$\reflectbox{$\lambda$}$

DIMITRI HENDRIKS and VINCENT VAN OOSTROM
Department of Philosophy, Universiteit Utrecht
P.O. Box 80089, 3508 TB Utrecht, The Netherlands
(*e-mail:* `hendriks,oostrom@phil.uu.nl`)

Abstract

We make the notion of scope in the λ-calculus explicit. To that end, the syntax of the λ-calculus is extended with an end-of-scope operator $\reflectbox{$\lambda$}$, matching the usual opening of a scope due to λ. Accordingly, β-reduction is extended to the set of scoped λ-terms by performing *minimal* scope extrusion before performing replication as usual. We show confluence of the resulting scoped β-reduction. Confluence of β-reduction for the ordinary λ-calculus is obtained as a corollary, by extruding scopes *maximally* before forgetting them altogether. Only in this final forgetful step, α-equivalence is needed. All our proofs have been verified in Coq.

1 Introduction

Performing a substitution $M[x{:=}N]$ in the λ-calculus can be decomposed into two subtasks: replicating N an appropriate number of times, and renaming in M in order to prevent unintended capture of variables of N. Indeed, the defining clauses of Curry's definition of substitution, see e.g. C.1 DEFINITION of (Barendregt, 1984), can be neatly partitioned into those dealing with replication (the variable and application clauses) and those dealing with renaming (the abstraction clauses). In this chapter we will focus on trying to understand the latter subtask. We do so, by extending λ-calculus with an explicit operator representing the (end of the) scope of a name, while leaving replication implicit.

Abstractions in the λ-calculus can be viewed as being composed of two parts: one part which is dual to application, and another which causes the opening of the scope of the bound variable. The scope of the binder λx in $\lambda x.M$ is (implicitly) assumed to extend to the whole of M. Hence to make the notion of scope explicit, it suffices to introduce an operator expressing the end of the scope of λx. This operator is denoted by $\reflectbox{$\lambda$}$ (adbmal). $\reflectbox{$\lambda$}x.M$ expresses that the scope of x is ended 'above' M. For instance, in the $\reflectbox{$\lambda$}$-term $\lambda x.\reflectbox{$\lambda$}x.\underline{x}$ the underlined occurrence of the variable x is free, since the binding effect of the λx is undone by the subsequent $\reflectbox{$\lambda$}x$. For another example, only the underlined occurrence of x is free in $\lambda x.x(\reflectbox{$\lambda$}x.\underline{x})x$; the first and third occurrences of x are in scope of the λx (see Figure 1).

Definition 1.1
The set $(M, N, P \in)\Lambda$ of $\reflectbox{$\lambda$}$-*terms* is defined by:

$$\Lambda ::= \mathcal{V} \mid \reflectbox{λ}x.\Lambda \mid \lambda x.\Lambda \mid \Lambda\Lambda$$

where $(x, y, z \in)\mathcal{V}$ is a collection of variable(name)s with decidable equality:

Axiom 1 (*Names with decidable equality*)
$x = y \vee x \neq y$, for all $x, y : \mathcal{V}$

We adopt the usual notational conventions for the λ-calculus (Barendregt, 1984), treating $\adbmal$ analogously to λ. λ-terms are embedded as $\adbmal$-terms without occurrences of $\adbmal$.

In order to extend the notions of α-equivalence and β-reduction, we should first try to make some semantic sense of $\adbmal$s. Thinking of λx and $\adbmal x$ as (named) opening '$[_x$' and closing '$]_x$' brackets,[1] it is clear that $\adbmal$-terms may come in different degrees of balancedness. For instance, scopes could seemingly be crossing one another as indicated by the boxes in:

$$P = \lambda x.\lambda y.\adbmal x.\adbmal y.\,Q$$

This would obviously cause semantical problems (try to define substitution). To overcome this problem we assume a simple minded jump semantics: an occurrence of $\adbmal x.M$ implicitly ends the scopes of all (non-matching) λs inbetween that occurrence and its matching λx, just as the occurrence of the variable x in $\lambda x.\lambda y.x$ can be thought of as implicitly ending the scope of the λy. Hence P is semantically equivalent to $\lambda x.\lambda y.\adbmal y.\adbmal x.\adbmal y.Q$. Our definitions of α-equivalence and β-reduction and hence our definition of substitution, as will be presented below, are meant to reflect this intuitive (operational) semantics.

Apart from such *jump* terms we identify the useful subclasses of scope-balanced and balanced terms, both of which are closed under α-equivalence and β-reduction. Balanced terms can be used to represent nameless λ-terms using de Bruijn-indices, by using only a single name. Ordinary λ-terms are not (necessarily) balanced, however they always are scope-balanced.

Definition 1.2
A term is *scope-balanced* if it is scope-balanced *under* some stack X. A term M being *scope-balanced under* a stack X is denoted by $\langle X\rangle M$ and defined by:

$$\frac{}{\langle X\rangle x} \qquad \frac{\langle Xx\rangle M}{\langle X\rangle \lambda x.M} \qquad \frac{\langle X\rangle M}{\langle Xx\rangle \adbmal x.M} \qquad \frac{\langle X\rangle M \quad \langle X\rangle N}{\langle X\rangle MN}$$

Balancedness is defined as scope-balancedness restricting the first clause to

$$\frac{}{\langle Xx\rangle x}$$

Here Xx is the result of pushing x on the stack X.

For instance, $\lambda x.\lambda y.\adbmal x.M$ is not scope-balanced (λy not closed before $\adbmal x$), $\lambda y.x$ is scope-balanced but not balanced (λy not closed before x), and $\lambda x.\adbmal x.M$ and $\lambda x.x$ are balanced (if M is in the former case).

[1] But note that brackets (parentheses) usually apply 'horizontally' to the textual representation of terms, whereas λ and $\adbmal$ apply 'vertically' to their abstract syntax trees (where brackets do not even occur).

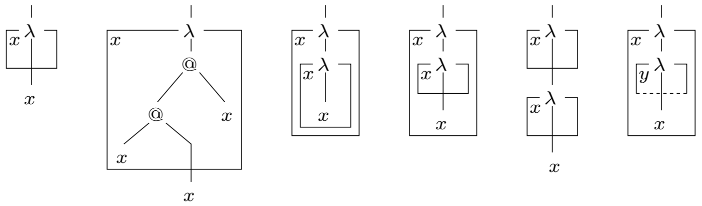

Fig. 1. $\lambda x.\reflectbox{$\lambda$}x.x$, $\lambda x.x(\reflectbox{$\lambda$}x.x)x$, $\lambda x.\lambda x.x$, $\lambda x.\lambda x.\reflectbox{λ}x.x$, $\lambda x.\reflectbox{$\lambda$}x.\lambda x.\reflectbox{λ}x.x$, and $\lambda x.\lambda y.x$.

It is easy to see that closed λ-terms are scope-balanced under any stack, hence in particular under the empty stack $\square$. Scopes in balanced $\reflectbox{$\lambda$}$-terms can be neatly visualised as boxes in their abstract syntax tree, as shown in Figure 1.[2] Vice versa, in the term representation of a box, only its 'doors' are kept. That is, λs and $\reflectbox{$\lambda$}$s are used to demarcate all places where the boundary of the box is crossed by the abstract syntax tree. In fact, there is a strong similarity (see Figure 1) between balanced terms and the context-free string language of *matching brackets* as presented by the grammar:

$$P \quad ::= \quad \epsilon \mid [P] \mid PP$$

- Scopes can be nested (similar to $[P]$). In the λ-term $\lambda x.\lambda x.x$, the occurrence of x is implicitly assumed to be bound by the rightmost λx. Similarly, the scope of the rightmost λx is ended by the $\reflectbox{$\lambda$}x$ in $\lambda x.\lambda x.\reflectbox{λ}x.x$.
- Scopes can be concatenated (similar to PP). In the $\reflectbox{$\lambda$}$-term $\lambda x.\reflectbox{$\lambda$}x.\lambda x.\reflectbox{λ}x.x$, the scopes of the two λxs do not have overlap/are not nested, in spite of the latter being 'to the right' of the former.

Indeed, the set of balanced $\reflectbox{$\lambda$}$-terms can be generated by a so-called context-free term grammar, where context-free term grammars are the natural generalization of context-free string grammars, see e.g. Section 2.5 of (Comon *et al.*, 1997). A difference between matching bracket strings and balanced $\reflectbox{$\lambda$}$-terms is that, due to the branching structure of terms, *several* $\reflectbox{$\lambda$}$'s may match the same λ as in $\lambda x.(\reflectbox{$\lambda$}x.\underline{x})(\reflectbox{λ}x.\underline{x})$, with both underlined occurrences of x free.

Related work This is the full version of (Hendriks & van Oostrom, 2003b), and will be part of the first author's forthcoming PhD thesis (2003). Compared to (Hendriks & van Oostrom, 2003b) the results in the present paper are more general, in particular the key substitution lemmas. Moreover, we have supplied more formal definitions and some typical proof ideas. For the complete proof development we refer the reader to (Hendriks & van Oostrom, 2003a).

When application of $\reflectbox{$\lambda$}x$ is restricted to variables (and end-of-scopes), it corresponds to Berkling's lambda-bar (1976), which is in turn seen to be a named version of the successor operator in de Bruijn's nameless (more precisely: single name) calculus (1972). Their calculi do not allow successions of boxes, only nestings of boxes.

[2] Scopes in non-balanced terms can be drawn as floorless boxes ($\lambda x.\lambda y.x$ in Figure 1).

This corresponds to the sublanguage of the language of matching brackets (see above) generated by the grammar: $B ::= \epsilon \mid [B]$.

Restricting to a single name, i.e. to de Bruijn-indices, $\reflectbox{$\lambda$}x$ corresponds to the shift substitution $[\uparrow]$ in the λ-calculus with explicit substitutions $\lambda\sigma$ of (Abadi *et al.*, 1991), or the shift operation *Shi* of (Chen & Xi, 2002). The earliest generalisation of de Bruijn indices seems to be due to Paterson (1991). The idea is to allow the successor S to appear on subterms, instead of just on indices; as it is written in (Bird & Paterson, 1999):

> Substitution on de Bruijn terms transforms arguments as well as function bodies, thus precluding sharing. Consider the example term from Section 1, with the variables written in unary notation:
>
> $$\lambda.0\,(\lambda.\mathsf{S}0\,0\,(\lambda.\mathsf{SS}0\,\mathsf{S}0\,0))$$
>
> If this term is applied to the term $\lambda.0\,\mathsf{S}0$, the result is
>
> $$(\underline{\lambda.0\,\mathsf{S}0})\,(\lambda.(\underline{\lambda.0\,\mathsf{SS}0})\,0\,(\lambda.(\underline{\lambda.0\,\mathsf{SSS}0})\,\mathsf{S}0\,0))$$
>
> where the three versions of the argument are underlined. There is a generalisation of de Bruijn notation in which S can be applied to any term, not just a variable (Paterson, 1991). Its effect is to escape the scope of the matching λ. With this looser representation of terms, one can avoid transforming arguments while substituting. In the above example, substitution yields
>
> $$(\underline{\lambda.0\,\mathsf{S}0})\,(\lambda.\mathsf{S}(\underline{\lambda.0\,\mathsf{S}0})\,0\,(\lambda.\mathsf{SS}(\underline{\lambda.0\,\mathsf{S}0})\,\mathsf{S}00))$$
>
> In effect, we have postponed pushing the S's down to the variables.

where the 'example term from Section 1' is the λ-term $\lambda x.x\ (\lambda y.x\ y\ (\lambda z.x\ y\ z))$ which translates to $\lambda 0.\ (\lambda.1\ 0\ (\lambda.2\ 1\ 0))$ in (their) de Bruijn notation. Applying the λ-term to a named version, say $\lambda x.xy$ of $\lambda.0\,\mathsf{S}0$, yields (see Subsection 4.2) $(\underline{\lambda x.xy})\ (\lambda y.(\reflectbox{λ}y.\underline{\lambda x.xy})\ y\ (\lambda z.(\reflectbox{λ}z.\reflectbox{λ}y.\underline{\lambda x.xy}))\ y\ z)$. So obviously $\reflectbox{$\lambda$}$ corresponds to the successor S in de Bruijn notation.

Bird and Paterson go on to show that in the balanced (single name) case the term language of the $\reflectbox{$\lambda$}$-calculus is context-free by presenting it by means of the following context-free term grammar:

$$\begin{aligned} \mathit{Term}\ a &::= \mathit{Var}\ a \mid \mathit{App}(\mathit{Term}\ a, \mathit{Term}\ a) \mid \mathit{Abs}(\mathit{Term}(\mathit{Incr}(\mathit{Term}\ a))) \\ \mathit{Incr}\ a &::= \mathit{Zero} \mid \mathit{Succ}\ a \end{aligned}$$

the idea being that *Term*s are balanced by generating *Incr*s, i.e. variables (*Zero*s) or end-of-scopes (*Succ*s), at the same time as their matching *Abs* (abstraction).[3]

When restricting to the balanced case, our boxes correspond closely to boxes in MELL proof nets for linear logic, see e.g. (Lafont, 1995). In fact, in our optimal implementation (see Section 5) λx disintegrates into a λ (a par in MELL) and (part of the boundary of) an x-box ((Asperti's version of) a box in MELL), upon encountering an application. One can think of these two phases of abstraction as turning a free variable x into a bound one by closing it off from the outside world inside an x-box, but providing a handle to x to the outside world again in the form of the λ. Many proposals for decomposing abstraction into more elementary

[3] This does not work (directly) for non-balanced terms in the many-variable case.

notions can be found in the literature, a recent one being (Baro & Maurel, 2003). Similarly, notions of enclosure abound. Analogous to the conflation of the enclosure with the enclosed as found in (the etymology of) words such as town, garden, park and paradise, these formalizations may or may not make the boundary explicit, see e.g. (Cardelli & Gordon, 1998; Parrow, 1998; Blom, 2001).

In the area of dynamic semantics for natural language, a stack-based semantics for a variant of predicate logic is presented in (Hollenberg & Vermeulen, 1996). Although, the exact relationship is not clear to us yet, a difference seems to be that in their semantics every variable has its own stack, whereas we have a single stack. However, also in (Berkling, 1976) variables have their own stack.

Implementation All results informally presented here are formalised in Coq. The source files are available from (Hendriks & van Oostrom, 2003a). The size of the development is 9543 lines of Coq-code, 259160 bytes; 324 lemmas are proved. The development time is estimated one man-year approximately.

Outline The outline of the rest of this paper is as follows. In Section 2 we define some preliminary notions on abstract rewriting systems. We provide several definitions of α-equivalence for $\rotatebox[origin=c]{180}{\ensuremath{\lambda}}$-terms in Section 3, extending classical definitions as found in the literature on the λ-calculus, prove them to be decidable congruence relations, and show them to be equivalent. Then we present a definition of β-reduction for $\rotatebox[origin=c]{180}{\ensuremath{\lambda}}$-terms in Section 4, extending the usual definition for the λ-calculus, and prove this notion of reduction to be confluent *without* α-equivalence. In both (α and β) cases it is shown how the results on the $\rotatebox[origin=c]{180}{\ensuremath{\lambda}}$-calculus entail the corresponding results for the ordinary λ-calculus, e.g. confluence of β-reduction up to α-equivalence. Applications are presented in Section 5. Finally, in Section 6, we conclude, and discuss upon the relationship of the $\rotatebox[origin=c]{180}{\ensuremath{\lambda}}$-calculus to explicit substitution calculi having the property of preservation of normalisation.

Acknowledgments We would like to thank the participants of the TCS seminar at the Vrije Universiteit Amsterdam, PAM and the 7th Dutch Proof Tools Day both at CWI, Amsterdam, ZIC at the Technische Universiteit Eindhoven, the CS seminar at the University of Leicester, and the TF lunch seminar at the Universiteit Utrecht, for feedback. Eduardo Bonelli, Marko van Eekelen, Joost Engelfriet, Stefan Kahrs, Kees Vermeulen, Albert Visser, and the CADE referees provided useful comments and pointers to the literature.

2 Preliminaries

Definition 2.1
We define the n-fold composition $\to^n$ of a binary relation $\to$ as the smallest relation satisfying the following clauses:

$$\frac{}{x \to^0 x}\ \text{reflexivity} \qquad \frac{x \to y}{x \to^1 y}\ \text{extension} \qquad \frac{x \to^n y \quad y \to^m z}{x \to^{n+m} z}\ \text{transitivity}$$

The reflexive and transitive closure $x \to^* y$ of $\to$ is defined as $\exists n.x \to^n y$. The *equivalence closure* of $\to$ is denoted $\leftrightarrow^*$ and defined inductively by the former three clauses (replacing all occurrences $\to^k$ with $\leftrightarrow^*$) plus the following one.

$$\frac{y \leftrightarrow^* x}{x \leftrightarrow^* y}\ \text{symmetry}$$

Definition 2.2
A binary relation R on a set A has the *diamond* property, if for all $a, b, c : A$, $a\ R\ b$ and $a\ R\ c$ implies there exists $d : A$, such that $c\ R\ d$ and $b\ R\ d$. R is *confluent* if its reflexive–transitive closure R^* has the diamond property.

Definition 2.3
We say R *transits* S if $R \subseteq S \subseteq R^*$, where $R_1 \subseteq R_2$ is defined by $\forall x, y.\, x\ R_1\ y \Rightarrow x\ R_2\ y$.

Lemma 2.4
If R has the diamond property, then it is confluent.

Proof
(See, for example, (Terese, 2003).) By induction on the complexity of the diverging steps, loading it by: converging steps have the same complexity as opposite diverging steps. Here we express the complexity by the number of R-steps. The following statement is proved by induction on the diverging steps.

$$x\ R^n\ y \wedge x\ R^m\ z \Rightarrow \exists u.y\ R^m\ u \wedge z\ R^n\ u \quad \Box$$

Lemma 2.5
If R transits S and S has the diamond property, then R is confluent.

Proof
By monotonicity (if $R \subseteq R'$, then $R^* \subseteq R'^*$) and idempotence ($R^{**} = R^*$) of * and the first assumption, we have $R^* \subseteq S^* \subseteq R^*$. We conclude from the previous lemma and the second assumption. $\Box$

3 α

We present three distinct definitions of α-equivalence for the λ-calculus known from the literature, in historical order. We then compare these notions, present our adaptations of each of them to the $\lambda\!\!\!/$-calculus, and prove them to be equivalent. For this the existence of fresh variables is required.

Axiom 2 (*Fresh variable*)
For any given (finite) stack of variables, there is a variable not among these, i.e. a fresh variable, $\forall X.\exists x.x \notin X$.

3.1 $\lambda\alpha$

3.1.1 Church

Our first notion of α-equivalence is the usual one based on Church's Postulate I for the λ-calculus (1932), which reads (page 355):

If J *is true, if* L *is well-formed, if all the occurrences of the variable* x *in* L *are occurrences as a bound variable, and if the variable* y *does not occur in* L*, then* K*, the result of substituting* $\mathsf{S}^{\mathsf{x}}_{\mathsf{y}}\,\mathsf{L}|$ *for a particular occurrence of* L *in* J*, is also true.*

where $\mathsf{S}^{\mathsf{X}}_{\mathsf{Y}}\,\mathsf{U}|$ *represents the formula which results when we operate on the formula* U *by replacing* X *by* Y *throughout, where* Y *may be any symbol or formula but* X *must be a single symbol, not a combination of symbols* (page 350 of (Church, 1932)).

Due to Curry, Postulate I is nowadays known as the α-conversion rule. An α-conversion *step* is obtained from the α-conversion rule by allowing its application to any subterm of a term. An α-*conversion* consists of a sequence of α-conversion steps. Finally, a term is said to be α-equivalent to another one, if there exists an α-conversion from the former to the latter.

An advantage of this definition is that it is operational and fine-grained; each α-conversion step itself is easy to understand since it does only little work. A disadvantage of this fine-grainedness is that it is at first sight not clear whether structural properties such as symmetry and decidability of α-conversion hold. Moreover, it needs the Fresh variable axiom due to the *Extra-hand principle*: if both your hands are full, you need a third hand in order to swap their contents.[4]

Example 3.1
The terms $\lambda x.\lambda y.xy$ and $\lambda y.\lambda x.yx$ are α-equivalent. However, both α-conversion steps replacing x by y and vice versa are forbidden. Hence, an α-conversion needs to introduce a third, fresh, variable, say z, first:

$$\lambda x.\lambda y.xy \to_\alpha \lambda z.\lambda y.zy \to_\alpha \lambda z.\lambda x.zx \to_\alpha \lambda y.\lambda x.yx$$

3.1.2 Schroer

In order to prove symmetry and decidability of α-equivalence as defined in the previous paragraph, one may try to find a strategy for α-conversion such that the number of α-conversion steps needed in a conversion from s to t is bounded by, say, the sum of the sizes of s and t. An obvious way to bound this number is by restricting α-conversion by:

> Never rename twice.

However, from Example 3.1 we immediately see that this is too strict a restriction; the leftmost λ-abstraction needs to be renamed twice. Hence renaming once is not enough, but, as the example suggests our assumption may be replaced by:

> Never rename thrice.

Such an idea appears at least as early as Schoer's PhD thesis, see page 384 of his (1965):

[4] There is the well-known way to swap the contents of two registers *in situ* by performing three exclusive-or's (xor); in Java: `r1 ^= r2; r2 ^= r1; r1 ^= r2` where `op1 ^= op2` is equivalent to `op1 = op1 ^ op2` and `^` is bitwise xor. Here, we will *not* assume the structure needed for this, e.g. a Boolean ring on the variables.

```
          Scholium 3.44. The proof of Theorem 3.44 below
has as its germ the following procedure to determine of
A,B ε Wocc  whether or not  A adj B: Let Z1, Z2,... be
singleton expressions of the alphabetically earliest
variables not occurring at all in either of  A,B ,
enumerated without repetitions. In each of  A,B , change
quantifiers from left to right, replacing the given variables
by the  Z's  in order. There will result  A',B'  such that
A adj A' . B adj B' , and such that  A adj B . ≡ . A' = B' .
```

where `adj` is his notion of α-equivalence and `Theorem 3.44` states decidability.

Example 3.2
Applied to Example 3.1 Schroer's procedure yields:

$$\lambda x.\lambda y.xy \to_\alpha \lambda z_1.\lambda y.z_1 y \to_\alpha \lambda z_1.\lambda z_2.z_1 z_2 \leftarrow_\alpha \lambda z_1.\lambda x.z_1 x \leftarrow_\alpha \lambda y.\lambda x.yx$$

Of course, to prove that this is an α-conversion one needs to prove that the last two backward α-steps are forward α-steps as well; they are.

Symmetry of a definition based on Schroer's procedure is trivial, decidability and reflexivity are also not too difficult, but now transitivity is not so simple because of the choosing of the `alphabetically earliest variables not occurring at all in either of  A,B` which may differ for `A,B` and `B,C`, when proving `A adj C`.[5] Also note that the procedure is *not* very parsimonious; it allocates as many fresh variables as there are λ-abstractions (quantifiers) in a term, where a single one (one extra hand) would suffice, as noted, e.g., by (Gabbay & Pitts, 1999). This fact may be seen by proceeding in a top-down fashion, the only interesting case being abstraction:

(abstraction) Suppose $s = \lambda x.s'$ and $t = \lambda y.t'$, such that the variables 'above' them have already been made identical. We proceed by first converting *every* x in s into z. Next, *every* y is converted into x and finally, *every* z is converted into y, resulting in, say, $\hat{s}$. Now $\hat{s}$ and t have the same initial binder, and we proceed on the respective subterms. To prove that this procedure is correct, one uses that y does not occur free in s' since otherwise s and t would not be α-equivalent. Symmetrically, x does not occur free in t'.

Example 3.3
α-converting, say, $\lambda x_1 x_2 x_3.x_1 x_2 x_3$ into $\lambda x_2 x_3 x_1.x_2 x_3 x_1$ using this procedure proceeds as follows. First, we swap, using a fresh variable y, x_1 with x_2 yielding $\lambda x_2 x_1 x_3.x_2 x_1 x_3$. Hence the first variable has been appropriately renamed, and we may proceed on the respective subterms. In order to α-convert $\lambda x_1 x_3.x_2 x_1 x_3$ into $\lambda x_3 x_1.x_2 x_3 x_1$, we swap, using the same y, x_1 and x_3 in the former yielding the latter and we are done (formally one needs to continue with twice the subterm $\lambda x_1.x_2 x_3 x_1$, but nothing 'happens' anymore.)

[5] Compared to Church's α-conversion Schroer's procedure needs variables to be alphabetically sorted. Here, we will *not* assume the structure needed for this (e.g. a well-order) on the collection of variables.

$$\frac{}{\epsilon \vdash x = x} \qquad \frac{}{\Gamma, x = y \vdash x = y} \qquad \frac{v \neq x \quad y \neq z \quad \Gamma \vdash v = z}{\Gamma, x = y \vdash v = z}$$

$$\frac{x, y \in \mathrm{Var} \quad \Gamma \vdash x = y}{\Gamma \vdash x \equiv y} \qquad \frac{F \in \mathit{Sym}}{\Gamma \vdash F \equiv F} \qquad \frac{\Gamma, x = y \vdash t \equiv u}{\Gamma \vdash [x]t \equiv [y]u} \qquad \frac{\Gamma \vdash A \equiv C \quad \Gamma \vdash B \equiv D}{\Gamma \vdash AB \equiv CD}$$

Fig. 2. Proof rules for α-congruence (Kahrs, 1993).

3.1.3 Kahrs

Both the problem of showing transitivity and the need for the Fresh variable axiom can be overcome by making renaming implicit. That is, instead of explicitly relating terms by explicitly renaming variables, one may set up an (implicit) correspondence between their respective variables. For instance, the two terms in Example 3.1 are shown α-equivalent by letting x and y in the first correspond to y and x in the second. However, the correspondence needs more structure than just a bijection between the sets of variables in both terms.

Example 3.4
The terms $\lambda x.x\lambda y.y$ and $\lambda x.x\lambda x.x$ are α-equivalent, but this cannot be shown by means of a bijection between variables.

To define α-equivalence inductively, one has to set up a correspondence between *stacks* of variables. Such an idea appears in Kahrs' paper (1993); to quote from it:

> We also define a notion of α-congruence for our terms. It is the usual one, but we shall use it in a slightly more general setting, based on proof rules.
> **Definition 11**. *Sentences* are of the form $\Gamma \vdash t \equiv u$ or $\Gamma \vdash x = y$, where x and y are variables, t and u are terms of the same type and arity, and Γ is an *environment*. An environment is a list $x_1 = y_1, \cdots, x_n = y_n$ of equations between variables. We write ϵ for the empty environment ($n = 0$). A sentence *holds*, if it can be derived by the proof rules in figure 2. (see Figure 2)

One easily proves by induction that α-congruence defined in this way, has all the desired structural properties, e.g. transitivity and decidability. But, of course, it is less clear how to decompose α-equivalence into 'atomic' renaming steps.

3.2 ⅄α

We show that each of the three definitions of α-equivalence can be straightforwardly extended from λ-terms to ⅄-terms. In each case, we highlight the key aspect of our formalization in Coq. We start with some preliminary definitions.

We shall need to test whether a variable is fresh with respect to a term.

Definition 3.1
The relation $x \in M$, saying whether x occurs in M (free or bound) is defined by:

$$\begin{aligned} x &\in y, \text{ if } x = y \\ x &\in \lambda y.M, \text{ if } x = y \text{ or } x \in M \\ x &\in ⅄y.M, \text{ if } x = y \text{ or } x \in M \end{aligned}$$

$$x \quad \in \quad MN, \text{ if } x \in M \text{ or } x \in N$$

We say x *is fresh for* M if $x \notin M$.

As usual, α-conversion is defined using a renaming function to identify expressions that differ only in the names assigned to their bound variables. For $\text{\reflectbox{$\lambda$}}$-terms this means the renaming of *boxes* (viz. Figure 1). Renaming the (outer) x-box in $\lambda x.M$ into a y-box, means to replace λx by λy and to replace all *matching* occurrences of variables x and end-of-scopes $\text{\reflectbox{$\lambda$}}x$ in M by y and $\text{\reflectbox{$\lambda$}}y$ respectively. Matching is performed using a stack to record the binders encountered while descending recursively:

Definition 3.2
Renaming, $M[x{:=}y]$, is defined as $M[x{:=}y,\Box]$, where $M[x{:=}y,Z]$ is defined by the following recursive equations.

$$\begin{array}{rcl}
z[x{:=}y,Z] & = & z, \text{ if } z \in Z \text{ or } z \neq x \\
z[x{:=}y,Z] & = & y, \text{ if } z \notin Z \text{ and } z = x \\
(\text{\reflectbox{λ}}z.M)[x{:=}y,\Box] & = & \text{\reflectbox{λ}}y.M, \text{ if } x = z \\
(\text{\reflectbox{λ}}z.M)[x{:=}y,\Box] & = & \text{\reflectbox{λ}}z.M, \text{ if } x \neq z \\
(\text{\reflectbox{λ}}z.M)[x{:=}y,z'Z] & = & \text{\reflectbox{λ}}z.M[x{:=}y,Z], \text{ if } z = z' \\
(\text{\reflectbox{λ}}z.M)[x{:=}y,z'Z] & = & (\text{\reflectbox{λ}}z.M)[x{:=}y,Z], \text{ if } z \neq z' \\
(\lambda z.M)[x{:=}y,Z] & = & \lambda z.M[x{:=}y,zZ] \\
(M_1M_2)[x{:=}y,Z] & = & M_1[x{:=}y,Z]M_2[x{:=}y,Z]
\end{array}$$

In the fourth clause $\text{\reflectbox{$\lambda$}}z$ implicitly closes the scope of x; therefore, in the result, we can think of $\text{\reflectbox{$\lambda$}}z$ as implicitly closing the scope of y.

Example 3.5
$x\text{\reflectbox{$\lambda$}}x.x\lambda x.(x\text{\reflectbox{λ}}x.x)[x{:=}y] = y\text{\reflectbox{λ}}y.x\lambda x.(x\text{\reflectbox{λ}}x.y)$

In case of ordinary λ-terms the stack Z only grows during renaming, rendering the variable to be renamed inaccessible once it is abstracted from again:

Lemma 3.3
If M is a λ-term, then $(\lambda x.M)[x{:=}y] = \lambda x.M$.

Proof
Note that $(\lambda x.M)[x{:=}y] = \lambda x.M[x{:=}y,x]$. One proves by induction on the λ-term M, that if x occurs in X, then $M[x{:=}y,X] = M$. $\Box$

The following two properties of renaming are needed in the proof of commutation of β-reduction and α-conversion. The first states commutativity of renaming:

Lemma 3.4
$M[x{:=}z,XyY][y{:=}z',X] = M[y{:=}z',X][x{:=}z,Xz'Y]$, if z fresh for y,M,X and z' fresh for M,X.

Secondly, renaming x into z, followed by renaming z into z', amounts to the same as directly renaming x into z':

Lemma 3.5
$M[x{:=}z, X][z{:=}z', X] = M[x{:=}z', X]$, if $z \notin M, X$.

When we project β-reductions in the ʎ-calculus to β-reductions in the λ-calculus (Section 4.4), we need to reason about the set of free variables of a term.

Definition 3.6
The set of free variables of a term M, $\mathsf{FV}(M)$, is defined as $\mathsf{FV}(M, \Box)$, where $\mathsf{FV}(M, X)$ is defined by the following recursive equations.

$$
\begin{array}{rcl}
\mathsf{FV}(x, X) & = & \emptyset, \text{ if } x \in X \\
\mathsf{FV}(x, X) & = & \{x\}, \text{ if } x \notin X \\
\mathsf{FV}(\lambda x.M, X) & = & \mathsf{FV}(M, xX) \\
\mathsf{FV}(ʎ x.M, \Box) & = & \mathsf{FV}(M, \Box) \\
\mathsf{FV}(ʎ x.M, x'X) & = & \mathsf{FV}(M, X), \text{ if } x = x' \\
\mathsf{FV}(ʎ x.M, x'X) & = & \mathsf{FV}(ʎ x.M, X), \text{ if } x \neq x' \\
\mathsf{FV}(MN, X) & = & \mathsf{FV}(M, X) \cup \mathsf{FV}(N, X)
\end{array}
$$

Note that, in the third ʎ-clause, $ʎ x$ implicitly closes the scope x', which is therefore popped from the stack of currently open scopes.

Remark 3.7
Note that for a term M balanced under X it holds $\langle X \rangle M \Rightarrow \mathsf{FV}(M, X) = \emptyset$. This implies that terms which are balanced under the empty stack are closed.

In the sequel it will sometimes be useful to forget about the binding structure of terms. To that end, we map terms to first-order terms by simply forgetting names (better: mapping all names to a single one):

Definition 3.8
The *skeleton* $[M]_{\mathsf{skel}}$ of a term M is a first-order term defined inductively by:

$$
\begin{array}{rcl}
[x]_{\mathsf{skel}} & = & \odot \\
[ʎ x.M]_{\mathsf{skel}} & = & ʎ[M]_{\mathsf{skel}} \\
[\lambda x.M]_{\mathsf{skel}} & = & \lambda[M]_{\mathsf{skel}} \\
[M_1 M_2]_{\mathsf{skel}} & = & @[M_1]_{\mathsf{skel}}[M_2]_{\mathsf{skel}}
\end{array}
$$

For instance, the proof of Lemma 4.24 proceeds by induction over the skeleton of a term, using the fact that renaming preserves skeletons:

Lemma 3.9
$[M[x{:=}y]]_{\mathsf{skel}} = [M]_{\mathsf{skel}}$.

Unary contexts are used to express congruences.

Definition 3.10
Unary contexts are typed $\Lambda \to \Lambda$ and built from

$$[\,], C \circ C', \lambda x.[\,], ʎ x.[\,], M[\,], [\,]M$$

where C, C' are unary contexts. We write CM to denote the result of filling the hole in C by term M; $(C \circ C')M = C(C'M)$.

Now that we have defined some preliminary notions, we are ready to extend the three notions of α-equivalence given in Subsection 3.1 to the $\reflectbox{$\lambda$}$-calculus.

3.2.1 Church

The notion of α-conversion is extended to the $\reflectbox{$\lambda$}$-calculus.

Definition 3.11
The α-rule is $\lambda x.M \to \lambda y.M[x{:=}y]$ if $y \notin M$. Single-step α-renaming, $\to_\alpha$, is defined as the compatible closure of the α-rule:

$$\frac{y \notin M}{\lambda x.M \to_\alpha \lambda y.M[x{:=}y]}$$

$$\frac{M \to_\alpha N}{\lambda x.M \to_\alpha \lambda x.N} \qquad \frac{M \to_\alpha N}{\reflectbox{λ} x.M \to_\alpha \reflectbox{λ} x.N}$$

$$\frac{M \to_\alpha M'}{MN \to_\alpha M'N} \qquad \frac{N \to_\alpha N'}{MN \to_\alpha MN'}$$

The relation $=^{\mathsf{c}}_\alpha$, which we define as $\leftrightarrow^*_\alpha$, i.e. the equivalence closure of $\to_\alpha$, is called *α-conversion*.

The clause dealing with $\reflectbox{$\lambda$}$ is just a compatibility clause, see 3.1.1. Definition of (Barendregt, 1984), since at the time one comes across an $\reflectbox{$\lambda$}$, all the work has already been performed by its matching abstraction. Due to Lemma 3.3, our definitions of α-step and α-conversion coincide with that of (Barendregt, 1984) in the case of λ-terms.

3.2.2 Schroer

Our definition of α-equivalence à la Schroer makes use of an auxiliary stack Z which records the variables chosen thusfar for renaming.

Definition 3.12
α-equivalence à la Schroer, $M =^{\mathsf{s}}_\alpha N$, is defined as

$$\exists Z. M =^Z_\alpha N$$

where $M =^Z_\alpha N$ is defined by

$$\frac{M[x{:=}z] =^Z_\alpha N[y{:=}z] \quad z \notin M, N, Z}{\lambda x.M =^{zZ}_\alpha \lambda y.N}$$

$$\frac{}{x =^Z_\alpha x} \qquad \frac{M =^Z_\alpha N}{\reflectbox{λ} x.M =^Z_\alpha \reflectbox{λ} x.N} \qquad \frac{M =^Z_\alpha M' \quad N =^Z_\alpha N'}{MN =^Z_\alpha M'N'}$$

Again all the work is performed in the clause for abstraction. Compared to α-conversion $=^{\mathsf{c}}_\alpha$ above, the variable chosen for renaming is now much fresher: not only must it be fresh for M, but also for N and for the variables Z chosen thusfar. The clause dealing with $\reflectbox{$\lambda$}$ is just a compatibility clause, as above.

During the proof development, we experimented with all three α-equivalences $=^{\mathsf{c}}_{\alpha}$, $=^{\mathsf{s}}_{\alpha}$, and $=^{\mathsf{k}}_{\alpha}$. For the commutation lemma of $\rightarrow_{\beta}$ and $=_{\alpha}$ (Lemma 4.24), we took $=^{\mathsf{s}}_{\alpha}$ for $=_{\alpha}$. Here we present some lemmas used in the proof of that lemma. If we rename the free occurrences of a variable x (and the occurrences of λx) in α-equivalent terms M and N, the results are still α-equivalent:

Lemma 3.13
$M =^{Z}_{\alpha} N \Rightarrow M[x{:=}z, Y] =^{Z}_{\alpha} N[x{:=}z, Y]$ if z fresh for M, N, Z.

Proof
By induction on $M =^{Z}_{\alpha} N$, using Lemma 3.4 in the abstraction case. □

In renaming expressions $M[x{:=}z]$ it is safe to replace z by a z' just as fresh:

Lemma 3.14
$M[x{:=}z] =^{Z}_{\alpha} N[y{:=}z] \Rightarrow M[x{:=}z'] =^{Z}_{\alpha} N[y{:=}z']$, if z fresh for M, N and z' fresh for M, N, Z.

Proof
Let $z \neq z'$ (the statement trivially holds if $z = z'$), then $z' \notin M[x{:=}z]Y$ and $z' \notin N[y{:=}z]Y$. By the previous lemma we obtain $M[x{:=}z]Y[z{:=}z']Y =^{Z}_{\alpha} N[y{:=}z]Y[z{:=}z']Y$. Conclude by rewriting Lemma 3.5 twice. □

The relation $=^{Z}_{\alpha}$ depends on the 'freshness' of Z and of Z being sufficiently long only:

Lemma 3.15
If $|Z_2| \geq |Z_1|$, Z_2 fresh for M, N, Z_1, and all elements of Z_2 are distinct, then: $M =^{Z_1}_{\alpha} N \Rightarrow M =^{Z_2}_{\alpha} N$.

Proof
By induction over $M =^{Z_1}_{\alpha} N$ and Lemma 3.14. □

Lemma 3.15 solves the difficulty of proving transitivity of $=^{\mathsf{s}}_{\alpha}$ mentioned on page 8.

Lemma 3.16
The relation $=^{\mathsf{s}}_{\alpha}$ is transitive.

Proof
First prove that, for given Z, $=^{Z}_{\alpha}$ is transitive (*). Then, given $M =^{Z_1}_{\alpha} N =^{Z_2}_{\alpha} P$, choose Z_3 of length $\mathsf{max}(|Z_1|, |Z_2|)$ fresh for Z_1, Z_2, M, N, P. Then, by Lemma 3.15 we obtain $M =^{Z_3}_{\alpha} N =^{Z_3}_{\alpha} P$. Finally $M =^{Z_3}_{\alpha} P$ follows from (*). □

3.2.3 Kahrs

Our definition of α-equivalence à la Kahrs reads as follows. It makes use of two auxiliary stacks (both initially empty), to set up the correspondence between the variables in M and N mentioned above.

Definition 3.17

We define $M =_\alpha^{\mathsf{k}} N$, if $\langle\Box\rangle M =_\alpha^{\mathsf{k}} \langle\Box\rangle N$, where for vectors of variables X and Y, $\langle X\rangle M =_\alpha^{\mathsf{k}} \langle Y\rangle N$ is inductively defined as follows. By $|X|$ we denote the length of vector X.

$$
\begin{array}{rcl}
\langle\Box\rangle x & =_\alpha^{\mathsf{k}} & \langle\Box\rangle x \\
\langle Xx\rangle x & =_\alpha^{\mathsf{k}} & \langle Yy\rangle y, \text{ if } |X| = |Y| \\
\langle Xx'\rangle x & =_\alpha^{\mathsf{k}} & \langle Yy'\rangle y, \text{ if } x' \neq x,\ y' \neq y, \text{ and } \langle X\rangle x =_\alpha^{\mathsf{k}} \langle Y\rangle y \\
\langle X\rangle \lambda x.M & =_\alpha^{\mathsf{k}} & \langle Y\rangle \lambda y.N, \text{ if } \langle Xx\rangle M =_\alpha^{\mathsf{k}} \langle Yy\rangle N \\
\langle\Box\rangle \text{ʎ} x.M & =_\alpha^{\mathsf{k}} & \langle\Box\rangle \text{ʎ} x.N, \text{ if } \langle\Box\rangle M =_\alpha^{\mathsf{k}} \langle\Box\rangle N \\
\langle Xx\rangle \text{ʎ} x.M & =_\alpha^{\mathsf{k}} & \langle Yy\rangle \text{ʎ} y.N, \text{ if } \langle X\rangle M =_\alpha^{\mathsf{k}} \langle Y\rangle N \\
\langle Xx'\rangle \text{ʎ} x.M & =_\alpha^{\mathsf{k}} & \langle Yy'\rangle \text{ʎ} y.N, \text{ if } x' \neq x,\ y' \neq y \text{ and } \langle X\rangle \text{ʎ} x.M =_\alpha^{\mathsf{k}} \langle Y\rangle \text{ʎ} y.N \\
\langle X\rangle M_1 M_2 & =_\alpha^{\mathsf{k}} & \langle Y\rangle N_1 N_2, \text{ if } \langle X\rangle M_1 =_\alpha^{\mathsf{k}} \langle Y\rangle N_1 \text{ and } \langle X\rangle M_2 =_\alpha^{\mathsf{k}} \langle Y\rangle N_2
\end{array}
$$

The variable, abstraction and application clauses in the definition above can easily be seen to be corresponding to the clauses in Figure 2. The end-of-scope clauses are analogous to the clauses for variables. Unique reading holds up to α-equivalence:

Lemma 3.18
If $\langle X\rangle M =_\alpha^{\mathsf{k}} \langle Y\rangle N$, then

- M and N have the same skeleton: $[M]_{\mathsf{skel}} = [N]_{\mathsf{skel}}$.
- X and Y have the same length: $|X| = |Y|$.
- M and N have the same set of free variables: $\mathsf{FV}(M, X) = \mathsf{FV}(N, Y)$.

As a consequence we have conservativity of α-equivalence (Lemma 3.22). Note that the third end-of-scope clause of Definition 3.17 expresses that ending the scope of some variable x automatically ends the scope of the variables which were bound later than x. By conservativity of $=_\alpha^{\mathsf{k}}$, this clause can be omitted for scope-balanced terms. For balanced terms we can do with only four clauses:

$$
\begin{array}{rcl}
\langle x\rangle x & =_\alpha^{\mathsf{k}} & \langle y\rangle y \\
\langle Xx\rangle \text{ʎ} x.M & =_\alpha^{\mathsf{k}} & \langle Yy\rangle \text{ʎ} y.N \\
\langle X\rangle \lambda x.M & =_\alpha^{\mathsf{k}} & \langle Y\rangle \lambda y.N, \text{ if } \langle Xx\rangle M =_\alpha^{\mathsf{k}} \langle Yy\rangle N \\
\langle X\rangle M_1 M_2 & =_\alpha^{\mathsf{k}} & \langle Y\rangle N_1 N_2, \text{ if } \langle X\rangle M_1 =_\alpha^{\mathsf{k}} \langle Y\rangle N_1 \text{ and } \langle X\rangle M_2 =_\alpha^{\mathsf{k}} \langle Y\rangle N_2
\end{array}
$$

If $\langle X\rangle M =_\alpha^{\mathsf{k}} \langle Y\rangle N$ holds, then the pair of vectors (X,Y) can be seen as an update on the identity relation (which obviously is a bijection) on variable names, the result of which is a bijection between the 'free' variables of M and N. Indeed, as stated by the following lemma, inserting the same variable on the bottom of both stacks is irrelevant. This lemma is needed, e.g., to show that $=_\alpha^{\mathsf{k}}$ is a congruence (abstraction case).

Lemma 3.19
$\langle X\rangle M =_\alpha^{\mathsf{k}} \langle Y\rangle N$ iff $\langle zX\rangle M =_\alpha^{\mathsf{k}} \langle zY\rangle N$

Proof
The proof is by induction on the derivations. The only interesting cases are the variable and end-of-scope cases, which are similar. So, suppose $M = x$ and $N = y$.

(⇒) — Case $\langle\Box\rangle x =^{\mathsf{k}}_{\alpha} \langle\Box\rangle x$. If $x = z$, then we conclude by application of the second defining clause of $=^{\mathsf{k}}_{\alpha}$. If $x \neq z$, then $\langle z\rangle x =^{\mathsf{k}}_{\alpha} \langle z\rangle x$ follows from the third and the first clause of $=^{\mathsf{k}}_{\alpha}$.

— Case $\langle Xx\rangle x =^{\mathsf{k}}_{\alpha} \langle Yy\rangle y$ with $|X| = |Y|$; so $|zX| = |zY|$ and $\langle zXx\rangle x =^{\mathsf{k}}_{\alpha} \langle zYy\rangle y$ follows from application of the second clause of $=^{\mathsf{k}}_{\alpha}$.

— Case $\langle Xx'\rangle x =^{\mathsf{k}}_{\alpha} \langle Yy'\rangle y$, where $x \neq x'$ and $y \neq y'$; then $\langle zXx'\rangle x =^{\mathsf{k}}_{\alpha} \langle zYy'\rangle y$ follows from application of the third rule of $=^{\mathsf{k}}_{\alpha}$ and the induction hypothesis.

(⇐) By induction on vector X, and inversion of the instances of $\langle zX\rangle x =^{\mathsf{k}}_{\alpha} \langle zY\rangle y$.

— If $X = \Box$, then $Y = \Box$ (by Lemma 3.18), and $\langle\Box\rangle x =^{\mathsf{k}}_{\alpha} \langle\Box\rangle y$ follows, since either $x = z = y$ or $x \neq z \neq y$.

— If $X = X'x'$, then $Y = Y'y'$ (by Lemma 3.18). Inverting $\langle zX'x'\rangle x =^{\mathsf{k}}_{\alpha} \langle zY'y'\rangle y$ gives two possibilities. Either $x = x'$, $y = y'$ and $|zX'| = |zY'|$, then $\langle X'x'\rangle x =^{\mathsf{k}}_{\alpha} \langle Y'y'\rangle y$ follows by application of the second clause; or $x \neq x'$ and $y \neq y'$, then our goal follows by application of the third clause of $=^{\mathsf{k}}_{\alpha}$ and the induction hypothesis. □

3.2.4 Results on α-equivalences

Theorem 3.20
All three notions of α-equivalence are equivalent, $=^{\mathsf{c}}_{\alpha} = =^{\mathsf{s}}_{\alpha} = =^{\mathsf{k}}_{\alpha}$.

Note that to prove that ʎ-terms which are α-equivalent à la Kahrs are α-equivalent according to the other two definitions, one essentially uses the Fresh variable axiom. (It is not needed in the other direction.)

Theorem 3.21
α-equivalence is a congruent equivalence relation.

Proof
Taking the inductive definition of Kahrs, the results are all proven by straightforward inductions on the definition, loading them appropriately with stacks. □

Lemma 3.22
α-equivalence preserves ʎ-terms, scope-balancedness, balancedness, and λ-terms.

Preservation of λ-terms implies that also for the ordinary λ-calculus, the three notions of α-equivalence are equivalent (in the way we have formalised them), yielding as far as we know the first formal such results, e.g. of transitivity and decidability (only assuming decidability of equality of names).

Remark 3.23
Proving the three definitions of α-equivalence to be equivalent served mainly as sanity check for our extension of α-equivalence from λ- to ʎ-calculus. We do not (cl)aim to have covered all definitions of α-equivalence in the literature, see e.g. (Vestergaard & Brotherston, 2001), but, based on the above, we strongly believe that the notion we have captured is the proper one. During proof development, (the generalization of) Kahrs' definition was by far the easiest to work with, because of it

being defined inductively. Note that his definition 'works' directly for the infinitary λ-calculus as well (defined, say, analogously to Chapter 12 of (Terese, 2003)).

4 β

We extend β-reduction to λ-terms, and show it to be confluent without renaming. Confluence of β-reduction up to α-equivalence is obtained as a corollary, by defining suitable projections and liftings of their respective reductions.

4.1 $\lambda\beta$

In Chapter 3 of (Barendregt, 1984), the binary relation $\to_\beta$ on Λ is defined as the compatible closure of the notion of reduction $\beta = \{((\lambda x.M)N, M[x{:=}N]) | M, N \in \Lambda\}$. The substitution $M[x{:=}N]$ in the right-hand side of β is the naive one, i.e. up to α-congruence which is denoted by $\equiv_\alpha$. The naive approach is in turn justified by showing α-congruence to be a congruence for Curry's definition of substitution:

Let M, $N \in \Lambda$. Then $M[x{:=}N]$ is defined inductively as follows (even if the variable convention is not observed).

M	$M[x{:=}N]$
x	N
$y \not\equiv x$	y
M_1M_2	$M_1[x{:=}N]M_2[x{:=}N]$
$\lambda x.M_1$	$\lambda x.M_1$
$\lambda y.M_1$, $y \not\equiv x$	$\lambda z.M_1[y{:=}z][x{:=}N]$ where $z \equiv y$ if $x \notin \mathrm{FV}(M_1)$ or $y \notin \mathrm{FV}(N)$, else z is the first variable in the sequence $v_0, v_1, v_2, \ldots$ not in M_1 or N.

Our notion of substitution on Λ differs from Curry's in several ways.[6]

The first difference is 'under the hood'. Curry's definition is *not* an recursive one (to Coq) because of its final clause. Instead, we base our recursive definition on the skeleton $[M]_{\mathsf{skel}}$ (Definition 3.8).

The second difference is more important and serves to 'make α-congruence explicit'. The point is that the last clause in Curry's definition of substitution is neither perspicuous nor technically convenient. On the one hand, it encodes several cases at once relying on the 'coding trick' that $M[y{:=}y]$ equals M, in case $x \notin \mathrm{FV}(M_1)$ or $y \notin \mathrm{FV}(N)$. On the other hand, renaming of bound variables is not incorporated in a modular way. Our definition addresses both issues by performing renaming first on $\lambda y.M_1$ in case there is the threat of confusion of variables. The definition is recursive (to Coq) if one decrees 'threat of confusion of variables' larger than 'no confusion'.

Definition 4.1

[6] Apart from that we do not assume variables to be ordered, as mentioned above.

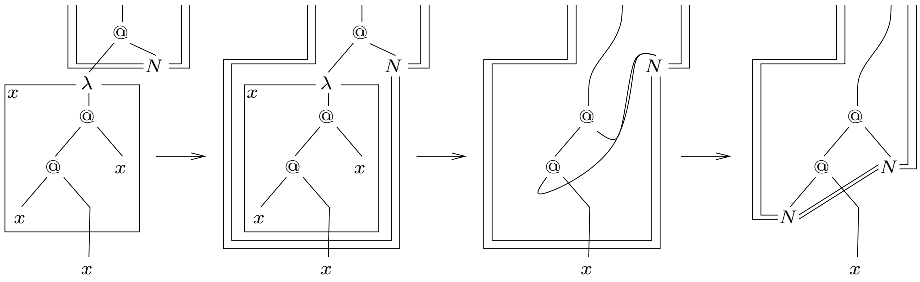

Fig. 3. β-reduction: scope extrusion, rewiring and x-box removal, and replication.

Substitution on λ-terms is defined as above, except for the clauses of λ-abstraction, which are to be replaced by:

$\lambda y.M_1$	$\lambda y.M_1[x{:=}N]$ if $x \neq y$ and $y \notin \mathrm{FV}(N)$
$\lambda y.M_1$	$(\lambda z.M_1[y{:=}z])[x{:=}N]$ otherwise, with z obtained via an α-step from $\lambda y.M_1$ such that $x \neq z$ and $z \notin \mathrm{FV}(N)$

Despite the apparent differences, this definition is seen (proven) to be more liberal than Curry's (it does not need the variables to be linearly ordered).

4.2 ⅄β

We present the definition of β-reduction and the salient points of its proof of confluence. Compared to the ordinary λ-calculus, the β-rule must now take care of an arbitrary number of ⅄s which are 'inbetween' the application and the abstraction. In such cases, the scopes of the ⅄s are *extruded* in a minimal way so as to contain the scope of the abstraction, after which β-reduction proceeds as usual (see Figure 3, where it is irrelevant where scopes are in N). In order to perform all these operations in one go, our notion of substitution as employed by β-reduction has three arguments, of which the second one is as usual.

Definition 4.2
The β-rule is $(\text{⅄}X.\lambda x.M)N \to M[X, x{:=}N, \square]$. The relation $\to_\beta$ is the compatible closure of the β-rule:

$$\frac{}{(\text{⅄}X.\lambda x.M)N \to_\beta M[X, x{:=}N, \square]}$$

$$\frac{M \to_\beta N}{\lambda x.M \to_\beta \lambda x.N} \qquad \frac{M \to_\beta N}{\text{⅄}x.M \to_\beta \text{⅄}x.N}$$

$$\frac{M \to_\beta M'}{MN \to_\beta M'N} \qquad \frac{N \to_\beta N'}{MN \to_\beta MN'}$$

The third argument of substitution, which initially is the empty stack, serves to determine whether an occurrence of x in M matches with the x to be substituted for. In particular, during substitution, this stack is pushed upon when encountering an abstraction, and popped from when meeting an end-of-scope:

Definition 4.3
Substitution $M[X, x{:=}N, Y]$ is defined by:

$$\begin{aligned}
y[X,x{:=}N,Y] &= y, \text{ if } y \in Y && (1)\\
y[X,x{:=}N,Y] &= \reflectbox{\ensuremath{\lambda}} Y.N, \text{ if } y \notin Y,\ x = y && (2)\\
y[X,x{:=}N,Y] &= \reflectbox{\ensuremath{\lambda}} Y.\reflectbox{\ensuremath{\lambda}} X.y, \text{ if } y \notin Y,\ x \neq y && (3)\\
(\lambda y.M)[X,x{:=}N,Y] &= \lambda y.M[X,x{:=}N,yY] && (4)\\
(\reflectbox{\ensuremath{\lambda}} y.M)[X,x{:=}N,Y_1yY_2] &= \reflectbox{\ensuremath{\lambda}} y.M[X,x{:=}N,Y_2], \text{ if } y \notin Y_1 && (5)\\
(\reflectbox{\ensuremath{\lambda}} y.M)[X,x{:=}N,Y] &= \reflectbox{\ensuremath{\lambda}} Y.\reflectbox{\ensuremath{\lambda}} X.M, \text{ if } y \notin Y,\ x = y && (6)\\
(\reflectbox{\ensuremath{\lambda}} y.M)[X,x{:=}N,Y] &= \reflectbox{\ensuremath{\lambda}} Y.\reflectbox{\ensuremath{\lambda}} X.\reflectbox{\ensuremath{\lambda}} y.M, \text{ if } y \notin Y,\ x \neq y && (7)\\
(M_1M_2)[X,x{:=}N,Y] &= M_1[X,x{:=}N,Y]M_2[X,x{:=}N,Y] && (8)
\end{aligned}$$

Capture of free variables in the argument N is avoided by closing all open scopes Y, as expressed in the second clause (2). In case $\reflectbox{\ensuremath{\lambda}} X$ is to be put, the open scopes Y have to be closed first $(3, 6, 7)$. Important clauses are (6) and (7), which explain the end-of-scope. Basically they say that if we have reached an end-of-scope, which matches (6) or jumps (7) the variable x to be subtituted for, then we can just throw the argument N away; this is safe since we know that x does not occur free in M.

Note that the defining clauses for $(\reflectbox{\ensuremath{\lambda}} y.M)[X, x{:=}N, Y]$ are exhaustive, because if $y \in Y$, then $Y = Y_1yY_2$ for some Y_1, Y_2 such that $y \notin Y_1$. The implementation actually uses a nested recursion on Y with an auxiliary stack J (initially empty) recording the opening scopes in Y that are jumped (and closed) by $\reflectbox{\ensuremath{\lambda}} y$. Define $(\reflectbox{\ensuremath{\lambda}} y.M)[X, x{:=}N, Y] = (\reflectbox{\ensuremath{\lambda}} y.M)[X, x{:=}N, Y](\Box)$, where $(\reflectbox{\ensuremath{\lambda}} y.M)[X, x{:=}N, Y](J)$ is defined as follows.

$$\begin{aligned}
(\reflectbox{\ensuremath{\lambda}} y.M)[X,x{:=}N,\Box](J) &= \reflectbox{\ensuremath{\lambda}} J.\reflectbox{\ensuremath{\lambda}} X.M, \text{ if } x = y && (J_1)\\
(\reflectbox{\ensuremath{\lambda}} y.M)[X,x{:=}N,\Box](J) &= \reflectbox{\ensuremath{\lambda}} J.\reflectbox{\ensuremath{\lambda}} X.\reflectbox{\ensuremath{\lambda}} y.M, \text{ if } x \neq y && (J_2)\\
(\reflectbox{\ensuremath{\lambda}} y.M)[X,x{:=}N,zY](J) &= \reflectbox{\ensuremath{\lambda}} y.M[X,x{:=}N,Y], \text{ if } y = z && (J_3)\\
(\reflectbox{\ensuremath{\lambda}} y.M)[X,x{:=}N,zY](J) &= (\reflectbox{\ensuremath{\lambda}} y.M)[X,x{:=}N,Y](Jz), \text{ if } y \neq z && (J_4)
\end{aligned}$$

The clauses for $\reflectbox{\ensuremath{\lambda}} y.M$ in Definition 4.3 are derived from these J-clauses.

Note that jumps *within* the body remain untouched. This can be inferred from (J_3), where J is thrown away. Put differently, we do *not* insert the balancing $\reflectbox{\ensuremath{\lambda}} z$, as in:

$$(\reflectbox{\ensuremath{\lambda}} y.M)[X, x{:=}N, zY] = \reflectbox{\ensuremath{\lambda}} z.(\reflectbox{\ensuremath{\lambda}} y.M)[X, x{:=}N, Y], \text{ if } y \neq z$$

However, the jumped scopes J have to be closed explicitly in case $\reflectbox{\ensuremath{\lambda}} y$ ends the scope of some λy outside the body, that is, above the opening scopes Y.[7] The reason is that when Y is empty, then $\reflectbox{\ensuremath{\lambda}} y$ ends the scope of the substitution variable and we want to put the $\reflectbox{\ensuremath{\lambda}} X$. Now if we do not put the J first, there is the threat that X *explicitly* closes J, which is unintended. For example, reducing

$$P = (\reflectbox{\ensuremath{\lambda}} z.\lambda y.(\lambda x.\reflectbox{\ensuremath{\lambda}} y.M)N)L$$

[7] The body can be thought of as $\lambda Y^{-}.M$, where Y^{-} denotes Y in reversed order.

inside-out gives $⅄z.M$. Reducing outside-in, we pass the intermediate

$$P' = \lambda x.(⅄y.M)[z, y{:=}L, x]N[z, y{:=}L, \Box]$$

Now, it would be wrong to forget that $⅄y$ implicitly closes the λx in front, and put $(⅄y.M)[z, y{:=}L, x] = ⅄z.M$. We would get $P' \to_\beta M$ if $x = z$. Correct is $(⅄y.M)[x, y{:=}L, x] = ⅄z.⅄x.M$, which follows by Definition 4.3 and so $P' \to_\beta ⅄z.M$.

4.2.1 Confluence of ⅄-calculus

We discuss our formalised proof of confluence of $\to_\beta$. Our proof strategy is the usual Tait and Martin-Löf proof (Barendregt, 1984), hence is essentially based on the so-called substitution lemma on page 27 of (Barendregt, 1984):

2.1.16. Substitution Lemma. If $x \not\equiv y$ and $x \notin \mathrm{FV}(L)$, then

$$M[\,x{:=}N][\,y{:=}L] \equiv M[\,y{:=}L]\Big[\,x{:=}N[\,y{:=}L]\Big]$$

which arises when computing the critical pair for the λ-term $(\lambda y.(\lambda x.M)N)L$. Interestingly, the substitution lemma now splits into three lemmas: the *closed* substitution lemmas arise when the scope of y is ended (explicitly or implicitly) in front of the λx; the *open* substitution lemma is the usual one, enriched with scoping information. We will comment on this below. Otherwise, the proof is entirely standard, (inductively) introducing multi-steps, proving that multi-steps have the diamond property and that β-reduction transits multi-steps.

What is interesting to note is that *no* α-conversion is needed. One might say that this is no surprise, since explicitly dealing with end-of-scopes constitutes a renaming mechanism in itself. Still, it *is* in our opinion surprising that the minimal scope-extrusion mechanism works nicely on non-balanced terms (cf. the discussion of confluence of proof net reduction in (Lafont, 1995)).

To find the general statement of our substitution lemmas, we have to simplify expressions $(⅄X.M)[Z, x{:=}N, Y]$, for arbitrary stacks X, Y, Z, ⅄-terms M, N and variable x. In case of scope-balanced terms, we have that $\langle WxY^-\rangle(⅄X.M)$, therefore X and Y are *overlapping*, that is, either:

- X exceeds Y, $X = YxX'$, then $(⅄X.M)[Z, x{:=}N, Y] = ⅄Y.⅄Z.⅄X'.M$; or:
- X is part of Y, $Y = XY'$, then $(⅄X.M)[Z, x{:=}N, Y] = ⅄X.M[Z, x{:=}N, Y']$.

Jump terms demand extra care and we need a different (more general) notion of comparison between opening scopes Y and closing scopes X. Consider, once more, $(⅄X.M)[Z, x{:=}N, Y]$. We distinguish three cases.

1. X closes more scopes than opened by Y, thus X includes the scope of the substitution variable x, which is closed either
   (a) explicitly; or
   (b) implicitly.
2. End-of-scopes X close some (possible all) of the open scopes Y.

To formalise this case distinction, we define *scope subtraction*. The idea is that $Y-X$ either results Y' (item 2) or $-zX'$.[8] In the latter case the substitution variable x is either matched (item 1a) or jumped (item 1b) by z.

For expressing the outcome of $(\curlywedge X.M)[Z, x{:=}N, Y]$, we need to record an auxiliary stack J, consisting of the scopes in Y jumped by the current top of X. This J argument is reset to $\Box$ in case the top of X matches the top of Y and is relevant only if $Y - X = -zX'\#J$; J then is the end-part of Y jumped over by z.

Definition 4.4
Define $Y - X = Y -_{\Box} X$, where $X -_J Y$ is defined by the following clauses.

$$\begin{array}{rcl} Y -_J \Box &=& Y\#J \\ \Box -_J xX &=& -xX\#J \\ yY -_J xX &=& Y -_{\Box} X, \text{ if } x = y \\ yY -_J xX &=& Y -_{Jy} xX, \text{ if } x \neq y \end{array}$$

If X and Y are overlapping, the fourth clause in Definition 4.4 never applies:

Lemma 4.5

$$\begin{array}{rcll} Y_1Y_2 - Y_1 &=& Y_2\#\Box & (Y \geq X, Y = Y_1Y_2, X = Y_1) \\ X_1 - X_1xX_2 &=& -xX_2\#\Box & (Y < X, Y = X_1, X = X_1xX_2) \end{array}$$

For arbitrary stacks X, Y, we can decide whether the outcome of subtracting X from Y is positive or negative:

Lemma 4.6
The result $Y - X$ of scope-subtracting X from Y is either

$$\begin{array}{l} Y_2\#\Box, \text{ and then } Y = Y_1Y_2; \text{ or} \\ -zX_2\#J, \text{ and then } X = X_1zX_2. \end{array}$$

Proof
In order to prove this disjunction, induction (over Y) has to be loaded over J and some extra arguments to express situations half-way. We refer to our Coq formalization for more details. $\Box$

Now we are ready to simplify expressions $(\curlywedge X.M)[Z, x{:=}N, Y]$ based on the case distinction mentioned above.

Lemma 4.7
If $Y - X_1zX_2 = -zX_2\#J$ and $x = z$, then

$$(\curlywedge X_1zX_2.M)[Z, x{:=}N, Y] = \curlywedge X_1.\curlywedge J.\curlywedge Z.\curlywedge X_2.M$$

Lemma 4.8
If $Y - X_1zX_2 = -zX_2\#J$ and $x \neq z$, then

$$(\curlywedge X_1zX_2.M)[Z, x{:=}N, Y] = \curlywedge X_1.\curlywedge J.\curlywedge Z.\curlywedge z.\curlywedge X_2.M$$

[8] The prefix '$-$' indicates that the second argument of subtraction is 'greater than' the first argument.

Lemma 4.9
If $Y_1Y_2 - X = Y_2\#\Box$, then

$$(\reflectbox{λ}X.M)[Z, x{:=}N, Y_1Y_2] = \reflectbox{λ}X.M[Z, x{:=}N, Y_2]$$

Let us now present the substitution lemmas. In general, we want to compute the critical pair(s) from

$$P = (\reflectbox{λ}Z.\lambda y.\lambda Y^-.(\reflectbox{λ}X.\lambda x.\lambda W^-.M)N)L$$

Inside-out reduction, $P \rightarrow^2_\beta P_{\mathsf{in,out}}$, gives:

$$P_{\mathsf{in,out}} = \lambda Y^-.\lambda W^-.M[X, x{:=}N, W][Z, y{:=}L, WY]$$

First β-reducing the outer redex, gives:

$$P_{\mathsf{out}} = \lambda Y^-.\underbrace{(\reflectbox{λ}X.\lambda x.\lambda W^-.M)[Z, y{:=}L, Y]}_{Q}N[Z, y{:=}L, Y]$$

We distinguish three cases for $P_{\mathsf{out}} \rightarrow_\beta P_{\mathsf{out,in}}$:

- $X = X_1zX_2$, $Y - X = -zX_2\#J$ and $y = z$, then, by Lemma 4.7:

  $$Q = \reflectbox{λ}X_1.\reflectbox{λ}J.\reflectbox{λ}Z.\reflectbox{λ}X_2.\lambda x.\lambda W^-.M; \text{ and we get:}$$
  $$P_{\mathsf{out,in}} = \lambda Y^-.\lambda W^-.M[X_1JZX_2, x{:=}N[Z, y{:=}L, Y], W].$$

  Then, $P_{\mathsf{in,out}} = P_{\mathsf{out,in}}$ by Lemma 4.10.
- $X = X_1zX_2$, $Y - X = -zX_2\#J$ and $y \neq z$, then by Lemma 4.8:

  $$Q = \reflectbox{λ}X_1.\reflectbox{λ}J.\reflectbox{λ}Z.\reflectbox{λ}z.\reflectbox{λ}X_2.\lambda x.\lambda W^-.M; \text{ and we get:}$$
  $$P_{\mathsf{out,in}} = \lambda Y^-.\lambda W^-.M[X_1JZzX_2, x{:=}N[Z, y{:=}L, Y], W].$$

  Then, $P_{\mathsf{in,out}} = P_{\mathsf{out,in}}$ by Lemma 4.12.
- $Y = Y_1Y_2$ and $Y - X = Y_2\#\Box$; then, by Lemma 4.9:

  $$Q = \reflectbox{λ}X.\lambda x.\lambda W^-.M[Z, y{:=}L, WxY_2]; \text{ and we get:}$$
  $$P_{\mathsf{out,in}} = \lambda Y^-.\lambda W^-.M[Z, y{:=}L, WxY_2][X, x{:=}N[Z, y{:=}L, Y_1Y_2], W].$$

  Then, $P_{\mathsf{in,out}} = P_{\mathsf{out,in}}$ by Lemma 4.13.

The *closed* substitution lemmas arise when the scope of y *is* ended by some (possibly implicit) $\reflectbox{$\lambda$}y$ in front of the λx, e.g. in $(\lambda y.(\reflectbox{$\lambda$}y.\lambda x.M)N)L$.

Example 4.1
As an illustration, we compute the critical pair arising from $P = (\lambda y.(\reflectbox{λ}y.\lambda x.M)N)L$. If we start with the inner redex, we get:

$$\begin{array}{lll} P & \rightarrow_\beta & (\lambda y.M[y, x{:=}N, \Box])L \\ & \rightarrow_\beta & M[y, x{:=}N, \Box][\Box, y{:=}L, \Box] \end{array}$$

Performing the outer redex first:

$$\begin{array}{lll} P & \rightarrow_\beta & ((\reflectbox{λ}y.\lambda x.M)N)[\Box, y{:=}L, \Box] \\ & = & (\reflectbox{λ}y.\lambda x.M)[\Box, y{:=}L, \Box]N[\Box, y{:=}L, \Box] \\ & = & (\lambda x.M)N[\Box, y{:=}L, \Box] \\ & \rightarrow_\beta & M[\Box, x{:=}N[\Box, y{:=}L, \Box], \Box] \end{array}$$

Note that the substitution for y in M has disappeared from the right-hand side, corresponding to the erasing effect of the $\not\lambda y$ in front of it. Indeed,

$$M[y, x{:=}N, \Box][\Box, y{:=}L, \Box] = M[\Box, x{:=}N[\Box, y{:=}L, \Box], \Box]$$

follows from Lemma 4.10.

If the scope of the substitution variable y is ended *explicitly* by some $\not\lambda y$ in front of the λx, the following lemma springs up.

Lemma 4.10 (*Closed substitution lemma (match)*)
If $Y - X_1 z X_2 = -z X_2 \# J$ and $y = z$, then:

$$\begin{aligned} & M[X_1 z X_2, x{:=}N, W][Z, y{:=}L, WY] \\ = \quad & M[X_1 J Z X_2, x{:=}N[Z, y{:=}L, Y], W] \end{aligned}$$

Remark 4.11
By Lemma 4.5 we obtain:

$$\begin{aligned} & M[X_1 y X_2, x{:=}N, W][Z, y{:=}L, W X_1] \\ = \quad & M[X_1 Z X_2, x{:=}N[Z, y{:=}L, X_1], W] \end{aligned}$$

which is applicable, e.g., when proving the multi-step substitution lemma for scope-balanced terms.

If the scope of the substitution variable y is ended *implicitly* by some $\not\lambda z$ in front of the λx, the following lemma springs up.

Lemma 4.12 (*Closed substitution lemma (jump)*)
If $Y - X_1 z X_2 = -z X_2 \# J$ and $y \neq z$, then:

$$\begin{aligned} & M[X_1 z X_2, x{:=}N, W][Z, y{:=}L, WY] \\ = \quad & M[X_1 J Z z X_2, x{:=}N[Z, y{:=}L, Y], W] \end{aligned}$$

The *open* substitution lemma arises when the scope of y *is not* ended by some end-of-scope in front of the λx. Then we obtain the usual substitution lemma, appropriately enriched with scoping information.

Lemma 4.13 (*Open substitution lemma*)
If $Y_1 Y_2 - X = Y_2 \# \Box$, then

$$\begin{aligned} & M[X, x{:=}N, W][Z, y{:=}L, W Y_1 Y_2] \\ = \quad & M[Z, y{:=}L, W x Y_2][X, x{:=}N[Z, y{:=}L, Y_1 Y_2], W] \end{aligned}$$

Remark 4.14
Because of Lemma 4.5, for scope-balanced terms we can do with:

$$\begin{aligned} & M[Y_1, x{:=}N, W][Z, y{:=}L, W Y_1 Y_2] \\ = \quad & M[Z, y{:=}L, W x Y_2][Y_1, x{:=}N[Z, y{:=}L, Y_1 Y_2], W] \end{aligned}$$

We introduce *multi-steps* that contract all β-redexes simultaneously.

Definition 4.15

Multi-steps $\mathrel{\ominus\!\!\!\to}$ are defined by:

$$\frac{M_1 \mathrel{\ominus\!\!\!\to} N_1 \quad M_2 \mathrel{\ominus\!\!\!\to} N_2}{(\text{\reflectbox{λ}}X.\lambda x.M_1)M_2 \mathrel{\ominus\!\!\!\to} N_1[X, x{:=}N_2, \Box]}$$

$$\frac{}{x \mathrel{\ominus\!\!\!\to} x} \qquad \frac{M_1 \mathrel{\ominus\!\!\!\to} N_1 \quad M_2 \mathrel{\ominus\!\!\!\to} N_2}{M_1M_2 \mathrel{\ominus\!\!\!\to} N_1N_2}$$

$$\frac{M \mathrel{\ominus\!\!\!\to} N}{\lambda x.M \mathrel{\ominus\!\!\!\to} \lambda x.N} \qquad \frac{M \mathrel{\ominus\!\!\!\to} N}{\text{\reflectbox{λ}}x.M \mathrel{\ominus\!\!\!\to} \text{\reflectbox{λ}}x.N}$$

Lemma 4.16 (*Multi-step substitution lemma*)

$$M \mathrel{\ominus\!\!\!\to} M' \wedge N \mathrel{\ominus\!\!\!\to} N' \Rightarrow M[Z, y{:=}N, Y] \mathrel{\ominus\!\!\!\to} M'[Z, y{:=}N', Y]$$

Proof
By induction on $M \mathrel{\ominus\!\!\!\to} M'$. In case $M = (\text{\reflectbox{$\lambda$}}X.\lambda x.M_1)M_2$, the proof obligation is:

$$(\text{\reflectbox{λ}}X.\lambda x.M_1)[Z, y{:=}N, Y]M_2[Z, y{:=}N, Y] \mathrel{\ominus\!\!\!\to} M_1'[X, x{:=}M_2', \Box][Z, y{:=}N', Y]$$

where $M_1 \mathrel{\ominus\!\!\!\to} M_1'$, $M_2 \mathrel{\ominus\!\!\!\to} M_2'$ and $N \mathrel{\ominus\!\!\!\to} N'$. The proof proceeds distinguishing cases in a similar way as on page 21, where we computed the general form of critical pairs and application of the substitution lemmas 4.10, 4.12 and 4.13. $\Box$

Lemma 4.17 (*Multi-step diamond property*)
Multi-steps satisfy the diamond property.

Proof
By induction on the diverging steps. All cases are trivial, except for the case when the starting term is a redex $(\text{\reflectbox{$\lambda$}}X.\lambda x.M)N$, and at least one step is a β-step.

- If both are β-steps, the result follows from the induction hypothesis, using the multi-step substitution lemma (Lemma 4.16).
- If only one of them is a β-step, then the respective results are $M_1[X, x{:=}N_1, \Box]$ and $(\text{\reflectbox{$\lambda$}}X.\lambda x.M_2)N_2$, for $M \mathrel{\ominus\!\!\!\to} M_i$ and $N \mathrel{\ominus\!\!\!\to} N_i$. By the induction hypothesis $M_i \mathrel{\ominus\!\!\!\to} M'$ and $N_i \mathrel{\ominus\!\!\!\to} N'$ for some M' and N'. Hence the result follows since

  $$M_1[X, x{:=}N_1, \Box] \mathrel{\ominus\!\!\!\to} M_2[X, x{:=}N_2, \Box]$$

  by the multi-step substitution lemma (Lemma 4.16), and

  $$(\text{\reflectbox{λ}}X.\lambda x.M_2)N_2 \mathrel{\ominus\!\!\!\to} M_2[X, x{:=}N_2, \Box]$$

  by definition of $\mathrel{\ominus\!\!\!\to}$. $\Box$

Lemma 4.18
$\to_\beta$ transits $\mathrel{\ominus\!\!\!\to}$.

Proof
By induction on the definition of $\to_\beta$ and $\mathrel{\ominus\!\!\!\to}$, respectively. The $\mathrel{\ominus\!\!\!\to}\subseteq\to_\beta^*$ part follows from congruence of $\to_\beta^*$: if $M \to_\beta^* N$, then $CM \to_\beta^* CN$, for unary contexts C. $\Box$

Theorem 4.19 (*Confluence of* $\to_\beta$)

$\to_\beta$ is confluent on Λ.

Proof
From Lemmas 2.5, 4.18 and 4.17. □

4.3 α and β

We prove that α and β commute on scope-balanced terms. We are confident that commutation of α and β holds for *all* $\reflectbox{$\lambda$}$-terms, but leave this for future work. In particular, this would require more general formulations (using scope subtraction) of the renaming/substitution lemmas (Lemmas 4.20 and 4.21).

Lemma 4.20 (*Open substitution/renaming lemma*)

$$\begin{aligned} & M[X_1, y{:=}N, X_0][x{:=}z, X_0X_1X_2] \\ = \quad & M[x{:=}z, X_0yX_2][X_1, y{:=}N[x{:=}z, X_1X_2], X_0] \end{aligned}$$

if z fresh for y, M, X_0.

Lemma 4.21 (*Closed substitution/renaming lemma*)

$$\begin{aligned} & M[X_1xX_2, y{:=}N, X_0][x{:=}z, X_0X_1] \\ = \quad & M[X_1zX_2, y{:=}N[x{:=}z, X_1], X_0] \end{aligned}$$

Note that, for balanced terms, as we have that $M[x{:=}y, X] = M[y, x{:=}y, X]$, Lemmas 4.20 and 4.21 follow from Lemmas 4.10 and 4.13, given that $\langle WxX^-\rangle M$ and y fresh for M, X.

It is safe to rename in β-reductions:

Lemma 4.22
If $z \notin M$, $\langle WxY^-\rangle M$, then $M \to_\beta N$ implies $M[x{:=}z, Y] \to_\beta N[x{:=}z, Y]$.

Proof
Consider the case $(\reflectbox{$\lambda$}X.\lambda x.M_1)M_2 \to_\beta M_1[X, x{:=}M_2, \Box]$. Because of the assumption $\langle WxY^-\rangle(\reflectbox{λ}X.\lambda y.M_1)$, either X exceeds Y, and then the closed substitution/renaming lemma applies, or X is part of Y, and then the open substitution/renaming lemma applies. □

If $\lambda x.M$ and $\lambda y.M'$ are α-equivalent (so also the outer (x- resp. y-)boxes have the same shape) and N and N' are α-equivalent, then the β-contractum of $(\reflectbox{$\lambda$}X.\lambda x.M)N$ is α-equivalent to the β-contractum of $(\reflectbox{$\lambda$}X.\lambda y.M')N'$:

Lemma 4.23

$$\begin{aligned} & M[x{:=}z, Y] =_\alpha^{Z_1} M'[y{:=}z, Y] \wedge N =_\alpha^{Z_2} N' \\ \Rightarrow \quad & \exists Z_3. M[X, x{:=}N, Y] =_\alpha^{Z_3} M'[X, y{:=}N', Y] \end{aligned}$$

if Z_1 fresh for X, Y, Z_2, M, M', N, N', and z fresh for M, M'.

On scope-balanced terms, β-reduction and α-equivalence commute (Schema E in Figure 4). Here, we take $=^{\mathsf{s}}_\alpha$ for $=_\alpha$.

Lemma 4.24
If $\langle X\rangle M$, $M \to_\beta N$, and $M =^Z_\alpha M'$, then there exists a term N' and a stack Z' such that $M' \to_\beta N'$ and $N =^{Z'}_\alpha N'$.

Proof
By induction on the skeleton of M. We show some interesting cases. Recall that renaming doesnot alter the skeleton of a term and that scope-balancedness is closed under $=_\alpha$.

- Case $\lambda x.M_0 \to_\beta \lambda x.N_0$. By inverting $\lambda x.M_0 =^Z_\alpha M'$, we obtain $M' = \lambda y.M'_0$ and $Z = zZ_0$ such that $M_0[x{:=}z] =^{Z_0}_\alpha M'_0[y{:=}z]$ for $z \notin M_0, M'_0, Z_0$. By Lemma 4.22, we have $M_0[x{:=}z] \to_\beta N_0[x{:=}z]$. Then, by the induction hypothesis, there exist P and Z'_0 such that $M'_0[y{:=}z] \to_\beta P$ and $N_0[x{:=}z] =^{Z'_0}_\alpha P$. We are able prove that, for some N'_0, $P = N'_0[y{:=}z]$ and $M'_0 \to_\beta N'_0$. Choose z' fresh for N_0, N'_0, Z'_0 (Axiom 2 guarantees the existence of z'). From Lemma 3.14, we obtain $N_0[x{:=}z'] =^{Z'_0}_\alpha N'_0[x{:=}z']$ ($\to_\beta$ doesn't introduce new names, therefore $z \notin N_0, N'_0$ follows from $z \notin M_0, M'_0$). Then, $N' = \lambda y.N'_0$ and $Z' = z'Z'_0$ witness the goal $\exists N'.\exists Z'.\lambda y.M'_0 \to_\beta N' \wedge \lambda x.N_0 =^{Z'}_\alpha N'$.
- Case $M_1M_2 \to_\beta N_1M_2$. So $M_1 \to_\beta N_1$, $M' = M'_1M'_2$, $M_1 =^{Z_1}_\alpha M'_1$, and $M_2 =^{Z_1}_\alpha M'_2$. By the induction hypothesis, there exist N'_1 and Z_2 such that $M'_1 \to_\beta N'_1$ and $N_1 =^{Z_2}_\alpha N'_1$. By Lemma 3.15, we obtain $\exists Z'.M'_1M'_2 =^{Z'}_\alpha N'_1M'_2$.
- Case $(\lambda\!\!\!\!\lambda X.\lambda x.M_1)M_2 \to_\beta M_1[X, x{:=}M_2, \Box]$. Derive $M' = (\lambda\!\!\!\!\lambda X.\lambda y.M'_1)M'_2$, $M_1[x{:=}z] =^Z_\alpha M'_1[y{:=}z]$, and $M_2 =^{zZ}_\alpha M'_2$ with $z \notin M_1, M_2, Z$. Choose Z_0 fresh for $M_1, M_2, M'_1, M'_2, X, Z$ and such that $|Z_0| = |Z|$ (apply Axiom 2 $|Z_0|$ times). Then Z_0 is provably fresh for $M_1[x{:=}z]$ and $M'_1[y{:=}z]$ as well. By Lemma 3.15 we obtain $M_1[x{:=}z] =^{Z_0}_\alpha M'_1[y{:=}z]$ Take $N' = M'_1[X, y{:=}M'_2, \Box]$ as witness for the existential statement we are proving. Finally, by Lemma 4.23, there exists Z' such that $M_1[X, x{:=}M_2, \Box] =^{Z'}_\alpha M'_1[X, y{:=}M'_2, \Box]$. $\Box$

4.4 Confluence of λ-calculus

As a corollary we obtain confluence of the ordinary λ-calculus (see Figure 4). The exposition of the proof proceeds in a top-down fashion, forward referring to lifting and projection lemmas. We use $\to_{\lambda\beta}$ and $\to_{\lambda\!\!\!\lambda\beta}$ to distinguish β-reduction in the λ-calculus from β-reduction in the $\lambda\!\!\!\!\lambda$-calculus, respectively.

Theorem 4.25
$\to_\beta/{=_\alpha}$ is confluent on Λ.

Proof
1. Consider two diverging $\lambda\beta$-reductions $M \to^*_{\lambda\beta} N$ and $M \to^*_{\lambda\beta} P$.
2. Lift these stepwise to diverging $\lambda\!\!\!\lambda\beta$-reductions $M \to^*_{\lambda\!\!\!\lambda\beta} N'$ and $M \to^*_{\lambda\!\!\!\lambda\beta} P'$ (Lemma 4.36). (Note that M being a λ-term, it is a scope-balanced $\lambda\!\!\!\!\lambda$-term .)

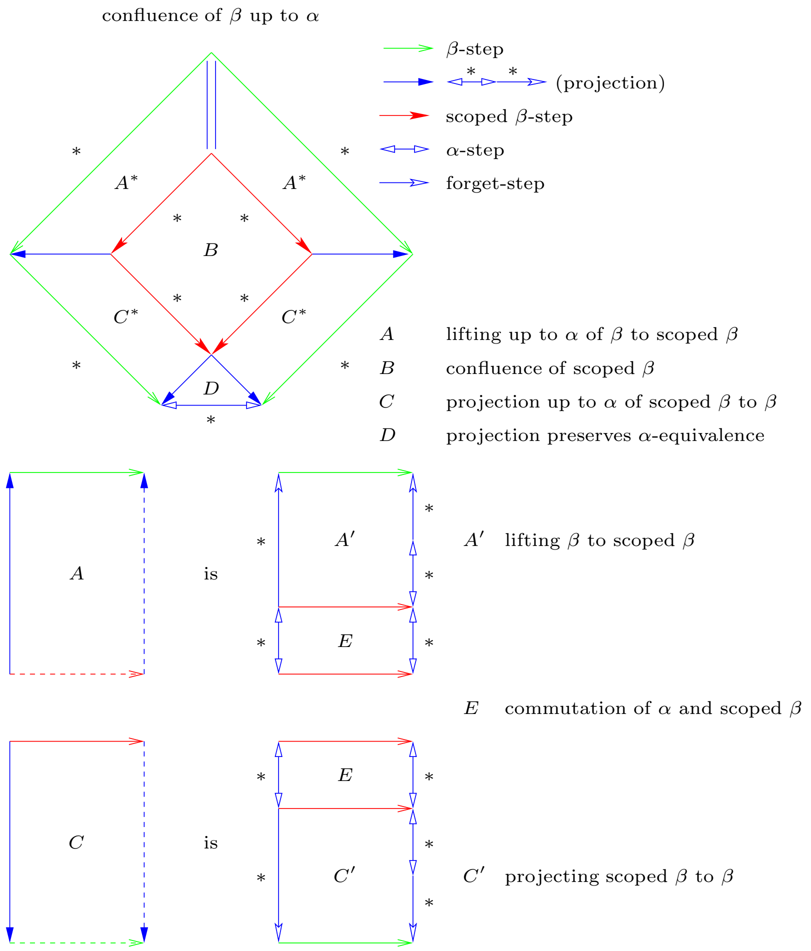

Fig. 4. Confluence of $\not\lambda$-calculus implies confluence of λ-calculus.

3. By confluence of $\not\lambda\beta$-reduction, we can find some $\not\lambda$-term Q' such that $N' \to^*_{\not\lambda\beta} Q'$, $P' \to^*_{\not\lambda\beta} Q'$ (Theorem 4.19).
4. Projecting $N' \to^*_{\not\lambda\beta} Q'$ and $P' \to^*_{\not\lambda\beta} Q'$ back to $\lambda\beta$-reduction yields $N \to^*_{\lambda\beta} Q_1$ and $P \to^*_{\lambda\beta} Q_2$ (Lemma 4.37), for some α-equivalent λ-terms Q_1 and Q_2 (Remark 4.30), establishing the desired confluence of $\lambda\beta$ up to α-equivalence. □

Remark 4.26
As far as we know the only formalised proof of confluence of β-reduction modulo α, in our setting, i.e. with a single variable space is (Vestergaard & Brotherston, 2001). However, their proof technique is entirely different, uniquely renaming all variables, before performing β-steps, whereas our schema, which works via the $\not\lambda$-calculus, only performs the necessary updates (in the sense of (David & Guillaume, 2001)).

4.4.1 Lifting and Projection of β-reduction

Projection of $\text{\reflectbox{$\lambda$}}$-terms to λ-terms is the composition of first performing an α-equivalence step followed by a so-called ω-step removing all $\text{\reflectbox{$\lambda$}}$s in one go.[9] For instance, no ω-step is possible from $\lambda x.\text{\reflectbox{λ}}x.x$ since removing $\text{\reflectbox{$\lambda$}}x$ would turn the free variable x into a bound variable in $\lambda x.x$. Obviously, uniquely renaming all variables would guarantee that an ω-step can be performed. However, we rename only if necessary.

Definition 4.27
We define $M \mathrel{-\!\!\bullet\!\!\rightarrow}_{\omega} N$, if $\langle\Box\rangle M \mathrel{-\!\!\bullet\!\!\rightarrow}_{\omega} N$, where $\langle X\rangle M \mathrel{-\!\!\bullet\!\!\rightarrow}_{\omega} N$ is defined by the following clauses.

$$\frac{}{\langle X\rangle x \mathrel{-\!\!\bullet\!\!\rightarrow}_{\omega} x} \qquad \frac{\langle Xx\rangle M \mathrel{-\!\!\bullet\!\!\rightarrow}_{\omega} M'}{\langle X\rangle \lambda x.M \mathrel{-\!\!\bullet\!\!\rightarrow}_{\omega} \lambda x.M'}$$

$$\frac{\langle X\rangle M_1 \mathrel{-\!\!\bullet\!\!\rightarrow}_{\omega} N_1 \quad \langle X\rangle M_2 \mathrel{-\!\!\bullet\!\!\rightarrow}_{\omega} N_2}{\langle X\rangle M_1M_2 \mathrel{-\!\!\bullet\!\!\rightarrow}_{\omega} N_1N_2} \qquad \frac{\langle X\rangle M \mathrel{-\!\!\bullet\!\!\rightarrow}_{\omega} M' \quad x \notin \mathsf{FV}(M)}{\langle Xx\rangle \text{\reflectbox{λ}}x.M \mathrel{-\!\!\bullet\!\!\rightarrow}_{\omega} M'}$$

Thus, ω-steps are maximal, in the sense that one ω-step removes all $\text{\reflectbox{$\lambda$}}$s in one go. We write $\langle X\rangle M =^{\mathsf{k}}_{\alpha} \langle Y\rangle N \mathrel{-\!\!\bullet\!\!\rightarrow}_{\omega} P$ to abbreviate $\langle X\rangle M =^{\mathsf{k}}_{\alpha} \langle Y\rangle N \wedge \langle Y\rangle N \mathrel{-\!\!\bullet\!\!\rightarrow}_{\omega} P$, and, conversely, we write $\langle X\rangle M \mathrel{-\!\!\bullet\!\!\rightarrow}_{\omega} N =^{\mathsf{k}}_{\alpha} \langle Y\rangle P$ to abbreviate $\langle X\rangle M \mathrel{-\!\!\bullet\!\!\rightarrow}_{\omega} N \wedge \langle X\rangle N =^{\mathsf{k}}_{\alpha} \langle Y\rangle P$. Note that the source of the $\mathrel{-\!\!\bullet\!\!\rightarrow}_{\omega}$ is, by definition, forced to be scope-balanced:

$$\langle X\rangle M \mathrel{-\!\!\bullet\!\!\rightarrow}_{\omega} N \Rightarrow \langle X\rangle M$$

Also note that $\mathrel{-\!\!\bullet\!\!\rightarrow}_{\omega}$ doesn't alter the set of free variables:

$$\langle YX^{-}\rangle M \mathrel{-\!\!\bullet\!\!\rightarrow}_{\omega} M' \Rightarrow \mathsf{FV}(M, X) = \mathsf{FV}(M', X)$$

Example 4.2
If we first rename the bound xs in $\lambda x.\text{\reflectbox{λ}}x.x$, by some $x' \neq x$, then it is safe to forget the $\text{\reflectbox{$\lambda$}}x'$: $\langle x\rangle \lambda x.\text{\reflectbox{λ}}x.x =^{\mathsf{k}}_{\alpha} \langle x\rangle \lambda x'.\text{\reflectbox{λ}}x'.x \mathrel{-\!\!\bullet\!\!\rightarrow}_{\omega} \lambda x'.x$.

Example 4.3
It is incorrect to forget end-of-scopes in the jump calculus, as witnessed by the $\text{\reflectbox{$\lambda$}}$-term $\lambda x.\lambda y.\text{\reflectbox{λ}}x.y$. The variable y is free in this term since $\text{\reflectbox{$\lambda$}}x$ implicitly closes the scope of y. However, forgetting this end-of-scope would yield the λ-term $\lambda x.\lambda y.y$ where y is bound. The easiest way to proceed seems to be to insert as many $\text{\reflectbox{$\lambda$}}$s as are needed to make the $\text{\reflectbox{$\lambda$}}$-term scope-balanced:

$$\begin{array}{rcl}
\mathsf{scb}(x, X) & = & x \\
\mathsf{scb}(\lambda x.M, X) & = & \lambda x.\mathsf{scb}(M, xX) \\
\mathsf{scb}(\text{\reflectbox{λ}}x.M, \Box) & = & \mathsf{scb}(M, \Box) \\
\mathsf{scb}(\text{\reflectbox{λ}}x.M, yX) & = & \text{\reflectbox{λ}}x.\mathsf{scb}(M, X) \text{ if } x = y \\
\mathsf{scb}(\text{\reflectbox{λ}}x.M, yX) & = & \text{\reflectbox{λ}}y.\mathsf{scb}(\text{\reflectbox{λ}}x.M, X) \text{ if } x \neq y \\
\mathsf{scb}(M_1M_2, X) & = & \mathsf{scb}(M_1, X)\mathsf{scb}(M_2, X)
\end{array}$$

[9] ω could be decomposed itself by first pushing $\text{\reflectbox{$\lambda$}}$s to the variables, i.e. performing *maximal* scope extrusion before omitting $\text{\reflectbox{$\lambda$}}$s.

Check that $\langle X\rangle\mathsf{scb}(M,X)$ for all terms M and stacks X. Applied to the example, we first obtain $\mathsf{scb}(\lambda x.\lambda y.\lambdabar x.y,\Box) = \lambda x.\lambda y.\lambdabar y.\lambdabar x.y$. Now we see that in order to omit the $\lambdabar y$, we have to rename it first, say to z yielding $\lambda x.\lambda z.\lambdabar z.\lambdabar x.y$. Forgetting end-of-scopes now yields the (correct) λ-term $\lambda x.\lambda z.y$.

Remark 4.28
In $\lambda\beta$-reduction renamings are performed, as soon as there is a confusion threat. However, such a threat may turn out to be innocuous, as in:

$$(\lambda y.\lambda x.(\lambda z.I)yx)x \to \lambda x'.(\lambda z.I)xx' \to \lambda x'.Ix'$$

The renaming is caused by the substitution for the variable x which is erased later anyway. On the other hand, no renaming takes place during $\lambdabar\beta$-reduction:

$$(\lambda y.\lambda x.(\lambda z.I)yx)x \to \lambda x.(\lambda z.I)(\lambdabar x.x)x \to \lambda x.Ix$$

Observe that despite the final term of this $\lambdabar\beta$-reduction being an ordinary λ-term, α-conversion *is* needed to project it (see Lemma 4.37).

The relations $=_{\alpha}^{\mathsf{k}}$ and $\mathrel{\bullet\!\!\!\rightarrow}_{\omega}$ commute:

Lemma 4.29
$N \;{}_{\omega}\!\!\leftarrow\!\!\bullet\; \langle X\rangle M =_{\alpha}^{\mathsf{k}} \langle X'\rangle M' \mathrel{\bullet\!\!\!\rightarrow}_{\omega} N'$ implies $\langle X\rangle N =_{\alpha}^{\mathsf{k}} \langle X'\rangle N'$.

Proof
We show the case $\langle Xx\rangle\lambdabar x.M =_{\alpha}^{\mathsf{k}} \langle X'y\rangle\lambdabar y.M'$. Then, $\langle X\rangle M \mathrel{\bullet\!\!\!\rightarrow}_{\omega} N$, $x \notin \mathsf{FV}(M)$, $\langle X'\rangle M' \mathrel{\bullet\!\!\!\rightarrow}_{\omega} N'$, and $y \notin \mathsf{FV}(M')$. Inversion gives $\langle X\rangle M =_{\alpha}^{\mathsf{k}} \langle X'\rangle M'$. By the induction hypothesis we have $\langle X\rangle N =_{\alpha}^{\mathsf{k}} \langle X'\rangle N'$, which, by the following lemma, implies the goal, $\langle Xx\rangle N =_{\alpha}^{\mathsf{k}} \langle X'y\rangle N'$, because N, N' are free of $\lambdabar$s. $\Box$

Remark 4.30
Schema D in Figure 4, stating that $Q' =_{\alpha} Q'_1 \mathrel{\bullet\!\!\!\rightarrow}_{\omega} Q_1$ and $Q' =_{\alpha} Q'_2 \mathrel{\bullet\!\!\!\rightarrow}_{\omega} Q_2$ imply $Q_1 =_{\alpha} Q_2$, now easily follows: first show that $Q'_1 =_{\alpha} Q'_2$ (by symmetry and transitivity of $=_{\alpha}$) and then apply the previous lemma.

Lemma 4.31
If M is a λ-term, that is, M contains no $\lambdabar$s, and $x \notin \mathsf{FV}(M)$, then

$$\langle Xx\rangle M =_{\alpha}^{\mathsf{k}} \langle Yy\rangle N \text{ implies } \langle X\rangle M =_{\alpha}^{\mathsf{k}} \langle Y\rangle N \text{ and } y \notin \mathsf{FV}(N)$$

Conversely, if $x \notin \mathsf{FV}(M)$ and $y \notin \mathsf{FV}(N)$, then

$$\langle X\rangle M =_{\alpha}^{\mathsf{k}} \langle Y\rangle N \text{ implies } \langle Xx\rangle M =_{\alpha}^{\mathsf{k}} \langle Yy\rangle N$$

Given a sequence of $\mathrel{\bullet\!\!\!\rightarrow}_{\omega}$- and α-steps, the $\mathrel{\bullet\!\!\!\rightarrow}_{\omega}$-steps can always be postponed until the α-steps are performed:

Lemma 4.32
$\langle X\rangle M \mathrel{\bullet\!\!\!\rightarrow}_{\omega} P =_{\alpha}^{\mathsf{k}} \langle Y\rangle N \Rightarrow \exists Q.\langle X\rangle M =_{\alpha}^{\mathsf{k}} \langle Y\rangle Q \mathrel{\bullet\!\!\!\rightarrow}_{\omega} N$.

Proof

By induction on the definition of $-\!\!\bullet\!\!\to_\omega$. Consider case $\langle Xx\rangle ƛx.M \ -\!\!\bullet\!\!\to_\omega \ P =^{\mathsf{k}}_\alpha \langle Yy\rangle N$. Then $x \notin \mathsf{FV}(M)$ and $\langle X\rangle M -\!\!\bullet\!\!\to_\omega P$. Because $-\!\!\bullet\!\!\to_\omega$ doesn't change the set of free variables, we have that $x \notin \mathsf{FV}(P)$. By Lemma 4.31, we get $\langle X\rangle P =^{\mathsf{k}}_\alpha \langle Y\rangle N$ and $y \notin \mathsf{FV}(N)$. By the induction hypothesis, we have Q such that $\langle M\rangle X =^{\mathsf{k}}_\alpha \langle Y\rangle Q -\!\!\bullet\!\!\to_\omega N$. $y \notin \mathsf{FV}(Q)$ follows and we obtain $\langle Xx\rangle ƛx.M =^{\mathsf{k}}_\alpha \langle Yy\rangle ƛy.Q -\!\!\bullet\!\!\to_\omega N$.
$\square$

Both projection and lifting of reductions are performed stepwise. That is, a single $\lambda\beta$-step lifts to a single $ƛ\beta$-step and vice versa (not to reduction sequences, as in calculi with explicit substitutions). Lifting of $\to_{\lambda\beta}$ to $\to_{ƛ\beta}$ (Schema A' in Figure 4) is stated as follows.

Lemma 4.33
If $M \to_{\lambda\beta} N$ and $\langle X\rangle M' -\!\!\bullet\!\!\to_\omega \mathsf{emb}(M)$, then there are N_1, N_2 such that:

$$\langle X\rangle N_1 =^{\mathsf{k}}_\alpha \langle X\rangle N_2 -\!\!\bullet\!\!\to_\omega \mathsf{emb}(N) \text{ and } M' \to_{ƛ\beta} N_1$$

where emb is the canonical lifting of λ-terms to $ƛ$-terms.

Proof
As an illustration, consider the case $M = (\lambda x.M_1)M_2$, and $M' = L_1L_2$. We have $\langle X\rangle L_1 -\!\!\bullet\!\!\to_\omega \lambda x.\mathsf{emb}(M_1)$ and $\langle X\rangle L_2 -\!\!\bullet\!\!\to_\omega \mathsf{emb}(M_2)$. By inversion, we obtain $L_1 = ƛX_1.\lambda x.L_1'$, $X^- = X_1X_2$, $X_1 \cap \mathsf{FV}(\lambda x.L_1') = \emptyset$ and $\langle X_2^- x\rangle L_1' -\!\!\bullet\!\!\to_\omega \mathsf{emb}(M_1)$. The proof obligation is

$$\exists N_1, N_2.\langle X\rangle N_1 =^{\mathsf{k}}_\alpha \langle X\rangle N_2 -\!\!\bullet\!\!\to_\omega \mathsf{emb}(M_1[x{:=}M_2]) \wedge (ƛX_1.\lambda x.L_1')L_2 \to_{ƛ\beta} N_1$$

Take $N_1 = L_1'[X_1, x{:=}L_2, \square]$. The following lemma (Lemma 4.34) guarantees the existence of an $ƛ$-term P such that

$$\langle X\rangle L_1'[X_1, x{:=}L_2, \square] =^{\mathsf{k}}_\alpha \langle X\rangle P[X_1, x{:=}L_2, \square] -\!\!\bullet\!\!\to_\omega \mathsf{emb}(M_1[x{:=}M_2])$$

The witnessing $N_2 = P[X_1, x{:=}L_2, \square]$ solves our goal. $\square$

The following lemma states projection of $ƛ$-substitution to λ-substitution.

Lemma 4.34

$$\begin{array}{ll} \multicolumn{2}{l}{\langle Zx\rangle M_1 -\!\!\bullet\!\!\to_\omega \mathsf{emb}(M)} \\ \wedge & \langle ZX\rangle N' -\!\!\bullet\!\!\to_\omega \mathsf{emb}(N) \\ \wedge & X \cap \mathsf{FV}(\lambda x.M_1) = \emptyset \\ \Rightarrow & \exists P : \Lambda.\langle ZX\rangle M_1[X, x{:=}N', \square] =^{\mathsf{k}}_\alpha \langle ZX\rangle P[X, x{:=}N', \square] -\!\!\bullet\!\!\to_\omega \mathsf{emb}(M[x{:=}N]) \end{array}$$

Proof
The difficult part was to find the right induction loading:

$$\begin{array}{ll} \multicolumn{2}{l}{\langle ZxY_1\rangle M_1 =^{\mathsf{k}}_\alpha \langle ZxY_2\rangle M_2 -\!\!\bullet\!\!\to_\omega \mathsf{emb}(M)} \\ \wedge & \langle ZX\rangle N' -\!\!\bullet\!\!\to_\omega \mathsf{emb}(N) \\ \wedge & X \cap \mathsf{FV}(M_1, Y_1x) = \emptyset \\ \wedge & Y_2 \cap (\{x\} \cup \mathsf{FV}(N')) = \emptyset \end{array}$$

$$\Rightarrow \quad \exists P : \Lambda . \langle ZXY_1 \rangle M_1[X, x{:=}N', Y_1] =^{\mathsf{k}}_{\alpha} \langle ZXY_2 \rangle P[X, x{:=}N', Y_2] \\ \qquad\qquad\qquad\qquad\qquad \multimap\!\!\!\rightarrow_{\omega} \mathsf{emb}(M[x{:=}N])$$

Once appropriately loaded, the proof is a straightforward induction over M_1, the only interesting lemma used being Lemma 4.32. $\Box$

Projecting $\to_{\not\lambda\beta}$ to $\to_{\lambda\beta}$ (Schema C' in Figure 4) is stated as follows.

Lemma 4.35
If $M \to_{\not\lambda\beta} N$ and $\langle X \rangle M \multimap\!\!\!\rightarrow_{\omega} \mathsf{emb}(M')$, then there are N_1, N_2 such that:

$$\langle X \rangle N =^{\mathsf{k}}_{\alpha} \langle X \rangle N_1 \multimap\!\!\!\rightarrow_{\omega} \mathsf{emb}(N_2) \text{ and } M' \to_{\lambda\beta} N_2$$

Proof
The β-rule case calls Lemma 4.34 again. $\Box$

Lifting of $\lambda\beta$-reduction sequences to $\not\lambda\beta$-reduction sequences (Schema A^* in Figure 4) is stated as follows.

Lemma 4.36

$$M_1 \to^{*}_{\lambda\beta} M_2 \wedge P_1 =^{\mathsf{k}}_{\alpha} Q_1 \multimap\!\!\!\rightarrow_{\omega} \mathsf{emb}(M_1) \\ \Rightarrow \quad \exists P_2, Q_2 : \Lambda . P_2 =^{\mathsf{k}}_{\alpha} Q_2 \multimap\!\!\!\rightarrow_{\omega} \mathsf{emb}(M_2) \wedge P_1 \to^{*}_{\not\lambda\beta} P_2$$

Proof
By Lemmas 4.33 and 4.24. $\Box$

Lemma 4.37
Schema C^* in Figure 4, projection of $\not\lambda\beta$-reduction sequences to $\lambda\beta$-reduction sequences is stated by the following lemma.

$$P_1 \to^{*}_{\not\lambda\beta} P_2 \wedge P_1 =^{\mathsf{k}}_{\alpha} Q_1 \multimap\!\!\!\rightarrow_{\omega} \mathsf{emb}(M_1) \\ \Rightarrow \quad \exists M_2 : \Lambda . \exists Q_2 : \Lambda . P_2 =^{\mathsf{k}}_{\alpha} Q_2 \multimap\!\!\!\rightarrow_{\omega} \mathsf{emb}(M_2) \wedge M_1 \to^{*}_{\lambda\beta} M_2$$

Proof
By Lemmas 4.35 and 4.24. $\Box$

5 Applications

We think that the $\not\lambda$-calculus provides an intuitive understanding of scoping in the λ-calculus. We claim it can provide solutions to problems which are known to be hard for the λ-calculus.

Expressing free variable conditions In the λ-calculus one often has use for free variable conditions. Not only are these necessary to *express* e.g. the η-rule:

$\lambda x.Mx \to M$, if $x \notin \mathsf{FV}(M)$, but knowing that x does not occur in the free variables of M would also *speed up* reduction of the β-redex $(\lambda x.M)N$; in that case one may simply erase N. Rather than reifying the negative concept of *a variable not occurring free in a subterm*, cf. e.g. (Gordon & Melham, 1996), our $\reflectbox{$\lambda$}$-operator makes the positive concept of *the ending of the scope of a variable* explicit. Using it, the free-variable condition of the η-rule can be expressed in the object language as:

$$\lambda x.(\reflectbox{λ} x.M)x \to M,$$

and the β-redex becomes $(\lambda x.\reflectbox{$\lambda$} x.M)N$, which indeed executes more efficiently. In (David & Guillaume, 2001) some statistical evidence is presented that this is a frequently occurring situation, i.e. that it is worthwhile to retain scoping information when evaluating ordinary λ-terms.

In the next section we present some further evidence to the usefulness of the $\reflectbox{$\lambda$}$-calculus.

6 Conclusion and Discussion

The reification of scopes supplies a fundamental understanding of scoping mechanisms of *named*[10] terms. Our results can be summarised as follows.

- Confluence of the $\reflectbox{$\lambda$}$-calculus *without* α-conversion.
- Scope information is retained, possibly *speeding up* β-reduction.
- Free variable conditions are expressable in the object language.
- Unintended capture of free variables in the substituens by binders in the substitution body is avoided, not by renaming the binders, but by prefixing the substituens by $\reflectbox{$\lambda$}$s ending the scope of those binders. Because the $\reflectbox{$\lambda$}$s are not pushed to the variables (*maximal* scope extrusion, that is), the transformation of arguments during substitution is avoided; thus, arguments can still be subject to sharing.
- α-conversion is a decidable congruent equivalence.

Restricting to a single name, the $\reflectbox{$\lambda$}$-calculus corresponds to the λ-calculus formalised using a generalisation of de Bruijn indices, with the *shift substitution* $[\uparrow]$ (cf. $\reflectbox{$\lambda$}$) as explicit term constructor (see the paragraph on related work in Section 1).

By lifting β-reduction of the λ-calculus to β-reduction of the $\reflectbox{$\lambda$}$-calculus, and projecting back, we can analyse more precisely renamings performed 'on the fly' by β-reduction in the λ-calculus. Confluence (up to α) of the λ-calculus is obtained as a derived result.

We conclude this paper by discussing two potential applications of the $\reflectbox{$\lambda$}$-calculus we are currently investigating. In fact, it were these two applications which have led us to the discovery of the calculus.

[10] Names are more pleasant for human beings (e.g. for debugging purposes), and we want our pen-and-paper proofs to be formalizable in a direct way. Moreover, to be user-friendly, implementations must use names, either internally or just for parsing and printing.

Explicit substitution calculi The first part of this work arose from trying to understand Chapter 4 of (Bonelli, 2001) on perpetuality in David and Guillaume's calculus with explicit substitutions λ_{ws}, in a named setting, cf. (Di Cosmo *et al.*, 2000), and in an atomic way. They introduce the λw-calculus as:

> We avoid the counter-example to the PSN property of the λs_e-calculus by adding to the usual syntax a new constructor that we call a *label* and which represents an updating information. The term t with label k (denoted by $\langle k \rangle t$) corresponds to the term t where all free indices have been increased by k (i.e. $\phi_0^k(t)$ in λs_e).
>
> In the terms we are finally interested in, two successive labels are not allowed. We first define preterms without this restriction.
>
> **Definition 3.1.** We define the set of λ_w-preterms by the following grammar:
>
> $$t ::= \underline{n} \mid \lambda t \mid (t\ t) \mid \langle k \rangle t \quad \text{with} \quad n, k \in \mathbb{N}$$

Observing that labels can be seen as repetitions of successors should make the relationship to the $\rotatebox[origin=c]{180}{λ}$-calculus clear.[11] In (Di Cosmo *et al.*, 2000) a named version of λ_w is presented having sets of names as labels. Our approach is more elementary as their labels are added only *after* α-conversion (not in their abstract syntax, but in their calculus), so they cannot get confluence of β-reduction without it.

Application: Preservation of Strong Normalisation The λ_{ws} calculus was introduced as a calculus having, among other desirable properties, the preservation of strong normalisation (PSN) property. From (David & Guillaume, 2001) we understand that λ_{ws} arose in a seemingly *ad hoc* way from barring counterexamples to PSN for existing calculi with explicit substitutions. We think the $\rotatebox[origin=c]{180}{λ}$-calculus offers an easy insight as to *why* the calculus works as follows.

The problem with PSN arises when one tries to orient, as a reduction rule, the critical pair arising from (an explicit version of) the substitution lemma. (see page 19). The problem with orienting the ensuing critical pair from right to left is that the resulting rule is non-left-linear (L occurs twice in its left-hand side), causing non-confluence, which is undesirable. However, orienting the critical pair from left to right is also problematic since the resulting rule is non-terminating, just by itself, since the left-hand side can be embedded into the right-hand side. (Note that this orientation corresponds to transforming from inside-out to outside-in (standard) order of contraction of the β-redexes.)

The key insight is that in the $\rotatebox[origin=c]{180}{λ}$-calculus, we can recognise the fact that we are already in outside-in order: consider the substitution lemma above oriented from left to right and enriched with end-of-scope information (but for the moment forgetting the first component of $\rotatebox[origin=c]{180}{λ}$-substitutions which are empty in this example):

$$M[x{:=}N, \square][y{:=}L, \square] \to M[y{:=}L, \underline{x}][\underline{x}{:=}N[y{:=}L, \square], \square]$$

Now we *recognise* that the two underlined xs in the right-hand side match with

[11] The relationship is not entirely trivial since in their λ_w calculus, David and Guilaume make use of commutativity and associativity of addition of natural numbers, whereas we are only allowed to manipulate *stacks* of names. However, one may reformulate their rules such that commutativity and associativity are not needed.

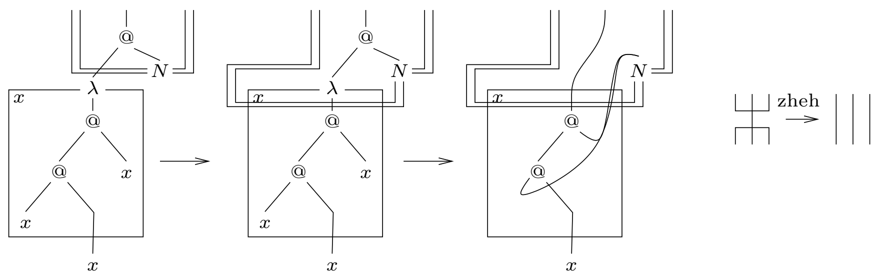

Fig. 5. Left: β-reduction: local scope extrusion and rewiring. Right: scope fusion.

one another, hence that these substitutions are already in standard order. Forbidding further applications of the rule in such situations, should break the infinite reduction and regain PSN (roughly speaking the maximal length of a reduction can be bounded by the length of a standard reduction, in the spirit of (Khasidashvili *et al.*, 2001)). Applying this idea to the closed and open substitution lemmas (Lemmas 4.10 and 4.13) should give rise to named versions of the following two λ_{ws} rules, see e.g. page 65 of (Bonelli, 2001):

$$\begin{array}{llll} M[k/N,l][\![i/P,j]\!] & \to_{lc1} & M[k/N[\![i-k/P,j]\!],j+l-1] & k \leq i < k+l \\ M[k/N,l[\![i/P,j]\!] & \to_{lc2} & M[\![i-l+1/P,j]\!][k/N[\![i-k/P,j]\!],l] & k+l \leq i \end{array}$$

which should yield a named version of λ_{ws} having PSN.

Localising scope extrusion The second part of this work arose from trying to understand Coppola's PhD thesis (2002) on the (complexity of) an optimal implementation of the λ-calculus. The idea is to view the boxes featuring in that work as our boxes, i.e. as representing scoping information, and to view their optimal implementation as a *local* implementation of our (minimal) scope extrusion.

Application: optimal reduction Lamping provided in (1990) the first implementation of the λ-calculus which was optimal in the sense of Lévy (1978). His implementation was based on a translation of λ-terms to graphs having nodes (fan-in and fan-out) for both explicit sharing and unsharing. In order for sharing and unsharing nodes to match up properly (the 'oracle'), he had to introduce two further types of nodes, the control nodes (square bracket and croissant). These control nodes had an *ad hoc* justification and their definitive understanding was considered to be the main open problem of this technique according to Chapter 9 of (Asperti & Guerrini, 1998).

We claim, the oracle can be understood to arise from making β-reduction in the ⅄-calculus *local* in the sense of (Lafont, 1990). That is scope extrusion and x-box removal as in Figure 3 are to be made local (replication is dealt with by the sharing nodes). A way in which this can be implemented is shown on the left in Figure 5. In fact, a key insight (cf. the second step of Figure 5) is that x-box removal is superfluous as long as scopes can always be moved out of the way (of a β-redex). We have a working optimal implementation of the λ-calculus based on rules achieving just that, such as the zheh-rule in Figure 5 for fusing two adjacent scopes. The implementation performs well on the examples in (Asperti & Guerrini, 1998), without the need for either their safe nodes or heuristics (we have only one control node). E.g. computing their most complex example, (f ten) in Figure 9.23 of (Asperti & Guerrini, 1998), takes us roughly 5 times as many interactions (compared to BOHM 1.1).[12]

References

Abadi, M., Cardelli, L., Curien, P.-L., & Lévy, J.-J. (1991). Explicit substitutions. *Journal of functional programming*, **1**(4), 375–416.

Asperti, A., & Guerrini, S. (1998). *The optimal implementation of functional programming languages*. Cambridge University Press.

Barendregt, H.P. (1984). *The lambda calculus: Its syntax and semantics*. Studies in Logic and the Foundations of Mathematics, vol. 103. North-Holland.

Baro, S., & Maurel, F. (2003). *The $q\nu$ and $q\nu k$ calculi : name capture and control*. PPS prépublication 16. Université Denis Diderot.

Berkling, K.J. (1976). *A symmetric complement to the lambda-calculus*. Interner Bericht ISF-76-7. GMD, D-5205, St. Augustin 1, West Germany.

Bird, R.S., & Paterson, R.A. (1999). De Bruijn notation as a nested datatype. *Journal of functional programming*, **9**(1), 77–91.

Blom, S.C.C. (2001). *Term graph rewriting, syntax and semantics*. Ph.D. thesis, Vrije Universiteit Amsterdam.

Bonelli, E. (2001). *Substitutions explicites et réécriture de termes*. Ph.D. thesis, Université Paris XI.

Cardelli, L., & Gordon, A.D. (1998). Mobile ambients. *Pages 140–155 of:* Nivat, M. (ed), *FOSSACS '98*. LNCS, vol. 1378. Springer.

Chen, C., & Xi, H. (2002). *Meta-programming through typeful code representation*. `http://www.cs.bu.edu/~hwxi/`.

Church, A. (1932). A set of postulates for the foundation of logic. *Annals of mathematics*, **33**, 346–366.

Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., & Tommasi, M. (1997). *Tree automata techniques and applications*. `http://www.grappa.univ-lille3.fr/tata`. October, 1st 2002.

Coppola, P. (2002). *On the complexity of optimal reduction of functional programming languages*. Ph.D. thesis, Università degli Studi di Udine.

David, R., & Guillaume, B. (2001). A λ-calculus with explicit weakening and explicit substitution. *Mathematical structures for computer science*, **11**, 169–206.

de Bruijn, N.G. (1972). Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation. *Indagationes mathematicae*, **34**, 381–392.

Di Cosmo, R., Kesner, D., & Polonovski, E. (2000). Proof nets and explicit substitutions. *Pages 63–81 of: Fossacs) '00*. LNCS, vol. 1784. Springer.

Gabbay, M.J., & Pitts, A.M. (1999). A new approach to abstract syntax involving binders. *Pages 214–224 of: LICS '99*. IEEE Computer Society.

Gordon, A.D., & Melham, T.F. (1996). Five axioms of alpha-conversion. *Pages 173–190 of:* Wright, J. von, Grundy, J., & Harrison, J. (eds), *TPHOLs '96*. LNCS, vol. 1125. Springer.

[12] The difference might be explainable by that we do not employ compound nodes.

Hendriks, D. (2003). *Metamathematics in Coq.* Ph.D. thesis, Department of Philosophy, Utrecht University. To appear.

Hendriks, D., & van Oostrom, V. (2003a). `http://www.preprints.uu.nl/lgpr`.

Hendriks, D., & van Oostrom, V. (2003b). ʎ. *CADE '03.* To appear.

Hollenberg, M., & Vermeulen, C.F.M. (1996). Counting variables in a dynamic setting. *Journal of logic and computation*, **6**(5), 725–744.

Kahrs, S. (1993). Context rewriting. *Pages 21–35 of:* Rusinowitch, M., & Rémy, J.-L. (eds), *CTRS '92.* LNCS, vol. 656. Springer.

Khasidashvili, Z., Ogawa, M., & van Oostrom, V. (2001). Uniform normalisation beyond orthogonality. *Pages 122–136 of:* Middeldorp, A. (ed), *RTA '01.* LNCS. Springer.

Lafont, Y. (1990). Interaction nets. *Pages 95–108 of: Popl '90.* ACM Press.

Lafont, Y. (1995). From proof-nets to interaction nets. *Pages 225–248 of:* Girard, J.-Y., Lafont, Y., & Regnier, L. (eds), *Advances in linear logic.* London Mathematical Society Lecture Note Series, vol. 222. Cambridge University Press.

Lamping, J. (1990). An algorithm for optimal lambda calculus reduction. *Pages 16–30 of: Popl '90.* ACM Press.

Lévy, J.-J. (1978). *Réductions correctes et optimales dans le λ-calcul.* Thèse de doctorat d'état, Université Paris VII.

Parrow, J. (1998). The fusion calculus: Expressiveness and symmetry in mobile processes. *Pages 176–185 of: LICS '98.* IEEE Computer Society.

Paterson, R.A. (1991). Non-deterministic lambda-calculus: A core for integrated languages. *Declarative programming.* SassBachwalden: Springer.

Schroer, D.E. (1965). *The Church–Rosser theorem.* Ph.D. thesis, Cornell University.

Terese. (2003). *Term rewriting systems.* Cambridge University Press.

Vestergaard, R., & Brotherston, J. (2001). A formalised first-order confluence proof for the λ-calculus using one-sorted variable names. *Pages 306–321 of:* Middeldorp, A. (ed), *RTA '01.* LNCS. Springer.