{It}(a, \lambda x.t, n)$ where $x : A$ is the free variable of t. □

5.3. A Declaration Model

Theorem 5.3 *Let R' be the sub-system of R without product or unit types. Then $\mathcal{D}(R')$ and $\mathcal{D}(R)$ are confluent cartesian closed categories whose hom-orders are weakly essentially normalising. Further, $\mathcal{D}(R)$ has a confluent weak natural numbers object.*

Proof Given Theorem 4.2, it remains to prove the existence of the exponential and the natural numbers object. The constructions will be given and the details left to the reader.

If $\Gamma = \{x : X\}$ and $\Delta = \{y : Y\}$ then $\Gamma \to \Delta = \{u : X \to Y\}$. More generally, if $\Gamma = \{x_i : X_i\}$ and $\Delta = \{y_j : Y_j\}$ are as above then $\Gamma \to \Delta = \{u_j : U_j\}$ where

$$U_j = X_1 \to X_2 \to \ldots \to X_m \to Y_j$$

(with $\to$ associated to the right). To define the currying of a declaration $d : \Gamma \times \Delta \to \Theta$ it suffices, since $\Delta \to (-)$ preserves products, to consider the case where $\Theta = \{z : Z\}$ is a single variable. Then the essential normal form of d in the declaration subsystem is an atomic declaration, say $z = e$, whose curried form is

$$\Gamma \xrightarrow{u = \lambda y_1 y_2 \ldots y_n.e} \{u : Y_1 \to Y_2 \to \ldots Y_n \to Z\}$$

Evaluation is then $z = u y_1 y_2 \ldots y_n : (\Delta \to \Theta) \times \Delta \to \Theta$.

Although the existence of products in $\mathcal{D}(R')$ does not depend on the existence of types to represent them, the latter are necessary for the object $\{n : N\}$ to be a natural numbers object, unless the iterator is strengthened to handle multiple recursion. However, in $\mathcal{D}(R)$ every object $\Gamma = \{x_i : X_i\}$ is equivalent to one with only a single variable $\{x : X\}$ where X is the product of the X_i's, and so we will assume Γ to be of that form. Now reduce all declarations to atomic form. Now we can use the term structure to show that the context $\{n : N\}$ is a weak confluent natural numbers object. Note that the compatibility square for the iterator cannot be strengthened to be a reduction, since both sides must reduce to an atomic declaration for the comparison. □

5.4. β-Reduction Through Call

The last model of β-reduction illustrated the nature of categories of declarations, but permitted two different ways of realising substitution, namely by β-reduction on expressions, and the call rule on declarations. By modifying the expression reductions we can use call to perform the substitution arising from β-reduction, just as (β) is sometimes interpreted by `let`.

Define the basic reduction β *through call* or (β') by

$$(\beta') \qquad y = (\lambda x.t_2)t_1 \Rightarrow x = t_1; y = t_2 : \Delta$$

provided that $x \notin \Delta$. Let R'' be the result of replacing (β) by (β') in R.

Lemma 5.4 *Reductions of R are reductions of R''. Conversely, if $d \Rightarrow d'$ in R'' then $d \sim d'$ in R. Hence R'' is confluent and weakly essentially normalising.*

Proof Observe that (β) is given by (β') followed by call and weakening. This implies the first statement, and, since compatibility in R is an equivalence relation, implies the next pair of claims too. Finally, weak essential normalisation follows upon observing that β-normal forms are β'-normal too. □

Theorem 5.5 *$\mathcal{D}(R'')$ is confluently cartesian closed with weak natural numbers object and essentially normal hom-orders.*

Proof The lemma above shows that R'' and R have the same compatibility relation and reversible reductions. □

6. Further Work

One line of development is to stop reduction from occurring under λ's. That is, abolish the ξ-rule. Of course this destroys confluence of expression reduction with call as currently described. A possible solution is to introduce closures, so that call defers substitution under λ's as in the $\lambda\sigma$-calculus [ACC90].

Another possible development is to restrict call on $x = t_1; y = t_2$ so that t_1 is required to be a *value* of some kind, e.g. a weak head normal form. This results in something of the flavour of call-by-need semantics (by-need since weakening can always remove unwanted arguments). The expression t_2 could be similarly constrained to capture some aspects of spine reduction (studied by Barendregt and Levy) in which the body of an evaluated λ-term is reduced before the value is passed. These restrictions increase the extent to which parallel reduction of a declaration is enforced. Various reduction

strategies, such as call-by-value, call-by-name and call-by-need may perhaps be conveniently expressed as reduction strategies in a category of declarations.

Beyond these considerations lie the problems of extending this work to a polymorphic setting, and to full recursion.

Acknowledgements My warm thanks go to P.-L. Curien, N. Ghani, S. Peyton-Jones, G.D. Plotkin and S. Soloviev for many useful discussions.

References

[ACC90] M. Abadi et al, *Explicit substitutions. INRIA-Rocuencourt Rapports de Recherche 1176 (1990).*

[CKS84] A. Carboni, S. Kasangian and R. Street, *Bicategories of spans and relations. J. Pure and Appl. Alg. 33 (1984) 259 – 267.*

[Coc90] J.R.B. Cockett, *List-arithmetic distributive categories: locoi. J. Pure and Appl. Alg. 66 (1990) 1–29.*

[HR91] B.P. Hilken and D.E. Rydeheard, *Towards a categorical semantics of type classes. Fundamenta Informatica, to appear.*

[HHM89] C.A.R. Hoare, He Jifeng & C.E. Martin, *Pre-adjunctions in Order Enriched Categories. Oxford University Computing Laboratory (1989).*

[Hue76] G. Huet, *Résolution d'équations dans des langages d'ordre* $1, 2, \ldots, \omega$. *Thèse d'Etat, Université de Paris VII, 1976.*

[CKW89] A. Carboni, G.M. Kelly & R.J. Wood, *A 2-categorical approach to Geometric morphisms, I. Sydney Category Seminar Reports 89–19 (1989).*

[Jay88] C.B. Jay, *Local adjunctions. J. Pure and Appl. Alg. 53 (1988) 227–238.*

[Jay90a] C.B. Jay, *Extendinging properties to categories of partial maps. Tech. Rep. ECS-LFCS 90-107.*

[Jay90e] C.B. Jay, *Partial functions, ordered categories, limits and cartesian closure. in: G. Birtwistle (ed), IV Higher Order Workshop, Banff, 1990. (Springer, 1991).*

[Jay91a] C.B. Jay, *Tail recursion from universal invariants. in: D.H. Pitt et al (eds), Category Theory and Computer Science Paris, France, September 1991 Proceedings. Lecture Notes in Computer Science 530 (Springer, 1991) 151–163.*

[Jay91b] C.B. Jay, *Long $\beta\eta$ normal forms and confluence (revised) preprint.*

[Jay91c] C.B. Jay, *Modelling Reduction in Confluent Categories. Tech. Rep. ECS–LFCS–91-187.*

[LS86] J. Lambek and P. Scott, *Introduction to higher order categorical logic. Cambridge Studies in Advanced Mathematics 7 (Cambridge Univ. Press, 1986).*

[Min79] G.E. Mints, *Teorija categorii i teoria dokazatelstv.I. in:* Aktualnye problemy logiki i metodologii nauky, Kiev, 1979 *252-278.*

[PR89] R. Paré and L. Roman, *Monoidal categories with natural numbers object. Studia Logica 48(3) (1989).*

[Plo75] G.D. Plotkin, *Call-by-name, call-by-value and the λ-calculus. Theoretical Computer Science 1 (1975) 125–159.*

[Plo81] G.D. Plotkin, *A structural approach to operational semantics. Tech. Rep. Aarhus University, Daimi FN-19.*

[Rom89] L. Roman, *Cartesian categories with natural numbers object. J. of Pure and Appl. Alg. 58 (1989) 267–278.*

[RS87] D.E. Rydeheard & J.G. Stell, *Foundations of equational deduction: A categorical treatment of equational proofs and unification algorithms. in: Pitt et al, (eds), Category Theory and Computer Science. Lecture Notes in Computer Science 283 (Springer, 1987) 114 – 139.*

[See87] R.A.G. Seely, *Modelling computations: a 2-categorical framework. in: Proceedings of the Second Annual Symposium on Logic in Computer Science (1987).*

7. Appendix: the Declaration Reduction System

Here is the proof of confluence and essential normalisation for the declaration reduction system, and a description of the essential normal forms. It is then straightforward to confirm that the binary products defined above are confluent cartesian products with terminal object given by the empty context.

The presence of restrictions on the call rule occasioned by the rebinding of identifiers obscures the nature of declaration reduction. Recall that a call of $x = t_1$ to $y = t_2$ in $d : \Delta$ may be forbidden if y equals x or is free in t_1. This can only happen if y appeared earlier, either as an identifier or in $\mathtt{dom}(d : \Delta)$ in which case we say that the leftmost appearance of the variable y as an

identifier is a *duplicate identifer* in $d : \Delta$. A *singular* declaration is a typed declaration $d : \Delta$ which has no duplicate identifiers.

In general it is impossible to eliminate duplicate identifiers in a typed declaration $d : \Delta = \{y_i : Y_i\}$ for $1 \leq i \leq n$ entirely, as was shown in (2). We will see, however, that it suffices to restrict our attention to a subclass of declarations for which this is possible. Let $\Delta' = \{y'_i : Y_i\}$ be a context whose variables are neither in Δ nor in $\mathrm{dom}(d : \Delta)$ and let d' be $y'_1 = y_1; \ldots; y'_n = y_n$. Then $\mathrm{dom}(d' : \Delta') = \Delta$ and

$$d \Rightarrow d_1 : \Delta \text{ iff } d; d' \Rightarrow d_1; d' : \Delta'$$

Thus to establish confluence and essential normalisation, it suffices to assume, as we will for the rest of this appendix, that every variable in Δ appears exactly once as an identifier in d and that $\Delta \cap \mathrm{dom}(d : \Delta) = \phi$. As no reductions increase the minimum domain, reductions preserve this property.

Now duplicate identifiers will be eliminated by relabelling. Consider an atomic declaration $z = t$ appearing in a typed declaration $d = d_1; z = t; d_2 : \Delta$ and a variable u of the same type as z such that $u \notin d_2 : \Delta$. The *relabelling* of this z by u in $d : \Gamma \rightarrow \Delta$ is the result of the renaming $d \Rightarrow d_1; u = t; z = u; d_2$ followed by as many calls to $z = u$ as are legal, then thinning or weakening it if possible. By convention, if the designated z is not there (as a result, say, of weakening) then the relabelling has no effect. Relabelling of a duplicate identifier will always result in either thinning (if it appears again as an identifier) or weakening (if it is in the minimum domain) and so reduce the number of duplicate identifiers.

Variables which are neither identifiers in d nor in $\mathrm{dom}(d : \Delta)$ are *fresh* for $d : \Delta$.

Lemma 7.1 *Relabelling by of a duplicate identifier z in $d : \Delta$ by a fresh variable u is reversible (by relabelling u by z). Hence if*

$$d_0 \Rightarrow d_1 \Rightarrow \ldots \Rightarrow d_n$$

is a sequence of 1-step reductions then there is a sequence of 1-step reductions

$$d'_0 \Rightarrow d'_1 \Rightarrow \ldots \Rightarrow d'_n$$

between singular terms with $d_i \Leftrightarrow d'_i$ and whose steps are reversible iff the corresponding steps of the original sequence are.

Proof The first statement is self-explanatory. Thus each d_i can be reversibly reduced to a singular form by the following process. Find the leftmost occurence of a duplicate identifier z in d_i and rename it by a variable u

which is of the same type and fresh for all the d_i's. Now a trivial case analysis shows that the relabellings commute with the 1-step reductions. Since the relabellings are all reversible, the 1-step reductions of the original sequence are reversible iff their counterparts in the new sequence are. □

Thus we can restrict our attention to 1-step reductions between singular terms and hence the retrictions on call never arise, and thinning is no longer relevant. Thus we can restrict our attention to the subsystem whose reductions are call and weakening, and renaming by fresh identifiers.

Lemma 7.2 *If $d_0 \Rightarrow d_1 \Rightarrow \ldots \Rightarrow d_m$ is a sequence of 1-step reductions then there is a similar reduction sequence*

$$d_0 \Rightarrow d'_1 \Rightarrow \ldots d'_n \tag{3}$$

of the same essential length which does not use renaming, such that there is a reversible reduction $d'_n \Rightarrow d_m$.

Proof We will see that we can choose $d'_n \Rightarrow d_m$ as an initial fragment of a relabelling, which is thus reversible. The proof is by induction on m. If $m = 0$ then there is nothing to prove. If $m = k + 1$ then we have a reduction sequence of the same type as (3) whose essential length is that of the first k steps of the original sequence. Now a simple case analysis shows that if $d'_n \Rightarrow d_k$ is an initial fragment of the relabelling of some z by u to obtain d''_n then completing the relabelling of that z by u in d_{k+1} is either d''_n again or a 1-step reduct of it. As fragments of relabellings are reversible, the essential length is preserved. □

Thus we can ignore renaming, and stick to call, weakening and nil. Now nil can always be postponed till the end, and is always reversible, and weakening can be delayed till after call, at which point it is confluent and strongly normalising. So we have reduced the problem to considering call alone, without any restrictions.

Theorem 7.3 *Call reduction is essentially normalisable.*

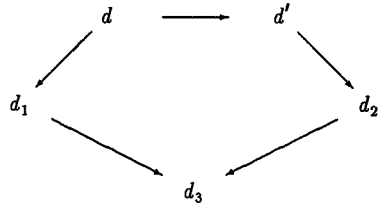
Proof The amount of *interaction* in a singular declaration is defined by

$$\begin{aligned}
\mathtt{Int}(\mathtt{nil}) &= 0 \\
\mathtt{Int}(z = t) &= 0 \\
\mathtt{Int}(d; x = t_1; y = t_2) &= \mathtt{Int}(d; x = t_1) + \mathtt{Int}(d; y = t'_2) + C(x, t_2) \\
C(x, t_2) &= \begin{cases} 1 & \text{if } x \text{ is free in } t_2 \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

where $t_2' = t_2[t_1/x]$.

Now call applied to a singular declaration is either a permutation and leaves the interaction unchanged, or strictly reduces the amount of interaction. □

For confluence we must modify the usual result linking it with strong normalisation. Let R be a reduction system, and let R' be the system whose 1-step reductions are given by a reversible reduction followed by a 1-step reduction of R. Define R to be *weakly essentially confluent* if R' is weakly confluent. Equivalently, this means that if $d \Leftrightarrow d'$ is reversible with $d \Rightarrow d_1$ and $d' \Rightarrow d_2$ then d_1 and d_2 have a common reduct d_3



Theorem 7.4 *If R is weakly essentially normalising and weakly essentially confluent then it is confluent.*

Proof This follows from the usual result for R' (strong normalisation plus weak confluence implies confluence). □

Hence confluence of call will follow from weak essential confluence. There is only one critical pair to consider, namely the reductions of $x = t_1; y = t_2; d; z = t_3$ given by calling x in y, and by calling y in z (after permutation through d), which is completed as follows:

$$\begin{aligned} x = t_1; y = t_2; d; z = t_3 &\Rightarrow y = t_2'; x = t_1; d; z = t_3 \\ &\Rightarrow z = t_4[t_1/x][t_2'/y]; y = t_2'; x = t_1; d \end{aligned}$$

while

$$\begin{aligned} x = t_1; y = t_2; d; z = t_3 &\Rightarrow x = t_1; d; z = t_3'; y = t_2 \\ &\Rightarrow z = t_4'[t_1/x]; y = t_2'; x = t_1; d \end{aligned}$$

where t_2' is defined as above and $t_3' = t_3[t_2/y]$ while t_4 is the result of calling d across z and t_4' is the result of calling d across t_3'. The resulting terms are equal as $t_4[t_1/x][t_2'/y] = t_4[t_2/y][t_1/x] = t_4'[t_1/x]$ since y permutes with d.

This completes the proof of confluence and essential normality of the declaration reduction system. Clearly the essential normal forms all have no interaction, and so are exactly the conjunctions in which every variable in the codomain appears once.