Lambda Calculi plus Letrec

Zena M. Ariola

Department of Computer & Information Sciences

University of Oregon. Eugene, OR 97401, USA

email: `ariola@cs.uoregon.edu`

Stefan Blom

Department of Mathematics and Computer Science

Vrije Universiteit, De Boelelaan 1081a, 1081 HV Amsterdam

email: `sccblom@cs.vu.nl`

The paper consists of three parts.

Part I: We establish an isomorphism between the *well-formed cyclic lambda-graphs* and their syntactic representations. To define the well-formed cyclic lambda-graphs we introduce the notion of a scoped lambda-graph. The well-formed lambda-graphs are those that have associated scoped lambda-graphs. The scoped lambda-graphs are represented by terms defined over lambda calculus extended with the `letrec` construct. On this set of terms we define a sound and complete axiom system (the representational calculus) that equates different representations of the same scoped lambda-graph. Since a well-formed lambda-graph can have different scoped lambda-graphs associated to it, we extend the representational calculus with axioms that equate different representations of the same well-formed graph. Finally, we consider the unwinding of well-formed graphs to possibly infinite trees and give a sound and complete axiomatization of tree unwinding.

Part II: We add computational power to our graphs and terms by defining β-reduction on scoped lambda-graphs and its associated notion on terms. The representational axiom system developed in the first part combined with β-reduction constitutes our cyclic extension of lambda calculus. In contrast to current theories, which impose restrictions on where the rewriting can take place, our reduction theory is very liberal, *e.g.,* it allows rewriting under lambda-abstractions and on cycles. As shown previously, the reduction theory is non-confluent. We thus introduce an approximate notion of confluence, which guarantees uniqueness of infinite normal forms. We show that the infinite normal form defines a congruence on the set of terms. We relate our cyclic lambda calculus to the plain lambda calculus and to the infinitary lambda calculi. We conclude by presenting a variant of our cyclic lambda calculus, which follows the tradition of the explicit substitution calculi.

Part III: Since most implementations of non-strict functional languages rely on sharing to avoid repeating computations, we develop a variant of our cyclic lambda calculus that enforces the sharing of computations and show that the two calculi are observationally equivalent. For reasoning about strict languages we develop a call-by-value variant of the sharing calculus. We state the difference between strict and non-strict computations in terms of different garbage collection rules. We relate the call-by-value calculus to Moggi's computational lambda calculus and to Hasegawa's calculus.

Note: The research of the first author is supported by NSF grants CCR-9410237 and CCR-9624711. The second author was partially supported by NWO grants SIR 13-3416 and SIR 13-3908 and by NSF grant CCR-9624711. A shorter version of this paper appears in the Proceedings of TACS '97 as "Cyclic Lambda Calculi" [AB97].

Contents

1. Introduction

Cyclic lambda-graphs are ubiquitous in a program development system [PJ87]. However, previous work falls short of capturing them in an adequate way. This lack of explicit treatment of cycles results in the loss of important intensional information and in weak theories that cannot express many transformations on recursive functions. For example, consider the following term:

$$\begin{array}{lll} M \equiv & \texttt{letrec} & \texttt{even} \ = \lambda x.\texttt{if}\ x = 0\ \texttt{then true else odd}(x\text{-}1) \\ & & \texttt{odd} \ = \lambda x.\texttt{if}\ x = 0\ \texttt{then false else even}(x\text{-}1) \\ & \texttt{in even}\ y & . \end{array}$$

(A note on syntax: the construct `letrec` $\cdots$ `in` $\cdots$ stands for a collection of unordered equations and a main expression written after the keyword `in`.) At compile time it might make sense to *unfold* or *inline* `odd` in the definition of `even`, triggering the *constant folding* and *unused lambda expression* transformations obtaining the term below:

$$\begin{array}{ll} N \equiv & \texttt{letrec even} \ = \lambda x.\texttt{if}\ x = 0\ \texttt{then true else if}\ x = 1\ \texttt{then false else even}(x\text{-}2) \\ & \texttt{in even}\ y \ . \end{array}$$

We can express terms M and N in the lambda calculus extended with pairs (denoted by $\langle\ ,\ \rangle$ with destructors `Fst` and `Snd`) and the μ-operator (rendered by the μ-rule $\mu x.M \to M[x := \mu x.M]$) as follows (we denote the translation by $[\![\cdot]\!]_\mu$):

$$\begin{array}{ll} [\![M]\!]_\mu \equiv & \texttt{let even\_odd} = \mu z.\langle\ \lambda x.\texttt{if}\ x = 0\ \texttt{then true else Snd}\ z\ (x\text{-}1), \\ & \qquad\qquad\qquad\quad \lambda x.\texttt{if}\ x = 0\ \texttt{then false else Fst}\ z\ (x\text{-}1)\rangle \\ & \texttt{in Fst even\_odd}\ y \\ [\![N]\!]_\mu \equiv & \texttt{let even} = \mu y.\lambda x.\texttt{if}\ x = 0\ \texttt{then true else if}\ x = 1\ \texttt{then false else}\ y(x\text{-}2) \\ & \texttt{in even}\ y \ . \end{array}$$

Note that $[\![M]\!]_\mu$ does not rewrite to $[\![N]\!]_\mu$ in $\lambda\mu$. The two terms are not even provably equal. This means that these simple inlining optimizations are not expressible as source-to-source transformations in $\lambda\mu$; thus, one cannot use the calculus to reason about their correctness or to study the efficiency of different application strategies.

Cycles are also important for reasoning about run-time issues. For example, the execution of M will involve a substitution of `even` in the main expression, followed by a β-reduction, obtaining:

$$\begin{array}{ll} \texttt{letrec} & \texttt{even} \ = \lambda x.\texttt{if}\ x = 0\ \texttt{then true else odd}(x\text{-}1) \\ & \texttt{odd} \ = \lambda x.\texttt{if}\ x = 0\ \texttt{then false else even}(x\text{-}1) \\ \multicolumn{2}{l}{\texttt{in if}\ y = 0\ \texttt{then true else odd}(y\text{-}1) \ .} \end{array}$$

Let us consider $[\![M]\!]_\mu$. We first apply the μ-rule to expose the lambda-abstraction, and then, as before, perform one substitution followed by a β-reduction, obtaining the following term, in which we have denoted the μ-expression occurring in $[\![M]\!]_\mu$ by P:

$$\begin{array}{ll} \texttt{let even\_odd} = & \langle \lambda x.\texttt{if}\ x = 0\ \texttt{then true else Snd}\ P\ (x\text{-}1), \\ & \ \ \lambda x.\texttt{if}\ x = 0\ \texttt{then false else Fst}\ P\ (x\text{-}1)\rangle \\ \multicolumn{2}{l}{\texttt{in if}\ y = 0\ \texttt{then true else Snd}\ P\ (y\text{-}1) \ .} \end{array}$$

The unsuitability of a calculus such as $\lambda\mu$ for reasoning about execution now comes to the surface. While the execution of M has made only one copy of `even`, the execution of $[\![M]\!]_\mu$ has created four copies of `even` and three copies of `odd`.

Interestingly enough, a theory of cycles turns out to be useful also for defining a parser. As described by Tomita [Tom85] and Billot et al. [BL89], a compact representation of all possible parse trees (that could be an infinite number) associated with a string is a cyclic graph, called a *parse forest*. The lack of a theory regarding cyclic objects has forced some research projects investigating automatic programming environment generators, such as the ASF+SDF system developed by Klint [Kli91], to apply a disambiguation process to remove the cycles [KV94] and so retrieving a tree object. Familiar rewriting can then be applied on that object. This disambiguation process would not be required if cycles were part of the rewriting technology.

We conclude that a theory of cycles is necessary if one wants to reason about compilation, optimization and execution of programs. Presentation of such a theory is the goal of this paper.

What makes a theory of cycles difficult to develop is that, once lambda-abstraction and cycles are admitted, confluence is lost, unless the theory is powerful enough to represent irregular structures, as shown in [AK94, AK]. To regain confluence, current formulations of cycles either impose restrictions, such as disallowing reduction under a lambda-abstraction or on a cycle ([Ros92, BLR96, Nie96, AK94, AK]), or adopt a framework based on interaction nets [Laf90]. As discussed in [Mac94] and [AL94] cycles do not destroy confluence in the context of interaction nets, but only at the expense of greater complexity.

In this paper, we limit our attention to cyclic lambda-graphs that occur in current implementations of strict and non-strict functional languages (we are not interested in optimality); thus, we will only consider cyclic lambda-graphs that unwind to regular trees. In contrast to the above mentioned approaches, we do not restrict the selection of redexes but introduce an alternative way of guaranteeing the consistency of the calculus. This consists of an approximate notion of confluence - *confluence up to information content*. This new notion is somewhat reminiscent of another 'approximate' notion of rewriting and confluence, namely, rewriting modulo an equivalence relation and confluence modulo equivalence [Hue80]. However, unlike these notions, we do not combine rewriting with an equivalence relation, but with an equally fundamental notion, namely a quasi order, expressing a comparison between the 'information content' of the objects in question. Explicit studies of such a combination of a rewrite relation with a quasi order are not abundant; the only study that we are aware of is [Sel96]. In the context of process algebra, Sangiorgi and Milner ([SM93]) also consider equivalences of processes up to an asymmetric relation, such as a quasi order, as a technique to prove bisimulation.

The paper is divided in three parts. The first part, consisting of Sections 2-8, is devoted to establishing an isomorphism between cyclic lambda-graphs and their syntactic representations. The second part, consisting of Sections 9-16, adds computational power to our graphs and terms and presents properties of the resulting equational theory. The third part, consisting of Sections 17-19, adds sharing to our previously developed calculus, resulting in two distinct calculi that are the foundation of strict and non-strict functional languages. An extension to data structures is also presented. We conclude in Section 20.

Part I: We start in Section 2 by introducing cyclic lambda-graphs. As in Wadsworth [Wad71], we do not deal with all possible lambda-graphs, but restrict our focus to the set of well-formed lambda-graphs. To define this class we introduce the notion of a scoped lambda-graph. The well-formed lambda-graphs are those that have associated scoped lambda-graphs. In Section 3, we introduce the syntactic formalism used to represent scoped lambda-graphs. Throughout this paper these syntactic objects are referred to as cyclic lambda terms. We present a mapping from cyclic lambda terms to scoped lambda-graphs. Since different cyclic terms can represent the same scoped lambda-graph, we introduce in Section 4 the *representational calculus* $\mathcal{R}_0$. $\mathcal{R}_0$ equates all distinct representations of a scoped lambda-graph. In the same section, we also introduce a confluent and terminating representational rewriting system, which allows us to associate a canonical representation to each scoped lambda-graph. Since we would also like to equate different representations of the same well-formed graph, we extend $\mathcal{R}_0$ with two other axioms in Section 5. We call the resulting system $\mathcal{R}_1$. In Section 6, we further extend $\mathcal{R}_1$ to make terms that unwind to the same tree provably equal. We call the calculus $\mathcal{R}_2$. In Section 7, we summarize the representational calculi and associated rewriting systems. In Section 8, we explore alternative representational rewriting systems and discuss their properties. In particular, we introduce the rewriting system $\mathcal{R}\circ^{\rightarrow}$ whose associated convertibility relation corresponds to the provable equality induced by $\mathcal{R}_2$. $\mathcal{R}\circ^{\rightarrow}$ constitutes our starting point for the next part.

Part II: In Section 9, we define β-reduction on scoped lambda-graphs and its associated notion on cyclic terms. $\mathcal{R}\circ^{\rightarrow}$ combined with β-reduction constitutes our cyclic lambda calculus. In Section 10, we introduce the notions of confluence up to a quasi order, of completeness up to a quasi order and of infinite normal form. We present some sufficient conditions that guarantee soundness of the infinite normal form with respect to reduction. A comparison between confluence, confluence modulo an equivalence relation and confluence up to a quasi order is also made. In the following two sections we introduce two applications of the notions introduced in this section. In Section 11, we show that $\mathcal{R}\circ^{\rightarrow}$ is confluent up to. This implies that we can define the tree unwinding of a cyclic term as the infinite normal form of that term. In Section 12, we show confluence up to of the cyclic lambda calculus. In Section 13, we prove that the infinite normal form defines a congruence with respect to the term formation rules, guaranteeing observational equivalence. In Sections 14 and 15, we relate our cyclic lambda calculus to the traditional lambda calculus and to the infinitary lambda calculus of Kennaway *et al.* [KKSdV95], respectively. In Section 16, we present a variant of our cyclic lambda calculus which follows the tradition of the explicit substitution calculi [ACCL91, Ros92].

Part III: In Section 17, we add the notion of sharing to our cyclic calculus for reasoning about current implementations of non-strict functional languages. In the call-by-name calculus every term is substitutable, while in the sharing calculus substitution is restricted to values, thus avoiding duplication of work. We show that this restriction does not change the infinite normal form of a cyclic term. For reasoning about strict languages, in Section 18, we introduce the cyclic call-by-value calculus, which is obtained by restricting the garbage collection axiom of the sharing calculus to collect values only. This expresses the fact that strict and non-strict computations capture the same amount of sharing. The call-by-value calculus is then equipped with a term model, which allows us to relate our calculus to the commutative version of Moggi's computational lambda calculus [Mog88] and to the recently developed calculus of Hasegawa [Has97]. An extension to data structures is presented in Section 19.

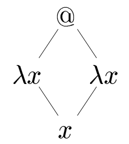


Figure 1. Wadsworth's non-admissible lambda-graph

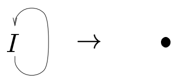


Figure 2. Why the black hole is needed

Part I

Graphs as terms and terms as graphs

2. Cyclic lambda-graphs and scoped lambda-graphs

We introduce a basic formalism for cyclic lambda-graphs in a format similar to the one used for first-order term graphs in [BvEG+87]. Following an idea used by Bourbaki in *Eléments de Théorie des Ensembles* to deal with quantifiers, an occurrence of a bound variable in a lambda-graph is represented by a back-pointer to the corresponding binding lambda-node. This implies we will not be able to represent the lambda-graph of Fig. 1, which Wadsworth [Wad71] calls a non-admissible lambda-graph. Each argument of a node is either a normal pointer to some node, a back-pointer to a lambda-node, or is a free variable from the set of variables $\mathcal{V}$. A normal pointer is denoted by v, w, a back-pointer by $\overline{v}, \overline{w}$ and a variable by x, y, z. We let $A(v)_i$ denote the i^{th} argument of node v. A graph has a root r, which can be anything that an argument can be. If the label of a node is $\bullet$ then that node is called a black hole. The black hole denotes provable non-termination. It was already introduced in the first-order case to be able to reduce a cyclic graph in the presence of collapsing rules, *i.e.,* rules of the form $Ix \to x$. Consider the reduction given in Fig. 2. If the cyclic graph on the left-hand side of the reduction reduces to itself then confluence is lost (see [AA95]). A more thorough discussion of the black hole is given in [AKK+94].

Notational conventions: given sets S and T, $\mathcal{P}(S)$ stands for the powerset of S, $S \setminus T$ stands for set difference, Σ^* stands for strings over the alphabet Σ, $|w|$ stands for the length of the string w, and $V \oplus W$ stands for the disjoint union of sets V and W. From now on, we will sometimes write graph for cyclic lambda-graph.

Definition 2.1. A lambda-graph is a tuple (V, L, A, r) where

- V is a set (possibly infinite) of nodes.
- $L : V \to \{\lambda, @, \bullet\}$ is a labeling function.
- $A : V \to (V \oplus \{\overline{v} \mid v \in V, L(v) = \lambda\} \oplus \mathcal{V})^*$ is a successor function such that

$$|A(v)| = \begin{cases} 0, \text{ if } L(v) = \bullet \\ 1, \text{ if } L(v) = \lambda \\ 2, \text{ if } L(v) = @ \end{cases}$$

Figure 3. Pointers versus back-pointers

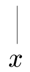


Figure 4. Labeled line

- $r \in (V \oplus \{\overline{v} \mid v \in V, L(v) = \lambda\} \oplus \mathcal{V})$.

In addition to the usual conventions on how to give a graphical representation of a cyclic lambda-graph, we follow the convention that if an arrow enters a lambda-node from above it represents a normal pointer and when it enters from below it represents a back-pointer. Thus, for example, we distinguish between the two lambda-graphs of Fig. 3. We draw a free variable as a labeled line and not as a node labeled x. For simplicity, we draw the label of the line at the end of the line. See the lambda-graph of Fig. 4 depicting a free variable x. We picture the root as a pointer or a labeled line that has no starting node. Fig. 5 shows some more examples of lambda-graphs. The lambda-graph on the left corresponds to a free variable x with a lambda-subgraph not accessible from the root. Note the difference between an arrow and a labeled line. Fig. 6 shows some examples of ill-formed lambda-graphs. In the left graph there are three errors: there is no root, the lambda has two arguments instead of one and the symbol x is shared by two labeled lines. In the middle graph the back-pointer to the lambda lacks an arrowhead and the labeled line is illegally drawn with an arrowhead. In the right graph there are two roots. Moreover, the top application does not have enough arguments. The fact that some pointers to the application nodes point to those nodes from below is unusual, but it is not wrong. This is because only back-pointers to lambda nodes are possible, so pointers to an application, even if they come from below, are always interpreted as normal pointers.

In the following, when we talk about a path in a lambda-graph, we mean a directed path using only the arrows representing normal pointers. We refer to a graph in which all nodes are reachable by a path starting at the root as a garbage free graph. We assume that equality between graphs stands for rooted graph isomorphism.

In this paper, we will only deal with a subset of all possible lambda-graphs. For example, we will not consider the third lambda-graph of Fig. 5. To characterize this subset, referred to as the subset of the *well-formed lambda-graphs*, we introduce the notion of scope, which associates a set of nodes

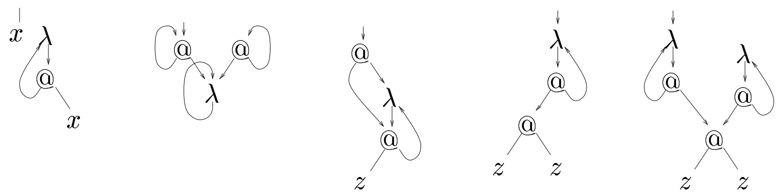


Figure 5. Examples of lambda-graphs

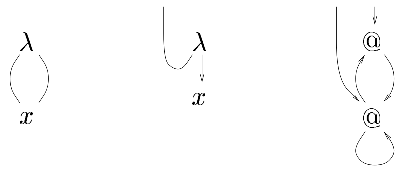


Figure 6. Examples of ill-formed lambda-graphs

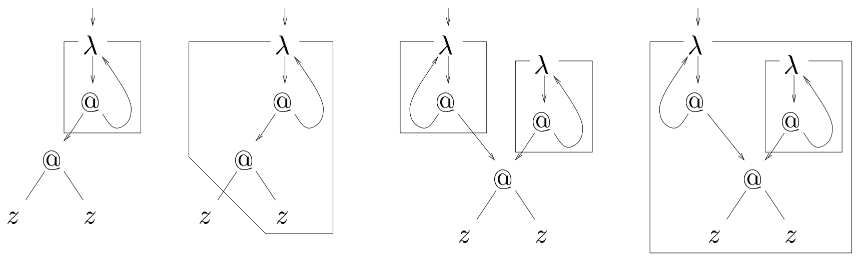


Figure 7. Examples of scoped lambda-graphs

to a lambda-node. Intuitively, as it will be discussed in the second part, the scope of a lambda-node v corresponds to that part of the graph that is copied when a β-reduction is performed with v denoting the function part of the application. This means that at run-time there is no decision to be made on how much copying to do. This differs from Wadsworth's graph reduction [Wad71], in which the amount of copying is determined at run-time. Since in our approach the copying occurs at once, we rule out optimal implementations which, as discussed in [Mac94], require to copy on a node-by-node basis.

Definition 2.2. A scoped lambda-graph is a tuple (V, L, A, S, r), where

- (V, L, A, r) is a lambda-graph
- $S : \{v \in V | L(v) = \lambda\} \to \mathcal{P}(V)$ is a function such that for every lambda-node v the following axioms apply.
  - *auto:* $v \in S(v)$
  - *bind:* $\forall w : \overline{v}$ is an argument of w implies $w \in S(v)$
  - *upward-closure:* If $w_2 \not\equiv v$ and w_2 is an argument of w_1 and $w_2 \in S(v)$ then $w_1 \in S(v)$.
  - *nesting:* $\forall$ lambda-nodes $w : S(w) \cap S(v) = \{\}$ or $S(w) \subseteq S(v)\backslash\{v\}$ or $S(v) \subseteq S(w)\backslash\{w\}$
- *root condition:* $r \in \mathcal{V}$ or $r \in V$ such that $\forall v : r \not\in S(v)\backslash\{v\}$.

For a lambda-node v we refer to $S(v)$ as the scope of v. We depict the scope of v by drawing a line starting at one side of the lambda-node around all other nodes that are a member of the scope ending at the other side of the lambda-node. Note that the upward-closure axiom is logically equivalent to: if $w_1 \not\in S(v)$, $w_2 \in S(v)$ and w_2 is an argument of w_1 then $w_2 \equiv v$.

Example 2.3. The graphs of Fig. 7 and 8 are examples of scoped and ill-formed scoped graphs, respectively. Since the scope of a lambda-node involves nodes and not edges, we can take the liberty of drawing the label of an edge either inside or outside the scope, *e.g.,* see the second graph of Fig.

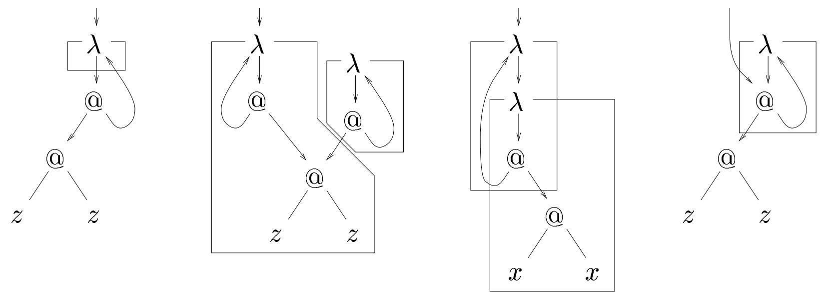


Figure 8. Examples of ill-formed scoped lambda-graphs

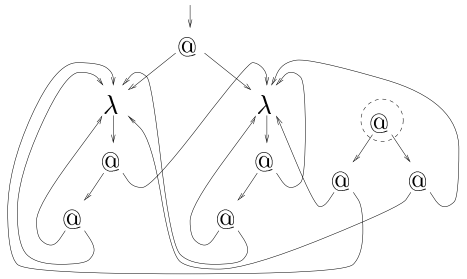


Figure 9. Example of ill-formed lambda-graph

7 in which the left label is outside the scope and the right-one is inside. However, we follow the convention of keeping the number of line crossings as small as possible. Thus, we draw the labels inside the box.

The first graph of Fig. 8 violates the *bind* scope axiom because the first application node has a back-pointer to the lambda-node v and is not a member of the scope of v. The second graph violates the *upward-closure* scope axiom because the scope of the upper lambda-node v is entered from a node different than v. The third graph violates the *nesting* scope axiom because the intersection of the two scopes is non-empty without one being a proper subset of the other. The fourth graph violates the *root* condition because the root points to a node inside a scope.

We remark that the *upward-closure* axiom implies that if you have an acyclic path from a lambda-node v to a node w in the scope of v then every node on that path is in the scope of v. However, the converse does not hold. That is, if for every acyclic path from a lambda-node v to a node w in the scope of v we have that every node on that path is in the scope of v then we do not have the upward-closure axiom; *e.g.,* the second graph in Fig. 8 satisfies the condition on paths, but not the upward-closure axiom.

By definition we get a lambda-graph when we drop the scope function from the tuple. However, not every lambda-graph is obtained by stripping the scope function from the tuple.

Definition 2.4. A lambda-graph (V, L, A, r) is well-formed if there exists a scope function S such that (V, L, A, S, r) is a scoped graph.

The garbage free and acyclic well-formed graphs correspond to the admissible graphs described by Wadsworth [Wad71].

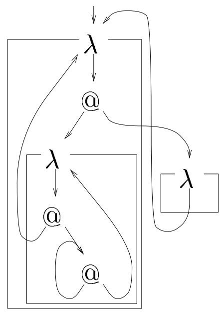


Figure 10. A graph with a minimal scope function

Example 2.5. The lambda-graph of Fig. 9 is not well-formed. According to the *root condition* the top application node cannot be the member of any scope. This forces the scopes of the two lambda-nodes to be disjoint. If we forget the circled application node then there is exactly one possible scope function for the graph: the scope of the left lambda-node is the node itself, the two application nodes below it and the right garbage application node. The scope of the second lambda-node consists of the node itself, the application nodes below it and the left garbage application node. When we do have the circled node then we are forced to add it to both scopes because it refers to a node in both scopes and is not the lambda-node. This violates the *nesting* scope axiom.

From now on we will assume that a graph consists of a finite set of nodes unless stated otherwise.

For garbage free graphs there is a necessary and sufficient condition to ensure that they are well-formed.

Proposition 2.6. *Given a garbage free graph $g \equiv (V, L, A, r)$. Then g is well-formed iff for every node v with a back-pointer to w, w is on each path from the root to v.*

Proof.

- $\Rightarrow$ Since the graph is well-formed it means there exists a function S such that (V, L, A, S, r) is a scoped graph. Given this scoped graph and a node v with a back-pointer to w and a path from r to v. If r is not a member of the scope of w then by the *upward-closure* axiom every path to v must pass through w because v is a member of the scope of w. If r is a member of the scope of w then according to the *root condition* $r \equiv w$ and therefore w is on the path.

- $\Leftarrow$ Given a lambda-graph that satisfies the condition we want to define a scope function S. We do this by defining a relation S which can be read as a function by taking $S(w) = \{v \mid w\ S\ v\}$. We define S as the smallest relation such that:

  - For every lambda-node w we have $w\ S\ w$.
  - If there exists an acyclic path from a lambda-node w to a node v_1 to a node v_2 with a back-pointer to w then $w\ S\ v_1$.
  - If $w_1\ S\ w_2$ and $w_2\ S\ v$ then $w_1\ S\ v$.

  The intention is that the first clause forces the auto axiom, the second clause forces both the bind and upward-closure axioms, and the third clause forces the nesting axiom.

  Before showing that the function S defined in this way is actually a scope function we give an example of a scope function defined in this way. Consider Fig. 10. From the top down let us call the lambda-nodes l_1, l_2 and l_3 and the application nodes a_1, a_2 and a_3. The first clause forces the lambda-nodes to be members of their own scopes. By the second clause, we have that l_2, a_2 and a_3 are members of the scope of l_2. By the same clause, we have that l_1, a_1, l_2 and a_2 are members of the scope of l_1. The third clause adds a_3 to the scope of l_1. Note that the second clause makes l_1 and l_2 members of their own scope, but does not make l_3 a member of the scope of l_3, therefore the first clause is not superfluous. Also note that the second clause does not require l_3 to be in the scope of l_1, since the path from l_1 to l_3 and a_2 is cyclic. The final result is $S(l_1) = \{l_1, a_1, l_2, a_2, a_3\}, S(l_2) = \{l_2, a_2, a_3\}, S(l_3) = \{l_3\}$.

To show that S is a scope function, let v be a lambda-node then:

- *auto:* $v \in S(v)$ by the first clause.
- *bind:* If a node w has $\overline{v}$ as an argument then because the graph is garbage free there must be a path from the root to w. By the assumption this path goes through v and so by the second clause we have that $w \in S(v)$.
- *upward-closure:* Suppose $w_1 \notin S(v)$ and $w_2 \in S(v)$ and w_2 is an argument of w_1. Suppose $w_2 \not\equiv v$. w_2 must have been included in $S(v)$ by application of one of the following clauses:
  - There exists a path from v to w_2 to w', where w' has $\overline{v}$ as argument. Because the graph is garbage free there is a path from r to w_1. So there also exists a path from r to w_1 to w_2 to w'. This can only be the case if v is on the path from r to w_1, but that would imply that $w_1 \in S(v)$. Contradiction.
  - There exists a w' such that vSw' and $w'Sw_2$, where w' is neither v or w_2. Suppose $w_1 \in S(w')$ then by the third clause $w_1 \in S(v)$. This means that $w_1 \notin S(w')$. We also have that $w_2 \in S(w')$. We then repeat the analysis and eventually end up in the previous case.
- *root condition:* Before proving that the nesting axiom holds, we prove that the condition on the root holds.
  - If the root is a variable then the graph has no nodes and the condition is met.
  - If the root is an application node then in order to be in some scope the root would have to be on a path from a lambda-node to a node with a back-pointer to that lambda-node. So there would be a path from the root to a node with a back-pointer that does not pass through the lambda-node first. Contradiction. So the application node cannot be the member of any scope.
  - If the root is a lambda-node then the root is obviously the member of the scope of itself. The root is however not the member of any other scope, by applying a similar argument as in the previous case.
- *nesting:* Let v_1 and v_2 be two different lambda-nodes. Let $w \in S(v_1) \cap S(v_2)$. Since the graph is garbage free there must be a path from the root to w. If the root is not a member of the scope of v_1 or v_2 then that path passes through v_1 and v_2 because w is a member of both scopes and by the upward-closure condition proven above. Let us assume it passes through v_1 first. This means that v_2 is in the scope of v_1 and therefore by the third clause $S(v_2) \subseteq S(v_1)$. Suppose $v_1 \in S(v_2)$. Then every path from the root to v_1 must pass through v_2 first. Contradiction. So $S(v_2) \subseteq (S(v_1) \setminus \{v_1\})$.

  If the root is a member of the scope of v_1 or v_2 then let us assume the root is v_1. By the root condition shown above we have that $v_1 \notin S(v_2)$. By the same reasoning as above we have that $S(v_2) \subseteq S(v_1)$ and so $S(v_2) \subseteq (S(v_1) \setminus \{v_1\})$.

□

Note that the third graph in Fig. 5 is not well-formed since it does not meet the condition stated in the proposition above.

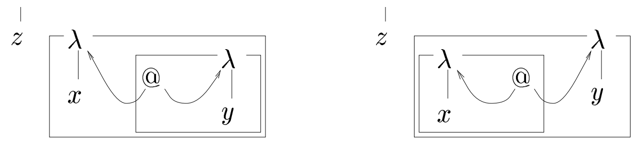


Figure 11. Two scoped versions of the same graph

PROPOSITION 2.7. *Given a well-formed garbage free graph* $g \equiv (V, L, A, r)$. *Then* g *has a unique minimal scope function* S, i.e., *for all scope functions* S'*:*

$$\forall v \in V : L(v) = \lambda \Rightarrow S(v) \subseteq S'(v) \ .$$

PROOF. It is not difficult to show that the scope function S built in the proof of Proposition 2.6 satisfies this property. □

REMARK 2.8. If a well-formed graph is not garbage free then there need not exist a minimal scope function. In Fig. 11 we have drawn two scoped versions of the same graph. Both scope functions are minimal. One could also ask about the existence of maximal scopes. The graphs in Fig. 11 also are a counterexample against the existence of maximal scopes for graphs with garbage. For every well-formed garbage free graph, we can define the unique maximal scoping by $w \in S(v)$, if all paths from the root to w pass through the lambda-node v.

The minimal scope corresponds to that part of a lambda-graph that is copied during reduction by Wadsworth's graph interpreter.

3. Cyclic lambda terms: a syntactic representation of scoped lambda-graphs

We now introduce the syntactic formalism used to represent finite scoped graphs. The same formalism was already introduced in [AK96] for the first-order case, and extended with lambda-abstraction in [AK94, AK]. However, in that work a precise connection between terms and lambda-graphs was not established.

DEFINITION 3.1. The following clauses define the syntax of cyclic lambda terms:

$$\begin{array}{llcl} \text{Terms } (\Lambda\circ) & M & ::= & x \mid \lambda x.M \mid M\,N \mid \langle M \mid D\rangle \\ \text{Declarations} & D & ::= & x_1 = M_1, \ldots, x_n = M_n \end{array}$$

where the *recursion variables* $x_i, 1 \leq i \leq n$, are distinct from each other.

In other words, the set of cyclic lambda terms consists of the lambda calculus terms (*i.e.,* variables, abstractions and applications) and the `letrec` construct:

$$\langle M \mid x_1 = M_1, \cdots, x_n = M_n\rangle \ .$$

We sometimes refer to M and D as the external and internal part of $\langle M \mid D\rangle$. Terms that differ in the order of the equations are identified. In the earlier papers on this formalism [AK96, AK94, AK] the entire letrec construct was referred to as a box. In the graphs drawn in those papers there are boxes that represent these constructs. They should not be confused with the boxes we have used

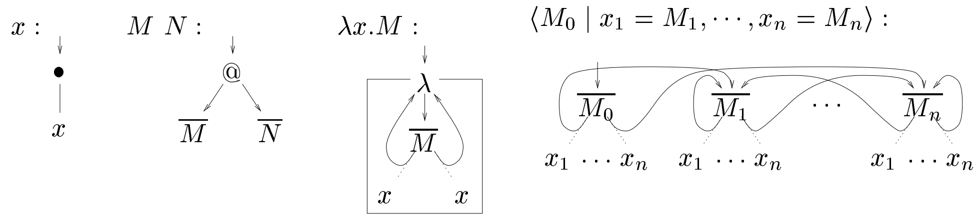


Figure 12. Pictorial definition of ρ_{pre}

here to represent scopes. Scopes are associated to lambda's and do not reflect the nesting of letrec's in a term.

We adopt the following notation: A context C is a term with a hole $\Box$ in the place of one subterm. The operation of filling the context C with a term M yields the term $C[M]$, possibly capturing some free variables of M in the process. By convention, bound and free variables are distinct from each other. $M[x := N]$ stands for the substitution of N for each free occurrence of x in M, without capturing free variables in N. $\equiv$ stands for syntactic equivalence up to α-renaming, applied to both lambda-bound variables and recursion variables. If D_1 and D_2 are the lists of declarations $x_1 = M_1, \cdots, x_m = M_m$ and $y_1 = N_1, \cdots, y_n = M_n$, respectively, such that $\forall i, j : x_i \not\equiv y_j$ then we denote the list of declarations $x_1 = M_1, \cdots, x_m = M_m, y_1 = N_1, \cdots, y_n = M_n$ by D_1, D_2. When it is convenient to do so we sometimes denote a list of declarations as a set, *e.g.*, $D_1 = \{x_1 = M_1, \cdots, x_m = M_m\}$. For example if $D_1 = \{x = y\}, D_2 = \{y = x\}$ then we cannot write D_1, D_1, but D_1, D_2 is fine. Let $\underset{R}{\rightarrow}$ be a reduction relation then $\underset{R}{\leftrightarrow}$ denotes the convertibility relation induced by $\underset{R}{\rightarrow}$ and $\underset{R}{\twoheadrightarrow}$ denotes the reflexive and transitive closure of $\underset{R}{\rightarrow}$.

3.1. Mapping cyclic lambda terms to scoped lambda-graphs

We define a mapping from cyclic terms to scoped graphs to give graph-semantics to cyclic terms. To simplify the definition of this mapping we introduce the notion of a *(scoped) pre-graph*, in which the condition on the arity of a black hole is relaxed.

Definition 3.2. A (scoped) pre-graph is a (scoped) graph where a node labeled with $\bullet$ may have 0 or 1 argument(s). If such a node has arity 0 we still call it a black hole but if it has arity 1 we call it an indirection node.

We have chosen the same symbol for indirection nodes and black holes because a black hole can be seen as a special case of an indirection node: an indirection node that refers to itself and therefore cannot be removed.

We map cyclic terms to scoped graphs via the mappings ρ_{pre} and Sim:

$$\begin{array}{rlcl} \rho_{\text{pre}}: & \Lambda\circ & \rightarrow & \text{scoped pre-graphs} \\ \text{Sim}: & \text{scoped pre-graphs} & \rightarrow & \text{scoped graphs}\ . \end{array}$$

The mapping ρ_{pre} transforms every lambda, every application and every occurrence of a variable, that is not on the left-hand side of an =-sign, into a node of the appropriate type (lambda, application and indirection, respectively). For a term $\lambda x.M$ the scope of the lambda contains the lambda-node itself and every node corresponding to the subterm M. Formally, we have:

Definition 3.3. Given $M \in \Lambda\circ$, $\rho_{\text{pre}}(M)$ is constructed recursively as follows:

- $\rho_{\text{pre}}(x)$ is the scoped pre-graph with a single node labeled $\bullet$ as the root with x as the argument.

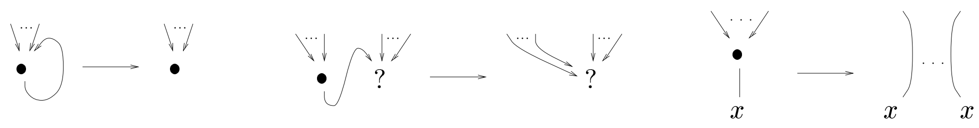


Figure 13. Simplification of scoped pre-graphs

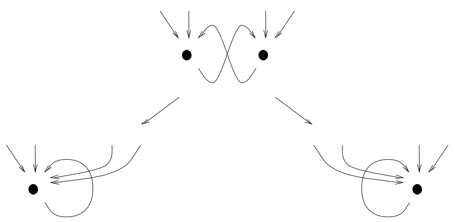


Figure 14. Example of simplification of mutually referring indirection nodes

- $\rho_{\text{pre}}(M\ N)$ is the union of the scoped pre-graphs $\rho_{\text{pre}}(M)$ and $\rho_{\text{pre}}(N)$ with a new root labeled @ whose arguments are the roots of $\rho_{\text{pre}}(M)$ and $\rho_{\text{pre}}(N)$.

- $\rho_{\text{pre}}(\lambda x.M)$ is the scoped pre-graph $\rho_{\text{pre}}(M)$ plus a new root node v labeled with λ that has the root of $\rho_{\text{pre}}(M)$ as it's argument and with every argument x in $\rho_{\text{pre}}(M)$ replaced with a back-pointer to v. The scope of the lambda-node v is the complete set of nodes of the generated graph.

- $\rho_{\text{pre}}(\langle M_0 \mid x_1 = M_1, \cdots, x_n = M_n\rangle)$ is the union of the scoped pre-graphs $\rho_{\text{pre}}(M_0)$ up to $\rho_{\text{pre}}(M_n)$ with the root of $\rho_{\text{pre}}(M_0)$ as root and every argument x_i, $i = 1, \cdots, n$ replaced by the root of $\rho_{\text{pre}}(M_i)$.

The pictorial equivalent of the above definition is given in Fig. 12, in which for simplicity we denote $\rho_{\text{pre}}(M)$ by $\overline{M}$. Note that in the case for a lambda-abstraction each labeled line x is transformed into a back-pointer. Instead, in the case for a letrec expression the labeled lines $x_1 \cdots x_n$ are transformed into pointers.

The mapping Sim transforms a scoped pre-graph into a scoped graph by removing all indirection nodes or transforming them into black holes. This is accomplished by the three rewriting rules described in Fig. 13. The first rule transforms an indirection node that has itself as argument into a black hole simply by forgetting the argument. The second rule replaces all pointers to an indirection node v, that points to a different node w, by copies of the (back-)pointer from v to w and then removes v. The last rule removes an indirection node v that has a free variable x as argument and changes every pointer to v to a line labeled x. The result of the normal form of these operations is well defined because we have local confluence and termination. The termination part is trivial. Local confluence follows mostly from the fact that redexes are disjoint. To this general case there is one exception. When two indirection nodes refer to each other, as in Fig. 14, then a redex may be destroyed. However, the results of contracting the two redexes are isomorphic, so local confluence holds.

We will refer to Sim(g) as the *simplification* of scoped pre-graph g. The mapping ρ from cyclic terms to scoped graphs is then obtained by composing ρ_{pre} and Sim.

Definition 3.4. Given $M \in \Lambda\circ$. The scoped graph $\rho(M)$ is $\text{Sim}(\rho_{\text{pre}}(M))$.

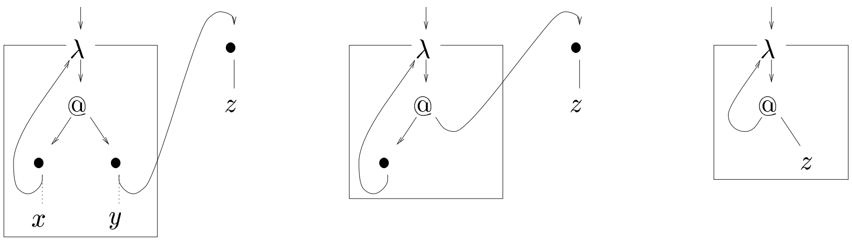


Figure 15. Construction of $\rho(\langle \lambda x.x\ y \mid y = z\rangle)$

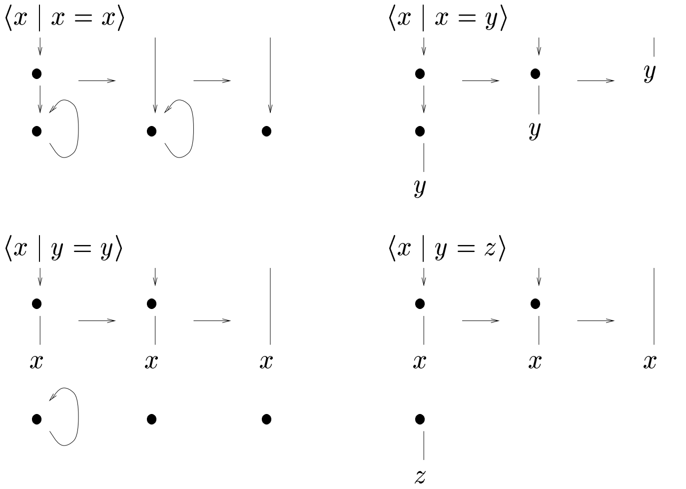


Figure 16. How simplification works with only variables

Example 3.5. In Fig. 15 from left to right we have $\rho_{\text{pre}}(\langle \lambda x.x\ y \mid y = z\rangle)$, the result of contracting one simplification redex and the result of contracting the remaining two simplification redexes. Note how in the second step the node labeled $\bullet$ with z as an argument gets erased by giving the label z to the second argument of the application node. In Fig. 16 we show ρ_{pre} of $\langle x \mid x = x\rangle$, $\langle x \mid x = y\rangle$, $\langle x \mid y = y\rangle$ and $\langle x \mid y = z\rangle$ and the rewriting sequences of those pre-graphs to their simplified form. Considering the last graph note that the indirection node gets deleted. This implies that even if in the graph the reference to a node is unique, the same is not necessarily true in the term. For example, consider $M \equiv \langle x \mid x = \lambda y.y, z = x\rangle$. In $\rho(M)$ the reference to the lambda is unique, in the term it is not.

Definition 3.6. Given $M \in \Lambda\circ$ and a scoped graph g. Then M represents g if $\rho(M) = g$.

Remark 3.7. Both ρ_{pre} and ρ map many different cyclic terms to the same graph. For example,

$$\langle x\ y \mid x = z, y = z\rangle \not\equiv \langle x \mid x = z\rangle\ \langle y \mid y = z\rangle\ ,$$

but

$$\rho_{\text{pre}}(\langle x\ y \mid x = z, y = z\rangle) = \rho_{\text{pre}}(\langle x \mid x = z\rangle\ \langle y \mid y = z\rangle)\ .$$

The ρ_{pre} of both terms is the pre-graph on the left of Fig 17.

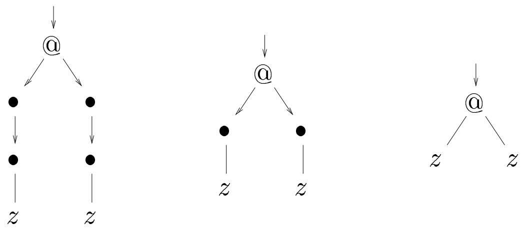


Figure 17. An illustration of representation

ρ_{pre} makes more distinctions than ρ, *e.g.*,

$$\rho_{\text{pre}}(\langle x\ y \mid x = z, y = z\rangle) \neq \rho_{\text{pre}}(zz) \text{ but } \rho(\langle x\ y \mid x = z, y = z\rangle) = \rho(zz)\ .$$

The ρ_{pre} of term zz is the middle pre-graph of Fig. 17. The ρ of the above three terms is the graph on the right of Fig. 17.

Next, we define a sound and complete axiom system on cyclic terms which we call the *representational system*. All terms that are provably equal in the representational system map to the same scoped graph, and all terms that map to the same scoped graph are provable equal.

4. Sound and complete axiomatization of scoped lambda-graphs

For all calculi developed in the paper, we assume the presence of the following axiom and inference rules that make provable equality a congruence relation.

$$\begin{array}{rcl} & & M = M \\ M = N & \Rightarrow & N = M \\ M = N,\ N = P & \Rightarrow & M = P \\ M = N & \Rightarrow & C[M] = C[N]\ . \end{array}$$

In giving the axioms we assume that no variable capture occurs, *e.g.*, $x\langle x \mid x = z\rangle$ is not equated to $\langle xx \mid x = z\rangle$.

Definition 4.1. The representational calculus $\mathcal{R}_0$ is given by the following axioms:

Lift:

$$\begin{array}{rcl} \langle M \mid D\rangle\ N & = & \langle M\ N \mid D\rangle \\ M\ \langle N \mid D\rangle & = & \langle M\ N \mid D\rangle \end{array}$$

Empty box garbage collection:

$$\langle M \mid \rangle = M$$

Merge:

$$\begin{array}{rcl} \langle\langle M \mid D_1\rangle \mid D_2\rangle & = & \langle M \mid D_1, D_2\rangle \\ \langle M \mid x = \langle N \mid D_1\rangle, D_2\rangle & = & \langle M \mid x = N, D_1, D_2\rangle \end{array}$$

Naming:

$$M = \langle x \mid x = M\rangle \qquad x \text{ a new variable}$$

Variable substitution:

$$\langle M \mid x = y, D\rangle = \langle M[x := y] \mid D[x := y]\rangle \qquad x \not\equiv y$$

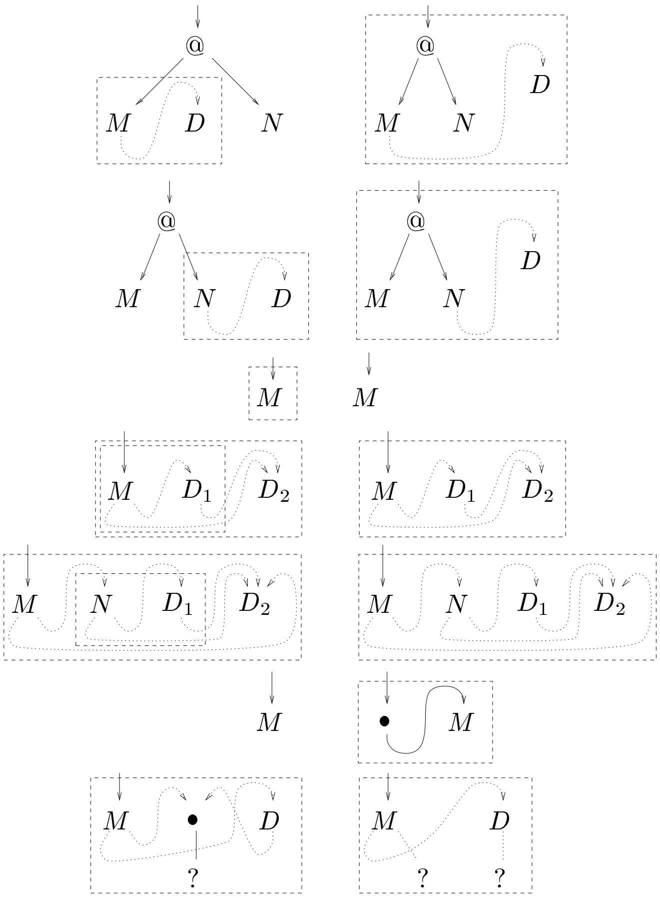


Figure 18. Pictorial description of $\mathcal{R}_0$

The first five axioms move the bindings around the term. The other two axioms deal with indirection nodes. They abstract away from the names of the nodes.

We can now prove that $\mathcal{R}_0$ is sound with respect to scoped graphs.

THEOREM 4.2. *Given* $M, N \in \Lambda\circ$. *If* $\mathcal{R}_0 \vdash M = N$ *then* $\rho(M) = \rho(N)$.

PROOF. When we construct ρ_{pre} for every left and right-hand side of every axiom we conclude that for the first five axioms the result is the same and for the last two the results simplify to the same graph. See the pictorial description in Fig. 18, in which we have drawn dashed boxes to indicate where the letrec constructs in the original term were. A dotted edge stands for zero or more references to variables in a set of definitions. A question mark stands for either a variable or a pointer to a set of definitions. □

4.1. A representational rewriting system

Among all possible representations of a graph, we would like to select a canonical one. To that end, we characterize the provable equality of the representational calculus as a confluent and terminating rewriting system[†]. We can then take as the canonical representation of a graph the unique normal form of one of its representations.

We start by orienting the axioms from left to right. To guarantee termination we then impose some restrictions on the naming axiom. These restrictions have to disallow reductions such as the following ones:

$$\begin{array}{l} x \to \langle z \mid z = x\rangle \to \langle w \mid w = \langle z \mid z = x\rangle\rangle \to \cdots \\ x \to \langle y \mid y = x\rangle \to \langle x \mid\rangle \to x \to \cdots \end{array}$$

We first replace the naming rule with the following two rules:

$$\begin{array}{rcl} \lambda x.M & \to & \langle y \mid y = \lambda x.M\rangle \\ M\ N & \to & \langle x \mid x = M\ N\rangle \ , \end{array}$$

that is, we only give names to lambda-abstractions and applications. Next, we only give names to expressions that do not have a name already, which is when an expression is embedded inside zero or more external parts of boxes and occurs at the top or as the direct subterm of a lambda or an application. This is formalized by restricting the application of the above two rules to safe contexts.

DEFINITION 4.3. A safe context C_{safe} is defined as:

$$\begin{array}{rcl} C_{\text{safe}} & ::= & C' \mid C[\lambda x.C'] \mid C[C'\ M] \mid C[M\ C'] \\ C' & ::= & \square \mid \langle C' \mid D\rangle \end{array}$$

EXAMPLE 4.4. Here are some examples of naming steps and the corresponding safe context:

$$\begin{array}{rcll} M\ N\ P & \to & \langle x \mid x = M\ N\rangle\ P & C_{\text{safe}} \equiv \square P \\ \langle x\ P \mid x = M\ N\rangle & \to & \langle\langle y \mid y = x\ P\rangle \mid x = M\ N\rangle & C_{\text{safe}} \equiv \langle\square \mid x = MN\rangle \\ \lambda x.\langle\lambda y.M \mid z = x\rangle & \to & \lambda x.\langle\langle u \mid u = \lambda y.M\rangle \mid z = x\rangle & C_{\text{safe}} \equiv \lambda x.\langle\square \mid z = x\rangle \ . \end{array}$$

The following naming steps are not allowed:

$$\begin{array}{rcll} x & \to & \langle y \mid y = x\rangle & C_{\text{safe}} \equiv \square \\ \langle M\ N \mid D\rangle & \to & \langle x \mid x = \langle M\ N \mid D\rangle\rangle & C_{\text{safe}} \equiv \square \\ \langle M \mid x = P\ Q\rangle & \to & \langle M \mid x = \langle y \mid y = P\ Q\rangle\rangle & C \equiv \langle M \mid x = \square\rangle \ . \end{array}$$

In the first two reductions we do not apply the proper naming rule. Instead, the context in the last reduction is not safe.

DEFINITION 4.5. The representational rewriting system $\mathcal{R}_0^{\to}$ is given by the following rewriting rules:

$$\begin{array}{rcll} \langle M \mid D\rangle\ N & \to & \langle M\ N \mid D\rangle & \\ M\ \langle N \mid D\rangle & \to & \langle M\ N \mid D\rangle & \\ \langle M \mid\rangle & \to & M & \\ \langle\langle M \mid D_1\rangle \mid D_2\rangle & \to & \langle M \mid D_1, D_2\rangle & \\ \langle M \mid x = \langle N \mid D_1\rangle, D_2\rangle & \to & \langle M \mid x = N, D_1, D_2\rangle & \\ \langle M \mid x = y, D\rangle & \to & \langle M[x := y] \mid D[x := y]\rangle & x \not\equiv y \\ C_{\text{safe}}[\lambda x.M] & \to & C_{\text{safe}}[\langle y \mid y = \lambda x.M\rangle] & y \text{ a new variable} \\ C_{\text{safe}}[M\ N] & \to & C_{\text{safe}}[\langle y \mid y = M\ N\rangle] & y \text{ a new variable} \end{array}$$

Even though the naming rule is restricted, we still have that the provable equality of the representational calculus is the same as the convertibility of the rewriting system.

[†] Also in the proof of completeness we will make use of the fact that the rewriting system is confluent and terminating.

PROPOSITION 4.6. *Given* $M, N \in \Lambda\circ$. $\mathcal{R}_0 \vdash M = N$ *iff* $M \xleftrightarrow[\mathcal{R}_0^{\rightarrow}]{} N$.

PROOF. Follows from the fact that if two terms are equal by applying the naming axiom in some context then they have a common reduct. Formally, given a cyclic term M and a context C, there exists an N such that

$$C[M] \twoheadrightarrow N \text{ and } C[\langle x \mid x = M\rangle] \twoheadrightarrow N \ ,$$

for x not occurring in M. We prove this fact by cases on the context.

- C is a safe context. We can then write M as $\langle \cdots \langle M' \mid D_1\rangle \cdots D_m\rangle$, where M' is either a variable, an application or an abstraction. If M' is not a variable then

$$C[\langle \cdots \langle M' \mid D_1\rangle \cdots D_m\rangle] \rightarrow C[\langle \cdots \langle\langle x \mid x = M'\rangle \mid D_1\rangle \cdots D_m\rangle] \twoheadrightarrow C[\langle x \mid x = M', D_1, \cdots, D_m\rangle]$$

and

$$C[\langle x \mid x = \langle \cdots \langle M' \mid D_1\rangle \cdots D_m\rangle] \twoheadrightarrow C[\langle x \mid x = M', D_1, \cdots, D_m\rangle] \ .$$

If M' is a variable, say y, then if $m = 0$ we just apply variable substitution and empty box removal. Otherwise,

$$C[\langle \cdots \langle y \mid D_1\rangle \cdots D_m\rangle] \twoheadrightarrow C[\langle y \mid D_1, \cdots, D_m\rangle]$$

and

$$C[\langle x \mid x = \langle \cdots \langle y \mid D_1\rangle \cdots D_m\rangle\rangle] \twoheadrightarrow C[\langle x \mid x = y, D_1, \cdots, D_m\rangle] \rightarrow C[\langle y \mid D_1, \cdots, D_m\rangle]$$

- C is not a safe context. This means that we can write C as

$$C'[\langle P \mid y = \langle \cdots \langle \Box \mid D_1\rangle \cdots D_n\rangle, D\rangle] \ .$$

It then follows that

$$C[M] \equiv C'[\langle P \mid y = \langle \cdots \langle M \mid D_1\rangle \cdots D_n\rangle, D\rangle] \twoheadrightarrow C'[\langle P \mid y = M, D_1, \cdots, D_n, D\rangle]$$

and

$$\begin{array}{rcl} C[\langle x \mid x = M\rangle] & \equiv & C'[\langle P \mid y = \langle \cdots \langle\langle x \mid x = M\rangle \mid D_1\rangle \cdots D_n\rangle, D\rangle] \\ & \twoheadrightarrow & C'[\langle P \mid y = x, x = M, D_1, \cdots, D_n, D\rangle] \\ & \rightarrow & C'[\langle P[y := x] \mid x = M[y := x], D_1[y := x], \cdots, D_n[y := x], D[y := x]\rangle] \ . \end{array}$$

The last terms of the two reduction sequences are α-equivalent and therefore considered the same.

□

LEMMA 4.7. $\mathcal{R}_0^{\rightarrow}$ *is confluent and terminating.*

PROOF. Termination follows from Theorem 7.4. A simple check verifies that the system is locally confluent. Confluence then follows from Newman's lemma [Bar84]. □

REMARK 4.8. The set of representational normal forms, *NF*, is the subset of cyclic terms that match the following specification:

$$\begin{array}{rcl} NF & ::= & x \mid \langle x \mid D'\rangle \\ D' & ::= & EQ \mid EQ, D' \\ EQ & ::= & x = x \mid x = y\ z \mid x = \lambda y.NF \end{array}$$

The restricted form of these normal forms gives rise to the following observations:

- The set of nodes of $\rho(M)$, for M in normal form, can be taken as the set of recursion variables, because every application, lambda and black hole has exactly one name.
- Given an equation $x = \lambda y.M$ the scope of x is precisely the set of recursion variables defined by the equation or any equation nested inside it. For example, given $x = \lambda y.\langle z \mid z = y\ y\rangle$ we have that $S(x) = \{x, z\}$.

In Section 8, we discuss an alternative rewriting system obtained by orienting the naming axiom from right to left.

4.2. Completeness

To show that the representational calculus contains sufficient axioms to equate all distinct representations of the same scoped graph, we introduce a mapping that associates a cyclic term to a scoped pre-graph:

$$\psi_{\text{pre}} : \text{scoped pre-graphs} \to \Lambda\circ \ .$$

This mapping is based on the scheme of translating every node to an equation ($x = x$ for a black hole, $x = y$ for an indirection node, $x = y\ z$ for an application node and $x = \lambda y.\langle z \mid D\rangle$ for a lambda-node) and placing these equations in such a way that every equation gets placed in a subterm of a lambda-abstraction corresponding to node v iff the equation came from a node in the scope of v except v itself.

Definition 4.9. Given a scoped pre-graph $g \equiv (V, L, A, S, r)$ we construct the cyclic term $\psi_{\text{pre}}(g)$ as follows: Let x_v be a variable for every node $v \in V$ and y_v be a variable for every lambda-node $v \in V$ such that they are all pairwise distinct. For every argument a we define N_a by distinguishing cases for a:

- a is a pointer to v: N_a is x_v
- a is a back-pointer to v: N_a is y_v
- a is the variable x: N_a is x

The term M_v is given by cases on the label of v.

- $L(v) = \bullet$: M_v is x_v if $A(v) = \epsilon$ and M_v is $N_{A(v)}$ otherwise
- $L(v) = \lambda$: M_v is $\lambda y_v.\langle N_{A(v)} \mid D_{S(v)\setminus\{v\}}\rangle$
- $L(v) = @$: M_v is $N_{A(v)_1}\ N_{A(v)_2}$

The set of equations D_W, for a set of node W, is given by

$$D_W = \{x_w = M_w \mid w \in W, \forall p \in W : w \in S(p) \Rightarrow w \equiv p\} \ .$$

$\psi_{\text{pre}}(g)$ is then defined as $\langle N_r \mid D_V\rangle$.

In the above definition the restriction imposed on D_W, *i.e.*, $\forall w, p \in W$, if $w \in S(p)$ then $w \equiv p$, guarantees that equations are put in the right list of declarations. For example, consider the right graph of Fig. 7. Let v_1 and v_2 be the left and right lambda-nodes, and v_3, v_4, v_5 be the top left, top right and bottom application nodes, respectively. Then,

$$D_{\{v_2,v_3,v_4,v_5\}} = \{x_3 = y_1 x_5, x_2 = \lambda y_2.\langle x_4 \mid x_4 = x_5 y_2\rangle, x_5 = zz\} \ .$$

The mappings ρ_{pre} and ψ_{pre} do not satisfy either $g = \rho_{\text{pre}}(\psi_{\text{pre}}(g))$ or $M \equiv \psi_{\text{pre}}(\rho_{\text{pre}}(M))$. For example, given $M \equiv xx$, $\psi_{\text{pre}}(\rho_{\text{pre}}(M))$ is

$$\langle x_1 \mid x_1 = x_2\ x_3, x_2 = x, x_3 = x\rangle \ .$$

And, given g as $\rho_{\text{pre}}(xx)$ (see the graph on the left of Fig. 19), $\rho_{\text{pre}}(\psi_{\text{pre}}(g))$ is the graph h on the right of Fig. 19. We do however have the following:

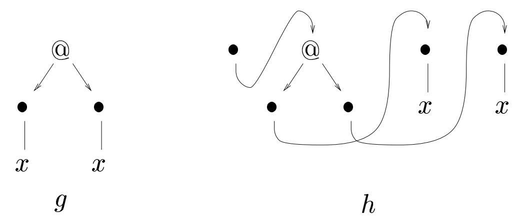


Figure 19. The result of applying ψ_{pre} and then ρ_{pre}

PROPOSITION 4.10. *Given a scoped pre-graph $g \equiv (V, L, A, S, r)$. Then $\rho_{\mathrm{pre}}(\psi_{\mathrm{pre}}(g))$ simplifies to g.*

PROOF. We map the graph g to a term M by ψ_{pre} and then map this term M to a graph h, by ρ_{pre}. We have to show that by applying simplification rules we can transform h into g. To show that this is possible we investigate the effects of ψ_{pre} and ρ_{pre}. The function ψ_{pre} maps nodes to equations and ρ_{pre} maps symbols to nodes. Since ρ_{pre} has a recursive definition and always produces a graph with a node as the root we can assume that for every equation $x_v = M_v$ the root of the graph of M_v is the node n_{x_v}. We will show that it is possible to simplify h to a graph that contains only the nodes $\{n_{x_v} \mid v \in V\}$ and such that $v \mapsto n_{x_v}$ is an isomorphism.

We will show this by considering every node $v \in V$. We will not distinguish between normal and back pointers explicitly. A simple analysis shows that this tagging of pointers is preserved. There are four cases for v:

- If v is a black hole it gets mapped to the equation $x_v = x_v$. The x_v in the right-hand side gets mapped to an indirection node n_{x_v}, that refers to itself. We may simplify this to n_{x_v} being a black hole.

- If v is an indirection node referring to w then it gets mapped to an equation $x_v = x_w$. The symbol x_w gets mapped to the indirection node n_{x_v} referring to the node n_{x_w}. No simplification is necessary.

- If v is an application node with arguments v_1 and v_2 then v gets mapped to an equation $x_v = x_{v_1}\ x_{v_2}$. The application gets mapped to the node n_{x_v}, the symbols x_{v_1} and x_{v_2} get mapped to indirection nodes referring to $n_{x_{v_1}}$ and $n_{x_{v_2}}$, respectively. These indirection nodes are the arguments of n_{x_v} and this situation simplifies to n_{x_v} having $n_{x_{v_1}}$ and $n_{x_{v_2}}$ as its arguments directly.

- If v is a lambda node with argument w then it gets mapped to the equation $x_v = \lambda y_v.\langle x_w \mid D_{S(v)\setminus\{v\}}\rangle$. The lambda gets mapped to the lambda node n_{x_v}, with the image of the x_w as its argument. The x_w gets mapped to an indirection node referring to n_{x_w}. This simplifies to n_{x_v} having n_{x_w} as its argument directly. All nodes in the scope of v get mapped to equations nested in the equation for v, which in turn get mapped to nodes in the scope of n_{x_v}.

□

An important corollary from this proposition is that every scoped graph can be represented.

COROLLARY 4.11. *Given a scoped graph g. Then $\psi_{\mathrm{pre}}(g)$ represents g.*

It is then sensible to define:

DEFINITION 4.12. The canonical representation of a scoped pre-graph g is the normal form of $\psi_{\mathrm{pre}}(g)$ with respect to $\mathcal{R}_0^{\rightarrow}$.

Note that the only case in which the representation of a scoped graph is not in normal form is when there is a lambda-node that has no nodes in the scope except itself, since the term associated to such a node will be of the form $\lambda y.\langle z \mid \rangle$ which must be rewritten to $\lambda y.z$.

LEMMA 4.13. *Given $M \in \Lambda\circ$. If M is in normal form with respect to $\mathcal{R}_0^{\rightarrow}$ then $\mathcal{R}_0 \vdash M = \psi_{\text{pre}}(\rho_{\text{pre}}(M))$.*

PROOF. By structural induction on M. According to Remark 4.8 we have the following two cases:

- M is a variable x. Then $\psi_{\text{pre}}(\rho_{\text{pre}}(x)) = \langle x_1 \mid x_1 = x\rangle$, which by application of the variable substitution and empty box garbage collection rules reduces to x.

- M is $\langle x \mid D'\rangle$. By cases on the type of equation in D':

  $x = x$. This equation gets mapped to an indirection node having itself as argument. This gets mapped back to $x = x$.

  $x = yz$. In the pre-graph this corresponds to an application node labeled with x and two indirection nodes pointing to y and z, respectively. When we translate back we get $x = x_1\ x_2, x_1 = y, x_2 = z$. Which reduces to $x = y\ z$ with two applications of variable substitution.

  $x = \lambda y.N$, where N is in normal form. Follows from the induction hypothesis.

□

LEMMA 4.14. *Given a scoped pre-graph g. Then $\mathcal{R}_0 \vdash \psi_{\text{pre}}(g) = \psi_{\text{pre}}(\text{Sim}(g))$.*

PROOF. From Fig. 13 it is clear that the application of the first rule does not change ψ_{pre}, and the application of the last two rules corresponds to the variable substitution axiom. □

We can now prove that $\mathcal{R}_0$ is complete with respect to scoped graphs.

THEOREM 4.15. *Given $M, N \in \Lambda\circ$. If $\rho(M) = \rho(N)$ then $\mathcal{R}_0 \vdash M = N$.*

PROOF. This follows trivially from the claim that if M is a representation of the scoped graph g then $\mathcal{R}_0 \vdash M = \psi_{\text{pre}}(g)$.

To prove the claim let M' be the normal form of M with respect to $\mathcal{R}_0^{\rightarrow}$. Obviously $\mathcal{R}_0 \vdash M = M'$. From the soundness of $\mathcal{R}_0$ (Theorem 4.2) we know that M' represents g and therefore that the simplification of $\rho_{\text{pre}}(M')$ is g. From Lemma 4.13 we know that $\mathcal{R}_0 \vdash M' = \psi_{\text{pre}}(\rho_{\text{pre}}(M'))$. Applying Lemma 4.14 we then have $\mathcal{R}_0 \vdash \psi_{\text{pre}}(\rho_{\text{pre}}(M')) = \psi_{\text{pre}}(g)$ and so:

$$\mathcal{R}_0 \vdash M = M' = \psi_{\text{pre}}(\rho_{\text{pre}}(M')) = \psi_{\text{pre}}(g)\ .$$

□

COROLLARY 4.16. *Given a scoped graph g. The canonical representation of g is the normal form of any representation of g.*

PROOF. Since $\mathcal{R}_0$ is complete (Theorem 4.15) we know that every two representations, say M and N, are provably equal. By Proposition 4.6, M and N are convertible with respect to $\mathcal{R}_0^{\rightarrow}$. Since the rewriting system is confluent and terminating (Lemma 4.7) it follows that M and N have the same normal form. This unique normal form is precisely the canonical representation of graph g. □

This corollary implies that the set of scoped graphs and the set of $\mathcal{R}_0^{\rightarrow}$-normal forms are isomorphic.

5. Complete axiomatization of well-formed lambda-graphs

We have now established how scoped graphs are represented by cyclic terms. Since a well-formed cyclic graph can have different scoped graphs associated with it, our next goal is to find axioms that equate the representations of these alternatively scoped graphs.

DEFINITION 5.1. A scoped graph h is an alternatively scoped version of a graph $g \equiv (V, L, A, S, r)$ (written as $h \sim g$) if $h \equiv (V, L, A, S', r)$.

For example, the first two graphs of Fig. 7 are alternatively scoped versions of the same graph (*i.e.,* the fourth graph of Fig. 5). These two graphs are represented by the following terms:

$$\langle x \mid x = \lambda y.\langle w \mid w = w_1 y\rangle, w_1 = zz\rangle \ ,$$

and

$$\langle x \mid x = \lambda y.\langle w \mid w = w_1 y, w_1 = zz\rangle\rangle \ .$$

To equate the above two terms we extend the representational calculus with the following lambda lift axiom:

$$\lambda x.\langle M \mid D\rangle = \langle \lambda x.M \mid D\rangle \quad x \text{ not free in } D \ .$$

We then have:

$$\begin{array}{rcll} \langle x \mid x = \lambda y.\langle w \mid w = w_1 y\rangle, w_1 = zz\rangle & = & \langle x \mid x = \langle \lambda y.\langle w \mid w = w_1 y\rangle | w_1 = zz\rangle\rangle & \text{internal merge} \\ & = & \langle x \mid x = \lambda y.\langle\langle w \mid w = w_1 y\rangle | w_1 = zz\rangle\rangle & \text{lambda lift} \\ & = & \langle x \mid x = \lambda y.\langle w \mid w = w_1 y, w_1 = zz\rangle\rangle & \text{external merge} \end{array}$$

However, on graphs with garbage, the lambda lift axiom is not necessarily powerful enough. Consider the following two terms (displayed in Fig. 11),

$$\langle z \mid x_0 = \lambda y_0.\langle x \mid x_1 = \lambda y_1.\langle y \mid x_2 = y_0\ y_1\rangle\rangle\rangle \text{ and } \langle z \mid x_1 = \lambda y_1.\langle y \mid x_0 = \lambda y_0.\langle x \mid x_2 = y_0\ y_1\rangle\rangle\rangle \ .$$

It is impossible to prove these terms equal using the lambda lift axiom because that would require lifting the inner lambda out of the scope of the outer lambda, which is impossible because of the garbage application node which has to be inside the scopes of both lambda's. Restricted to garbage free graphs $\mathcal{R}_0$ combined with the lambda lift axiom is sound and complete for alternatively scoped graphs, that is for scoped graphs that have a unique minimal scope function. We leave the problem of finding a sound and complete axiom system for all graphs with garbage open. Instead, we study graphs up to alternative scoping and garbage collection. Thus, we introduce the operation of garbage collection which removes nodes that are not reachable from the root. We denote the restriction of function L to the set V by $L\uparrow_V$.

DEFINITION 5.2. Given a scoped graph $g \equiv (V, L, A, S, r)$ and a subgraph W of g such that (i) $r \notin W$, (ii) for every node v that has a node in W as argument, $v \in W$, (iii) for every lambda-node $v \in W$, $S(v) \subseteq W$. Then, the result of garbage collecting W from g is defined as:

$$\mathrm{gc}(g, W) = (V \setminus W, L\uparrow_{V\setminus W}, A\uparrow_{V\setminus W}, S', r) \ ,$$

where $S'(v) = S(v) \cap (V \setminus W)$ for every lambda-node $v \in V \setminus W$.

For example, in Fig. 20 we have that for every $1 \le i \le j \le 4$ there exists a set W such that $\mathrm{gc}(g_i, W) = g_j$. Notation: if the set W represents all nodes not reachable from the root then we let $\mathrm{gc}(g) = \mathrm{gc}(g, W)$. Referring back to Fig. 20 we have that $\mathrm{gc}(g_i) = g_4, i = 1, 2, 3$.

Garbage collection is axiomatized by the following axiom:

$$\langle M \mid D\rangle = M \qquad\qquad D \perp M \ ,$$

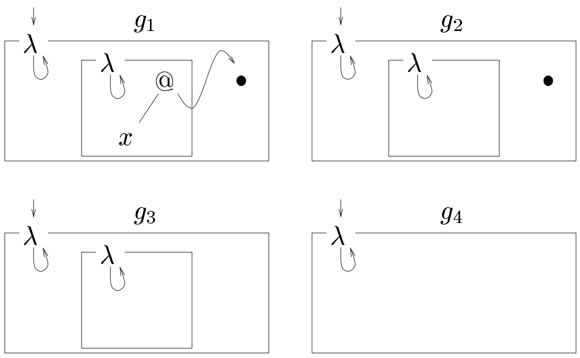


Figure 20. Garbage collection

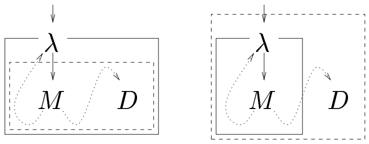


Figure 21. Pictorial description of the lambda lift axiom

where $D \perp M$ means that the set of variables that occur as the left-hand side of an equation in D does not intersect with the set of free variables of M. We will also adopt the notation $D \perp D'$, where D' is $x_1 = M_1, \cdots, x_n = M_n$, which stands for $D \perp M_i, 1 \leq i \leq n$.

THEOREM 5.3. *Given $M, N \in \Lambda\circ$.*

(i) *If $M = N$ by the lambda lift axiom then $\rho(M) \sim \rho(N)$.*

(ii) *If $M = N$ by the garbage collection axiom then there exists a set W such that* $\mathrm{gc}(\rho(M), W) = \rho(N)$.

PROOF. Follows trivially from the pictorial descriptions of the new axioms given in Fig. 21 and 22, respectively. The dashed boxes indicate the presence of letrec constructs. □

We call $\mathcal{R}_1$ the representational calculus extended with the lambda lift and garbage collection axioms. Its effects on the represented graphs can be stated as follows:

PROPOSITION 5.4. *Given $M, N \in \Lambda\circ$. If $\mathcal{R}_1 \vdash M = N$ then* $\mathrm{gc}(\rho(M)) \sim \mathrm{gc}(\rho(N))$.

PROOF. We know that $\sim$ is an equivalence relation. Hence the result follows from proving that the result holds for a single application of each axiom. This single axiom can be

Figure 22. Pictorial description of the garbage collection axiom

- An axiom from $\mathcal{R}_0$: By Theorem 4.2 we have that $\rho(M) = \rho(N)$.
- The lambda lift axiom: By Theorem 5.3(i) we have that $\rho(M) \sim \rho(N)$. It is then not hard to see that $\mathrm{gc}(\rho(M)) \sim \mathrm{gc}(\rho(N))$.
- The garbage collection axiom: By Theorem 5.3(ii) we know that there exists a set W such that $\mathrm{gc}(\rho(M), W) = \rho(N)$. It is then obvious that $\mathrm{gc}(\rho(M)) = \mathrm{gc}(\rho(N))$.

□

In order to show completeness of $\mathcal{R}_1$ we first study the associated rewriting system $\mathcal{R}_1^{\rightarrow}$, which is obtained by extending $\mathcal{R}_0^{\rightarrow}$ with a rule for the lambda lift axiom and one for garbage collection. The rewriting rule associated with the lambda lift axiom is:

$$\lambda x.\langle M \mid D_1, D_2\rangle \rightarrow \langle \lambda x.\langle M \mid D_1\rangle \mid D_2\rangle \quad D_1 \perp D_2, x \text{ not free in } D_2 \text{ and } D_2 \neq \{\ \}\ . \tag{5.1}$$

We need to introduce this more general rule to guarantee confluence of $\mathcal{R}_1^{\rightarrow}$. By just orienting the lambda lift axiom from left to right we would not be able to close the following diagram:

$$\begin{tikzcd}
\lambda x.\langle\langle y \mid y = z\ x\rangle \mid z = z\rangle \arrow[r] \arrow[d] & \langle \lambda x.\langle y \mid y = z\ x\rangle \mid z = z\rangle \\
\lambda x.\langle y \mid y = z\ x, z = z\rangle &
\end{tikzcd}$$

With the rule given in 5.1 we can rewrite the bottom left term to the top right term. Rule 5.1 is a combination of the external merge axiom and the lambda lift axiom:

$$\lambda x.\langle M \mid D_1, D_2\rangle = \lambda x.\langle\langle M \mid D_1\rangle \mid D_2\rangle = \langle \lambda x.\langle M \mid D_1\rangle \mid D_2\rangle\ .$$

For the same reason, we introduce a more general form of the garbage collection rule, which allows us to remove subsets of equations:

$$\langle M \mid D_1, D_2\rangle \rightarrow \langle M \mid D_1\rangle \quad D_2 \perp \langle M \mid D_1\rangle \text{ and } D_2 \neq \{\ \}\ .$$

Obviously, the provable equality relation of $\mathcal{R}_1$ is the same as the convertibility relation of $\mathcal{R}_1^{\rightarrow}$.

PROPOSITION 5.5. *$\mathcal{R}_1^{\rightarrow}$ is confluent and terminating.*

PROOF. Termination is proven in Theorem 7.4. Confluence then follows from local confluence and Newman's lemma. □

We can now prove that the axiomatization of garbage collection is complete and that $\mathcal{R}_1$ is complete with respect to well-formed graphs.

THEOREM 5.6. *Given $M, N \in \Lambda\circ$.*

(i) *If $\mathrm{gc}(\rho(M)) = \mathrm{gc}(\rho(N))$ then $\mathcal{R}_0 \cup \mathrm{gc} \vdash M = N$.*
(ii) *If $\rho(M) \sim \rho(N)$ then $\mathcal{R}_1 \vdash M = N$.*

PROOF.

(i) By a simple induction on the nesting of scopes one can prove that for every cyclic term P we have that $\mathcal{R}_0 \cup \mathrm{gc} \vdash \psi_{\mathrm{pre}}(\rho(P)) = \psi_{\mathrm{pre}}(\mathrm{gc}(\rho(P)))$. So we have that $\mathcal{R}_0 \cup \mathrm{gc} \vdash \psi_{\mathrm{pre}}(\rho(M)) = \psi_{\mathrm{pre}}(\rho(N))$. By Corollary 4.11 and Theorem 4.15 we may conclude that $\mathcal{R}_0 \cup \mathrm{gc} \vdash M = N$.

(ii) Let M_1 and N_1 be representations of $\mathrm{gc}(\rho(M))$ and $\mathrm{gc}(\rho(N))$, respectively. By the previous point we can prove $M = M_1$ and $N = N_1$. We still have that $\rho(M_1) \sim \rho(N_1)$. Let M_2 and N_2 be the $\mathcal{R}_1^{\rightarrow}$ normal forms of M_1 and N_1, respectively. It is easy to prove that M_2 and N_2 are minimally scoped. Because by Proposition 2.7 the minimally scoped graph is unique M_2 and N_2 must represent the same graph. Because M_2 and N_2 are also $\mathcal{R}_0^{\rightarrow}$-normal forms by Corollary 4.16 we have that $M_2 \equiv N_2$.

□

Next, we study the relationship between garbage collection on graphs and garbage collection on the representation. We know that if $M \xrightarrow[\mathrm{gc}]{} N$ then there exists a set W such that $\mathrm{gc}(\rho(M), W) = \rho(N)$. The other direction is more subtle. A simple induction on the structure of M shows that if M represents a scoped graph g then the garbage collection normal form of M represents $\mathrm{gc}(g)$. However, if $\mathrm{gc}(g, W)$ is well defined for some set W then it does not imply that we can find a term N such that N represents $\mathrm{gc}(g, W)$ and $M \xrightarrow[\mathrm{gc}]{} \!\!\!\!\! \rightarrow N$. For example, in the graph $\rho(\langle x \mid u = x\ \langle x \mid w = y\ y \rangle\rangle)$ we can garbage collect $x\ x$. In the term we first have to rewrite by lift and merge to $\langle x \mid u = x\ x, w = y\ y\rangle$, before we can garbage collect. The problem is that other garbage nodes are caught inside the garbage we want to remove. We can always free this garbage using lift and merge:

THEOREM 5.7. *Given a scoped graph g and a subgraph W of g that we can garbage collect. If M represents g then there exists N, $M \xrightarrow[\mathcal{R}_0 \cup \mathrm{gc}]{} \!\!\!\!\! \rightarrow N$ and N represents* $\mathrm{gc}(g, W)$.

PROOF. We first rewrite M to its $\mathcal{R}_0^{\rightarrow}$ normal from M'. A simple structural induction on M' then shows that we may rewrite M' using only the garbage collection rule to a term N representing $\mathrm{gc}(g, W)$. □

6. Sound and complete axiomatization of tree unwinding

We now want to prove equal every two representations of graphs with the same tree unwinding. Since these unwindings are in general infinite, in this section we assume that all graphs are possibly infinite, except when stated otherwise. When we say a graph can be represented it is implied that that graph is finite, since only finite graphs can be represented. A tree is a lambda-graph where every node is referenced by a normal pointer exactly once. So there is one pointer to every lambda-node and zero or more back-pointers. As usual the set of nodes of the tree unwinding is defined as the set of all paths starting at the root. Usually these paths are represented by strings of integers. Here, for convenience, we use a more verbose representation which keeps track of the nodes visited. For example, consider the following graph g (depicted on the left of Fig. 23):

$$\begin{array}{ll} (& \{v_1, v_2, v_3\}, \\ & \{v_1 \mapsto @, v_2 \mapsto \lambda, v_3 \mapsto @\}, \\ & \{v_1 \mapsto v_2\ v_2, v_2 \mapsto v_3, v_3 \mapsto \overline{v_2}\ \overline{v_2}\}, \\ & v_1 \\ ) \ . & \end{array} \tag{6.1}$$

The nodes of the unwinding of g (see the graph on the right of Fig. 23) are

$$\{v_1, v_1 1 v_2, v_1 1 v_2 1 v_3, v_1 2 v_2, v_1 2 v_2 1 v_3\} \ ,$$

instead of $\{\epsilon, 1, 11, 2, 21\}$.

DEFINITION 6.1. The unwinding of a graph $g \equiv (V, L, A, r)$ is a tree $g_u \equiv (V_u, L_u, A_u, r_u)$:

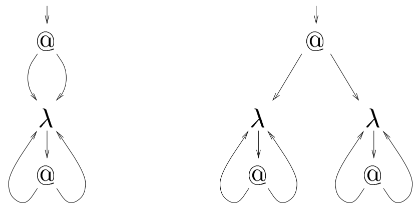


Figure 23. The unwinding of an acyclic graph

- $V_u = \{r \mid r \in V\} \cup \{p\ v\ i\ w \mid p\ v \in V_u, A(v)_i = w, w \in V\}$.
- $L_u(p\ v) = L(v)$.
- $A_u(p\ v)_n = \begin{cases} p\ v\ n\ w, \text{ if } A(v)_n = w \\ \overline{p_1 w} \quad\ , \text{ if } A(v)_n = \overline{w} \text{ and } p \equiv p_1\ w\ p_2 \text{ where } w \text{ does not occur in } p_2 \\ x \qquad\ , \text{ if } A(v)_n = x \end{cases}$
- $r_u = r$.

Note that the unwinding of a graph only depends on the accessible part, as in the first order case.

EXAMPLE 6.2. The unwinding of graph g given in equation 6.1 and displayed in Fig. 23 is given below.

$$\begin{array}{rcl} V_u & = & \{v_1, v_1 1 v_2, v_1 1 v_2 1 v_3, v_1 2 v_2, v_1 2 v_2 1 v_3\} \\ L_u & = & \{v_1 \mapsto @, v_1 1 v_2 \mapsto \lambda, v_1 1 v_2 1 v_3 \mapsto @, v_1 2 v_2 \mapsto \lambda, v_1 2 v_2 1 v_3 \mapsto @\} \\ A_u & = & \{ \begin{array}{rcl} v_1 & \mapsto & v_1 1 v_2\ v_1 2 v_2, \\ v_1 1 v_2 & \mapsto & v_1 1 v_2 1 v_3, \\ v_1 1 v_2 1 v_3 & \mapsto & \overline{v_1 1 v_2}\ \overline{v_1 1 v_2}, \\ v_1 2 v_2 & \mapsto & v_1 2 v_2 1 v_3, \\ v_1 2 v_2 1 v_3 & \mapsto & \overline{v_1 2 v_2}\ \overline{v_1 2 v_2} \end{array} \\ & & \} \\ r_u & = & v_1 \ . \end{array}$$

The $\mathcal{R}_1$ representational calculus is sound with respect to tree unwinding. To guarantee completeness we need another axiom, as was already pointed out in [AK94] for the first-order case. This axiom is the copy axiom, defined as:

$$M \quad = \quad N \quad \exists \sigma : \mathcal{V} \to \mathcal{V}, N^\sigma \equiv M \ ,$$

where σ is a function from recursion variables to recursion variables, and N^σ is the term obtained by replacing all occurrences of recursion variables x by $\sigma(x)$ (leaving the free variables of N unchanged), followed by a reduction to normal form with the unification rule:

$$x = M, x = M \to x = M \ . \tag{6.2}$$

EXAMPLE 6.3. The following equality is an example of the copy axiom:

$$\langle y \mid y = \lambda z.w, w = \lambda x.y \rangle = \langle y \mid y = \lambda z.w', w' = \lambda x.y', y' = \lambda z.w' \rangle \ ,$$

where the mapping σ is: $w' \mapsto w$, $y \mapsto y$ and $y' \mapsto y$. In this case, N^σ is

$$\langle y \mid y = \lambda z.w, w = \lambda x.y, y = \lambda z.w\rangle \ ,$$

which reduces to $\langle y \mid y = \lambda z.w, w = \lambda x.y\rangle$ by application of rule 6.2.

We let $\mathcal{R}_2$ denote $\mathcal{R}_1$ extended with the copy axiom. We denote by $\xrightarrow[\text{cp}]{}$ the rewriting rule obtained by reading the copy axiom from left to right.

In order to ease the relation between terms and possibly infinite trees we introduce next the notion of homomorphism on scoped graphs.

6.1. Homomorphisms

A homomorphism between two scoped graphs $(V_1, L_1, A_1, S_1, r_1)$ and $(V_2, L_2, A_2, S_2, r_2)$ is a function f from V_1 to V_2. We extend the application of function f to sets in the standard way. The extension to arguments is given by $\forall x \in \mathcal{V} : f(x) = x$ and $\forall v \in V : f(\overline{v}) = \overline{f(v)}$. The extension of function f to strings of arguments is again the standard (point-wise) extension. Among the natural conditions for a homomorphism f are:

$$\forall v \in V_1 : L_1(v) = L_2(f(v)) \ ,$$
$$\forall v \in V_1 : f(A_1(v)) = A_2(f(v))$$

and

$$f(r_1) = r_2 \ .$$

The last condition indicates that we are interested in a rooted homomorphism. For first-order graphs these conditions are sufficient to guarantee that two homomorphic graphs have the same tree unwinding. The presence of bound variables makes the matter more complicated, as shown next.

Consider the graphs of Fig. 24. They are represented by the following terms:

$$\begin{array}{lcl}
M_1 & \equiv & \langle nat \mid nat = \lambda x.cons\ (nat'\ (S\ x))\ x, nat' = \lambda x.cons\ (nat\ (S\ x))\ x\rangle \\
M_2 & \equiv & \langle nat \mid nat = \lambda x.cons\ (nat\ (S\ x))\ x\rangle \\
M_3 & \equiv & \langle nat \mid nat = \lambda x.\langle cons\ (nat'\ (S\ x))\ x \mid nat' = \lambda y.cons\ (nat'\ (S\ x))\ y\rangle\rangle \ .
\end{array}$$

According to the conditions for a homomorphism stated so far, there exist homomorphisms from both the left and right graph to the graph in the middle. However, while the first two graphs have the same tree unwinding, this is not so for the middle and right graph. This entails the introduction of another condition. Before stating this condition we introduce the belongs to relation.

Definition 6.4. A node v belongs to a lambda-node w if

(i) $v \not\equiv w$,

(ii) $v \in S(w)$,

(iii) for every u such that $v \in S(u)$ we have that $S(w) \subseteq S(u)$.

If a node does not belong to any lambda-node we say that it belongs to a special symbol called *root*.

For example, referring to the right graph of Fig. 24, we have that the lower lambda-node belongs to the lambda-node on the top, which in turn belongs to the *root*.

Remark 6.5. In addition to the observations made in Remark 4.8, we also note that if the $\mathcal{R}_0$-normal form of a term M is of the form

$$\langle x_1 \mid x_1 = M_1, \cdots, x_n = M_n\rangle$$

then exactly $x_1, \cdots, x_n$ are the nodes belonging to the *root*. And if M contains an equation of the form $z = \lambda y.\langle z_1 \mid z_1 = M_1, \cdots, z_n = M_n\rangle$ then exactly $z_1, \cdots, z_n$ are the nodes belonging to z. There is no other way a node can belong to some other node (or to the *root*).

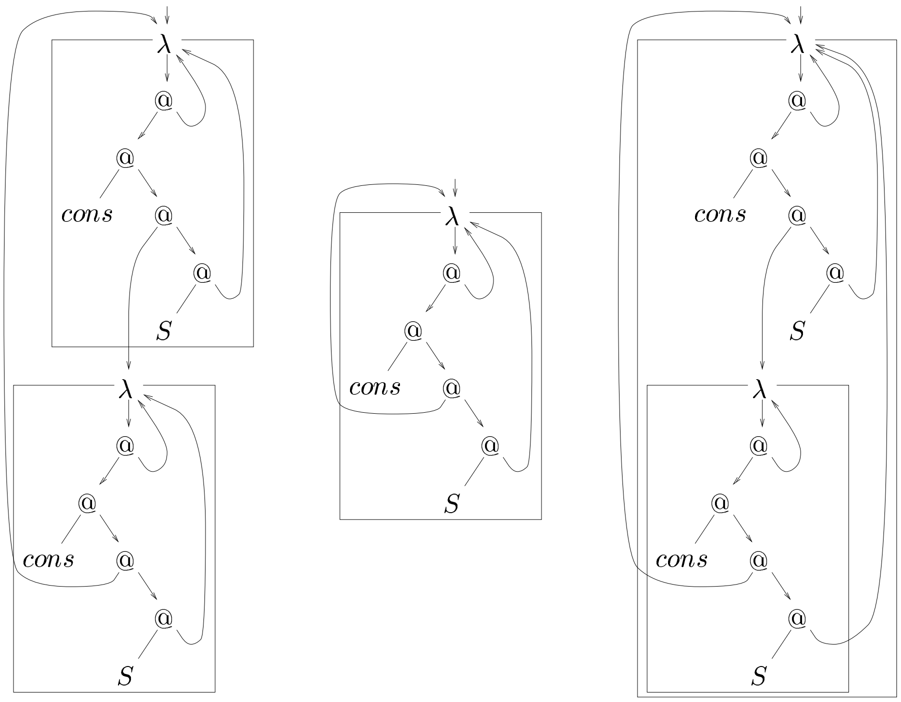


Figure 24. Why restricting the scope function is not enough

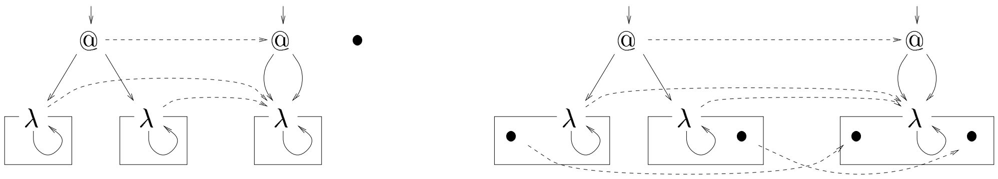


Figure 25. Homomorphism and garbage

We can now state our next requirement for a homomorphism f as follows:

- $\forall v \in V_1$: If v belongs to w then $f(v)$ belongs to $f(w)$.

Because a node may belong to the *root* we define $f(root) = root$. Referring back to Fig. 24, this new condition disallows the graph on the right to be homomorphic to the middle graph, since if we call v_1 and v_2 the upper and lower lambda-nodes, respectively, then we have: v_2 belongs to v_1 but $f(v_2)$ belongs to *root* which is different from $f(v_1)$.

If we only consider garbage free graphs then the notion of a homomorphism corresponds to copying the corresponding canonical representations. This is not so for graphs with garbage, as it can be shown by considering Fig. 25. This figure contains two examples. In both cases the dashed arrows define a function which satisfies all conditions for homomorphisms, stated so far. However, the corresponding terms are not in the copying relation. In the left example the left graph is not a copy of the right graph, because the garbage node gets deleted. The same holds for the right example in a more subtle way: if there is garbage in a scope then this garbage is present in all copies.

To maintain the correspondence between a homomorphism on graphs and copying on terms we impose the following conditions:

$$\begin{array}{l}\forall v \in V_1 : L(v) = \lambda \Rightarrow f(S_1(v)) = S_2(f(v)) \\ f(V_1) = V_2 \ .\end{array}$$

We summarize the discussion so far with the following definition.

DEFINITION 6.6. Given two scoped graphs $(V_1, L_1, A_1, S_1, r_1)$ and $(V_2, L_2, A_2, S_2, r_2)$. A function $f : V_1 \to V_2$ is a homomorphism if:

- $f(V_1) = V_2$
- $\forall v \in V_1 : L_1(v) = L_2(f(v))$
- $\forall v \in V_1 : f(A_1(v)) = A_2(f(v))$
- $\forall v \in V_1 : L(v) = \lambda \Rightarrow f(S_1(v)) = S_2(f(v))$
- $f(r_1) = r_2$
- $\forall v \in V_1$: If v belongs to w then $f(v)$ belongs to $f(w)$

The following common properties of homomorphisms hold:

PROPOSITION 6.7.

(i) *Given scoped graphs g, h. If $f : g \to h$ is a homomorphism then $f : \mathrm{gc}(g) \to \mathrm{gc}(h)$ is also a homomorphism.*

(ii) *Given scoped graphs g_1, g_2, g_3. If $f_1 : g_1 \to g_2$ and $f_2 : g_2 \to g_3$ are homomorphisms then $f_2 \circ f_1 : g_1 \to g_3$ is also a homomorphism.*

PROOF. Trivial. □

In the proofs of soundness and completeness we establish a homomorphism from the unwinding to the original graph. This can only be done if the unwinding has a scope function. The scope function we need is given by:

DEFINITION 6.8. Given a scoped graph $g \equiv (V, L, A, S, r)$. The scoped unwinding g_u^s of g is the unwinding g_u extended with the scope function:

$$S(p\ v) = \{p\ v\} \cup \{p\ v\ i\ A(w)_i \mid p\ w \in S(p\ v), A(w)_i \in S(v) \setminus v\} \ .$$

In Fig. 26 we have drawn a cyclic graph and its scoped unwinding. Note that the scopes of the lambda-nodes are not nested.

LEMMA 6.9. *Given a scoped graph g and the scoped unwinding g_u^s of g. The function $f : p\ v \mapsto v$ defines a homomorphism from g_u^s to $\mathrm{gc}(g)$.*

PROOF. All conditions except the condition on belonging to are trivial. Suppose $p_1\ v$ belongs to $p_2\ w$. Then by definition of belongs to we know that $p_1\ v \in S(p_2\ w) \setminus \{p_2\ w\}$. This implies that there is a path from $p_1\ v$ to $p_2\ w$ that does not include any lambda-nodes. In turn this implies that there is a path from w to v without including any lambda-nodes. Let us assume v does not belong to w. This means there exists a w' such that $v \in S(w') \setminus \{w'\}$ and $w' \in S(w) \setminus \{w\}$, which contradicts Proposition 2.6, since there is a path in $\mathrm{gc}(g)$ to v that does not go through the corresponding lambda-node. □

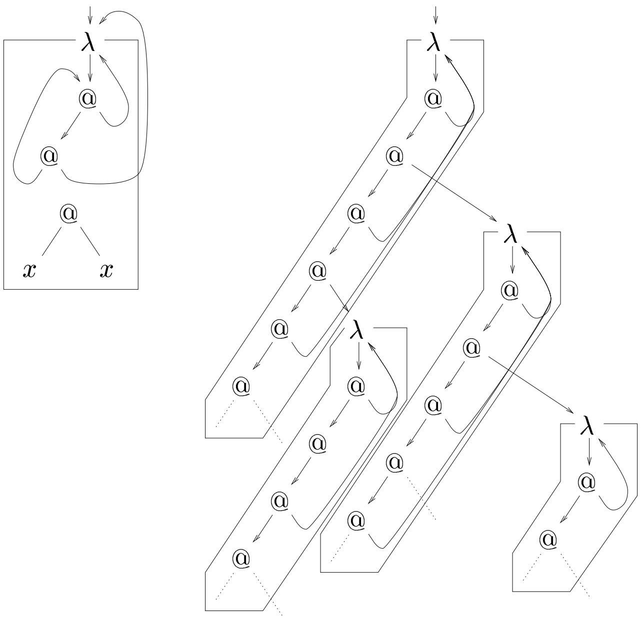


Figure 26. The unwinding of a cyclic graph

6.2. Soundness

Soundness with respect to tree unwinding depends on two lemmas. The first relates copying to a homomorphism and the second relates a homomorphism to unwinding.

LEMMA 6.10. *Given $M, N \in \Lambda\circ$. If $M \xrightarrow[\text{cp}]{} N$ then there exists a homomorphism*

$$f : \rho(N) \to \rho(M) \ \ .$$

PROOF. By definition of copy we know there exists a variable mapping σ such that $N^\sigma \equiv M$. We reason by cases.

- M is in normal form with respect to $\mathcal{R}_0^{\rightarrow}$. This means that N must also be in normal form. Let us assume M is not a variable, then by Remark 4.8 the sets of nodes of $\rho(M)$ and $\rho(N)$ can be taken as the sets of variables that occur on the left-hand sides of equations. If we do so then the restriction of σ to the set of variables occurring on the left-hand side of an equation in N is a homomorphism between $\rho(N)$ and $\rho(M)$, which can be easily verified.
- M is not in normal form with respect to $\mathcal{R}_0^{\rightarrow}$. This means that N cannot be in normal form either. To reduce this case to the previous one we are going to prove that the following diagram holds:

$$\begin{tikzcd} M \arrow[r, "\text{cp}"'] \arrow[d, twoheadrightarrow] & N \arrow[d, twoheadrightarrow] \\ P \arrow[r, dashed, "\text{cp}"'] & Q \end{tikzcd} \tag{6.3}$$

where the vertical arrows denote a reduction to normal form with respect to $\mathcal{R}_0^{\rightarrow}$. We then have a homomorphism $f : \rho(Q) \to \rho(P)$, which of course is also a homomorphism $f : \rho(N) \to \rho(M)$, as required.

To show diagram 6.3 we take a closer look at the computation of N^σ. This computation is of the form

$$N \xrightarrow[\sigma]{} N_1 \xrightarrow[\text{unif}]{}\!\!\!\!\!\rightarrow N^\sigma \quad ,$$

where the first step is the syntactic substitution and the other steps are the reduction to normal form with respect to unification rule (see equation 6.2). Note that if a representational rule is applicable in N then it is also applicable in N^σ and vice versa.

We obtain diagram 6.3 by showing the existence of diagrams of the form:

$$\begin{tikzcd}
{} \arrow[r, "\sigma"] \arrow[d, dashed, two heads] & {} \arrow[r, two heads, "\text{unif}"] \arrow[d, dashed, two heads] & {} \arrow[d, dashed, two heads] \\
{} \arrow[r, dashed, "\sigma"'] & {} \arrow[r, dashed, two heads, "\text{unif}"'] & {}
\end{tikzcd} \tag{6.4}$$

where in the vertical direction we have one or more steps in the representational rewriting system $\mathcal{R}_0^{\rightarrow}$.

We now reason by cases.

- A lift, empty box garbage collection, merge rule is applicable. We take the following steps:
  - We take a reduction to normal form with respect to the lift, empty box and merge rules. Since these redexes do not depend on the names of variables, we can easily fill in the left square of diagram 6.4.
  - We then proceed by tiling the right square. To make the tiling process work, we use lift, empty box and merge steps in the vertical direction and lift, empty box, merge and unification steps in the horizontal direction. We have only the following two non-trivial diagrams with a rewriting step other than a unification step in the horizontal direction. We have the generic pattern:

$$\begin{tikzcd}
x = M, x = M \arrow[rr, "\text{unif}"] \arrow[d] & & x = M \arrow[d, dashed] \\
x = M', x = M \arrow[r, dashed] & x = M', x = M' \arrow[r, dashed, "\text{unif}"'] & x = M'
\end{tikzcd}$$

    and the specific critical pair:

$$\begin{tikzcd}
x = \langle M \mid D\rangle, x = \langle M \mid D\rangle \arrow[rr, "\text{unif}"] \arrow[d] & & x = \langle M \mid D\rangle \arrow[d, dashed] \\
x = M, D, x = \langle M \mid D\rangle \arrow[r, dashed] & x = M, D, x = M, D \arrow[r, dashed, two heads, "\text{unif}"'] & x = M, D
\end{tikzcd}$$

    If we start from a term in normal form with respect to the lift, empty box and merge rules then the bottom reduction will have only unification steps, since a unification step can only create new unification redexes. Termination of the tiling process follows simply from the fact that the rewriting system containing lift, empty box, merge and unification is terminating.
- The naming rule is applicable. We will prove a special case of diagram 6.4:

$$\begin{tikzcd}
{} \arrow[r, "\sigma"] \arrow[d, dashed, two heads] & {} \arrow[r, two heads, "\text{unif}"] \arrow[d, dashed, two heads] & {} \arrow[d] \\
{} \arrow[r, dashed, "\sigma"'] & {} \arrow[r, dashed, two heads, "\text{unif}"'] & {}
\end{tikzcd} \tag{6.5}$$

  We work from right to left this time. The steps are:

- We can complete the right square by tiling. There is only one non-trivial elementary diagram:

$$\begin{tikzcd}
x = C[M], x = C[M] \arrow[r, "\mathrm{unif}"] \arrow[d, dashed] & x = C[M] \arrow[dd] \\
x = C[\langle y \mid y = M\rangle], x = C[M] \arrow[d, dashed] & \\
x = C[\langle y \mid y = M\rangle], x = C[\langle y \mid y = M\rangle] \arrow[r, dashed, "\mathrm{unif}"'] & x = C[\langle y \mid y = M\rangle]
\end{tikzcd}$$

Note that the new name is the same in every naming step. This is necessary to be able to do the unification step on the bottom. Thus we close the right square.

- The left square has n naming steps on the right. All of these steps use the same variable. To get a legal sequence on the left we must choose n new variables, extend the variable mapping by mapping each of the new variables to the variable used in the naming redexes.

- The variable substitution rule is applicable. We prove the diagram 6.4 as in the previous case, again working from right to left.
  - First we tile the right square. The non-trivial tiles needed are:

$$\begin{tikzcd}
x = M, x = M \arrow[r, "\mathrm{unif}"] \arrow[d, dashed] & x = M \arrow[dd] \\
x = M, x = M' \arrow[d, dashed] & \\
x = M', x = M' \arrow[r, dashed, "\mathrm{unif}"'] & x = M'
\end{tikzcd}$$

and

$$\begin{tikzcd}
\langle M \mid x = y, x = y, D\rangle \arrow[r, "\mathrm{unif}"] \arrow[d, dashed] & \langle M \mid x = y, D\rangle \arrow[dd] \\
\langle M[x := y] \mid x = y, D[x := y]\rangle \arrow[d, dashed] & \\
\langle M[x := y] \mid D[x := y]\rangle \arrow[r, phantom, "\equiv"] & \langle M[x := y] \mid D[x := y]\rangle
\end{tikzcd}$$

Remember that $y[x := y] = y$ and for every term P we have $P[x := y][x := y] = P[x := y]$.

  - To fill in the left square we again use a tiling argument with the following diagram:

$$\begin{tikzcd}
\langle M \mid x = y, D\rangle \arrow[r, "\sigma"] \arrow[d, dashed] & \langle M^\sigma \mid x^\sigma = y^\sigma, D^\sigma\rangle \arrow[d] \\
\langle M[x := y] \mid D[x := y]\rangle \arrow[r, dashed, "\sigma"'] & \langle M^\sigma[x^\sigma := y^\sigma] \mid D^\sigma[x^\sigma := y^\sigma]\rangle
\end{tikzcd}$$

$\square$

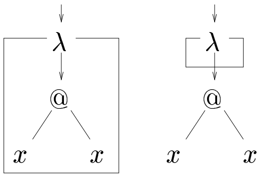


Figure 27. Two scoped graphs without an intermediate graph

LEMMA 6.11. *Given two scoped graphs g and h and a homomorphism $f : g \to h$. Then $g_u = h_u$.*

PROOF. By Proposition 6.7(i) we can assume g and h are garbage free. By Lemma 6.9, we have homomorphisms $f_1 : g_u^s \to g$ and $f_2 : h_u^s \to h$. By Proposition 6.7(ii) $f \circ f_1 : g_u^s \to h$ is a homomorphism. It is then easy to show that $g_u = h_u$. □

We can now prove that $\mathcal{R}_2$ is sound with respect to tree unwinding.

THEOREM 6.12. *Given $M, N \in \Lambda\circ$, representing graphs g and h, respectively. If $\mathcal{R}_2 \vdash M = N$ then $g_u = h_u$.*

PROOF. The soundness of all axioms except copying is trivial. The soundness of copying follows from Lemma 6.10 and Lemma 6.11. □

6.3. Completeness

The proof of completeness uses almost the same strategy as the proof for the first-order case given in [AK96]. There, given two graphs g_1 and g_2 with the same tree unwinding an intermediate finite graph h is constructed such that there exist homomorphisms from h to g_1 and g_2, respectively. It is then proven that if there exists a homomorphism from one graph to another, these two graphs are provably equal. Because our notion of homomorphism preserves scope information and garbage it is in general impossible to find such an intermediate graph for scoped lambda-graphs. Take the graphs g_1 and g_2 displayed in Fig. 27. Assume there is a homomorphism from a certain graph h to both g_1 and g_2. Then h must have two nodes l and a, respectively, such that l maps to the lambda-nodes and the argument a of l maps to the application nodes. From the condition on belongs to, it then follows that a belongs to the *root* and that a belongs to l. Contradiction.

However, given scoped lambda-graphs g_1 and g_2 we are able to find a pair of intermediate finite graphs (g_1', g_2'), differing only by their scope information, such that there exist homomorphisms from g_1' to $\mathrm{gc}(g_1)$ and from g_2' to $\mathrm{gc}(g_2)$ (Lemma 6.13). The two graphs in the figure are an intermediate pair themselves, with the identity as homomorphisms to themselves. We then prove that if there exists a homomorphism from one graph to another then any two representations of those graphs are provably equal (Lemmas 6.9 and 6.14). Completeness then follows from the fact that we can prove graphs that differ by garbage and scope information equal.

Given two scoped graphs g_1 and g_2, we can take as intermediate graphs the corresponding tree unwindings. Unfortunately the provable equality for homomorphisms only works for finite graphs (just as in the first-order case) and the unwinding is in general infinite. Thus, we prove next that if there exists a pair of possibly infinite intermediate graphs there also exists a pair of finite intermediate graphs:

LEMMA 6.13. *Given finite scoped graphs g_1, g_2, possibly infinite scoped graphs h_1, h_2 with $h_1 \sim h_2$, and homomorphisms $f_1 : h_1 \to g_1$ and $f_2 : h_2 \to g_2$. Then there exist two finite scoped graphs h_1', h_2' with $h_1' \sim h_2'$, and homomorphisms $f_1' : h_1' \to g_1$ and $f_2' : h_2' \to g_2$.*

PROOF. Define the scoped graph $g_1 \times g_2$ by

$$\begin{array}{lcl} V_\times & = & \{(f_1(v), f_2(v)) \mid v \in V_{h_1}\} \\ L_\times((v_1, v_2)) & = & L_{g_1}(v_1) \\ A_\times((v_1, v_2)) & = & (A_{g_1}(v_1), A_{g_2}(v_2)) \\ S_\times((f_1(v), f_2(v))) & = & \{(f_1(w), f_2(w)) \mid w \in S_{h_1}(v)\} \\ r_\times & = & (r_{g_1}, r_{g_2}) \end{array}$$

The function $\pi : (x, y) \mapsto x$ defines a homomorphism from $g_1 \times g_2$ to g_1 and from $g_2 \times g_1$ to g_2. All conditions are straight from the definition except belonging to. If a node $(f_1(w), f_2(w))$ belongs to a node $(f_1(v), f_2(v))$ then w must belong to v and by the fact that f is a homomorphism $f_1(w)$ must belong to $f_1(v)$.

We also have that $g_1 \times g_2 \sim g_2 \times g_1$. $\square$

LEMMA 6.14. *Given scoped graphs g, h and a homomorphism $f : g \to h$. If M and N represent g and h, respectively, then $\mathcal{R}_2 \vdash M = N$.*

PROOF. Since we can prove equal any two representations of the same scoped graph (Theorem 4.15), it follows that we only need to show there exist two representations of g and h that we can prove equal. The two representations we choose are $\psi_{\text{pre}}(g)$ and $\psi_{\text{pre}}(h)$. We prove them equal by showing there exists a variable mapping σ_f such that $\psi_{\text{pre}}(g)^{\sigma_f} \equiv \psi_{\text{pre}}(h)$. We define:

$$\sigma_f(x_v) = x_{f(v)} \quad ,$$

where x_v is the variable associated to node v in the construction of $\psi_{\text{pre}}(g)$ (see Definition 5.1). We can safely assume that the set of variables bound by lambda's in $\psi_{\text{pre}}(g)$ and $\psi_{\text{pre}}(h)$ are disjoint from the set of free variables and recursion variables. Since the scopes of distinct lambda-nodes that map to the same lambda-node are disjoint (due to the belong to condition of homomorphism) we can safely assume

$$y_v = y_{f(v)} \quad .$$

Before we prove $\psi_{\text{pre}}(g)^{\sigma_f} \equiv \psi_{\text{pre}}(h)$ we first prove two claims. The first claim is that for every argument a we have $N_a^{\sigma_f} \equiv N_{f(a)}$. We prove this claim by cases on a:
- free variable x: We have $f(x) \equiv x$.
- pointer v: We have $N_a^{\sigma_f} \equiv x_v^{\sigma_f} \equiv x_{f(v)} \equiv N_{f(a)}$.
- back-pointer $\overline{v}$: We have $N_a^{\sigma_f} \equiv y_v^{\sigma_f} \equiv y_v \equiv y_{f(v)} \equiv N_{f(a)}$.

The second claim is that for every argument a and every set of nodes $W \subseteq V_g$ we have:

$$\langle N_a \mid D_W \rangle^{\sigma_f} \equiv \langle N_{f(a)} \mid D_{f(W)} \rangle \quad .$$

We prove this second claim by structural induction. By the first claim we have $N_a^{\sigma_f} \equiv N_{f(a)}$. Let

$$W' = \{w \in W \mid \forall p \in W : w \in S(p) \Rightarrow w \equiv p\}$$

and

$$W'' = \{w \in f(W) \mid \forall p \in W : w \in S(p) \Rightarrow w \equiv p\} \quad .$$

We can derive from the fact that f preserves the belongs to relation that $f(W') = W''$. We also have that

$$D_W^{\sigma_f} = \{x_w^{\sigma_f} = M_w^{\sigma_f}\}_{w \in W'} \text{ and } D_{f(W)} = \{x_w = M_w\}_{w \in W''} \quad .$$

All we need to show now is that $(x_w^{\sigma_f} = M_w^{\sigma_f}) \equiv (x_{f(w)} = M_{f(w)})$. For the left-hand sides this is trivial, for the right-hand sides we need to do a case analysis on the type of w (which is also the type of $f(w)$ because f is a homomorphism):

- black hole. The right-hand side is the same as the left-hand side for both equations.

- application node. We have that

$$M_w^{\sigma_f} \equiv N_{A(w)_1}^{\sigma_f}\ N_{A(w)_2}^{\sigma_f} \text{ and } M_{f(w)} \equiv N_{A(f(w))_1)}\ N_{A(f(w)_1)}\ .$$

  Because f is a homomorphism we have for $i = 1, 2$ that $A(f(w))_i \equiv f(A(w)_i)$. By the first claim it then follows that $M_w^{\sigma_f} \equiv M_{f(w)}$.

- lambda-node. We have that

$$M_w^{\sigma_f} \equiv \lambda y_w.\langle N_{A(w)} \mid D_{S(w)\setminus\{w\}}\rangle^{\sigma_f} \text{ and } M_{f(w)} \equiv \lambda y_{f(w)}.\langle N_{A(f(w))} \mid D_{S(f(w))\setminus\{f(w)\}}\rangle\ .$$

  Because f is a homomorphism we have $A(f(w)) \equiv f(A(w))$ and $f(S(w) \setminus \{w\}) = S(f(w)) \setminus \{f(w)\}$. So by induction hypothesis we have that

$$\langle N_{A(w)} \mid D_{S(w)\setminus\{w\}}\rangle^{\sigma_f} \equiv \langle N_{A(f(w))} \mid D_{S(f(w))\setminus\{f(w)\}}\rangle\ .$$

  By assumption $y_w \equiv y_{f(w)}$ so also $M_w^{\sigma} \equiv M_{f(w)}$.

This completes the proof of the second claim.

Since f is a homomorphism we have $f(r_g) \equiv r_h$ and $f(V_g) \equiv V_h$. By those facts and the previous claim we have:

$$\psi_{\mathrm{pre}}(g)^{\sigma_f} \equiv \langle N_{r_g} \mid D_{V_g}\rangle^{\sigma_f} \equiv \langle N_{f(r_g)} \mid D_{f(V_g)}\rangle \equiv \langle N_{r_h} \mid D_{V_h}\rangle \equiv \psi_{\mathrm{pre}}(h)\ .$$

□

Given a well-formed graph $g = (V, L, A, r)$ we say that M is a representation of g if $\rho(M) = (V, L, A, S, r)$. We can now prove that $\mathcal{R}_2$ is complete with respect to tree unwinding.

THEOREM 6.15. *Given graphs g and h with the same tree unwinding. If M and N represent g and h, respectively, then $\mathcal{R}_2 \vdash M = N$.*

PROOF. Let g_1 and h_1 be $\rho(M)$ and $\rho(N)$, respectively. Since g and h have the same tree unwinding it follows that ${g_1}^s_u \sim {h_1}^s_u$. From Lemmas 6.9 and 6.13 it follows there also exists a pair of finite graphs g_1' and h_1' homomorphic to $\mathrm{gc}(g_1)$ and $\mathrm{gc}(h_1)$, respectively. The result then follows from Lemma 6.14 and Theorem 5.6.

We summarize this proof in the following diagram:

$$\begin{tikzcd}
\arrow[r, phantom, "6.9"] & {g_1}^s_u \arrow[d, "\mathrm{hom}"] & \arrow[r, phantom, "6.13"] & & {h_1}^s_u \arrow[d, "\mathrm{hom}"'] \arrow[r, phantom, "6.9"] & \\
g_1 \arrow[r, "\mathrm{gc}"'] \arrow[d, dash] & \mathrm{gc}(g_1) \arrow[d, dash] & g_1' \arrow[l, "\mathrm{hom}"'] \arrow[d, dash] \arrow[r, phantom, "\sim"] & h_1' \arrow[r, "\mathrm{hom}"'] \arrow[d, dash] & \mathrm{gc}(h_1) \arrow[d, dash] & h_1 \arrow[l, "\mathrm{gc}"'] \arrow[d, dash] \\
M \arrow[r, phantom, "5.6(\mathrm{i})"] & ? \arrow[r, phantom, "6.14"] & ? \arrow[r, phantom, "5.6(\mathrm{ii})"] & ? \arrow[r, phantom, "6.14"] & ? \arrow[r, phantom, "5.6(\mathrm{i})"] & N
\end{tikzcd}$$

The numbers at the top refer to the statement used to prove the squares they are in and the numbers in the bottom row refer to the statement used to prove the terms to the left and right of them equal.
□

	$\langle M \mid D\rangle\ N$	$=$	$\langle M\ N \mid D\rangle$	
	$M\ \langle N \mid D\rangle$	$=$	$\langle M\ N \mid D\rangle$	
	$\langle M \mid\rangle$	$=$	M	
	$\langle\langle M \mid D_1\rangle \mid D_2\rangle$	$=$	$\langle M \mid D_1, D_2\rangle$	
	$\langle M \mid x = \langle N \mid D_1\rangle, D_2\rangle$	$=$	$\langle M \mid x = N, D_1, D_2\rangle$	
	M	$=$	$\langle x \mid x = M\rangle$	x a new variable
$\mathcal{R}_0$	$\langle M \mid x = y, D\rangle$	$=$	$\langle M[x := y] \mid D[x := y]\rangle$	$x \not\equiv y$
	$\langle M \mid D\rangle$	$=$	M	$D \perp M$
$\mathcal{R}_1$	$\lambda x.\langle M \mid D\rangle$	$=$	$\langle \lambda x.M \mid D\rangle$	x not free in D
$\mathcal{R}_2$	M	$=$	N	$\exists\sigma : \mathcal{V} \to \mathcal{V}, N^\sigma \equiv M$

Table 1. Sound and complete axiomatization of tree unwinding

	$\langle M \mid D\rangle\ N$	$\to$	$\langle M\ N \mid D\rangle$	
	$M\ \langle N \mid D\rangle$	$\to$	$\langle M\ N \mid D\rangle$	
	$\langle M \mid\rangle$	$\to$	M	
	$\langle\langle M \mid D_1\rangle \mid D_2\rangle$	$\to$	$\langle M \mid D_1, D_2\rangle$	
	$\langle M \mid x = \langle N \mid D_1\rangle, D_2\rangle$	$\to$	$\langle M \mid x = N, D_1, D_2\rangle$	
	$C_{\text{safe}}[\lambda x.M]$	$\to$	$C_{\text{safe}}[\langle y \mid y = \lambda x.M\rangle]$	y a new variable
	$C_{\text{safe}}[M\ N]$	$\to$	$C_{\text{safe}}[\langle y \mid y = M\ N\rangle]$	y a new variable
$\mathcal{R}_0^{\to}$	$\langle M \mid x = y, D\rangle$	$\to$	$\langle M[x := y] \mid D[x := y]\rangle$	$x \not\equiv y$
	$\langle M \mid D_1, D_2\rangle$	$\to$	$\langle M \mid D_1\rangle$	$D_2 \neq \{\}, D_2 \perp \langle M \mid D_1\rangle$
$\mathcal{R}_1^{\to}$	$\lambda x.\langle M \mid D_1, D_2\rangle$	$\to$	$\langle \lambda x.\langle M \mid D_1\rangle \mid D_2\rangle$	$D_2 \neq \{\}, D_1 \perp D_2, x \notin \text{free}(D_2)$
$\mathcal{R}_2^{\to}$	M	$\to$	N	$\exists\sigma : \mathcal{V} \to \mathcal{V}, N^\sigma \equiv M$

Table 2. Summary of the representational rewriting systems

7. Summary of the representational calculi

In Table 1 we summarize the distinct representational calculi we have introduced so far. We remark that these systems are not minimal, *e.g.,* the empty box garbage collection axiom is a special case of garbage collection and therefore is superfluous. The external merge axiom is also superfluous as it may be derived from naming and internal merge:

$$\begin{array}{rcll}
\langle\langle M \mid D_1\rangle \mid D_2\rangle & = & \langle x \mid x = \langle\langle M \mid D_1\rangle \mid D_2\rangle\rangle & \text{naming} \\
& = & \langle x \mid x = \langle M \mid D_1\rangle, D_2\rangle & \text{internal merge} \\
& = & \langle x \mid x = M, D_1, D_2\rangle & \text{internal merge} \\
& = & \langle x \mid x = \langle M \mid D_1, D_2\rangle\rangle & \text{internal merge} \\
& = & \langle M \mid D_1, D_2\rangle & \text{naming}
\end{array}$$

We conjecture it is not possible to drop any of the other axioms without losing the completeness results proved in the previous sections. This does of course not mean that there are no other possible variations. We could for example replace the application lift axioms by the single axiom

$$\langle M \mid D_1\rangle\langle N \mid D_2\rangle = \langle M\ N \mid D_1, D_2\rangle\ ,$$

and still have an equivalent calculus. Since our goal is to have calculi that are easy to work with, we will not pursue the derivation of a minimal system any further.

In Table 2 we summarize the corresponding representational rewriting systems. The convertibility relation of these systems is the same as their associated provable equality. This was proven in Proposition 4.6 for $\mathcal{R}_0^{\to}$ and is trivial for $\mathcal{R}_1^{\to}$ and $\mathcal{R}_2^{\to}$.

Next, we prove termination of $\mathcal{R}_0^{\rightarrow}$ and $\mathcal{R}_1^{\rightarrow}$. To that end we define a function $|.|$ on cyclic terms that gives a decreasing measure. This function is a pair of two measures, $|.|_1$ and $|.|_2$. $|.|_1$ counts the number of naming redexes; $|.|_2$ associates a multiset to every term that counts for every letrec and every equation the distance to the root, counting passing through a lambda and an application but not counting passing through a letrec. In giving $|.|_1$, we use the measures $|.|_1^i$, $i = 0, 1$ that count the number of unnamed nodes in a term assuming the top node does not ($i = 0$) or does ($i = 1$) need a label.

Definition 7.1. The termination measure for normalization of the representation (written as $|.|$) is defined by:

$$|M| = (|M|_1, |M|_2)$$

where $|M|_1 = |M|_1^1$ and $|M|_1^i (i = 0, 1)$ is defined by

$$\begin{array}{rcl} |x|_1^i & = & 0 \\ |\lambda x.M|_1^i & = & i + |M|_1^1 \\ |M\ N|_1^i & = & i + |M|_1^1 + |N|_1^1 \\ |\langle M \mid x_1 = M_1, \cdots x_n = M_n\rangle|_1^i & = & |M|_1^i + \sum_{j=1}^n |M_j|_1^0 \end{array}$$

and $|M|_2$ is defined by:

$$\begin{array}{rcl} |x|_2 & = & \{\!\{\}\!\} \\ |M\ N|_2 & = & \mathrm{inc}(|M|_2 \cup |N|_2) \\ |\lambda x.M|_2 & = & \mathrm{inc}(|M|_2) \\ |\langle M_0 \mid x_1 = M_1, \cdots x_n = M_n\rangle|_2 & = & \bigcup_{i=0}^n (\{\!\{0\}\!\} \cup |M_i|_2) \end{array}$$

where $\mathrm{inc}(S) = \{\!\{n + 1 \mid n \in S\}\!\}$.

We use the standard ordering on both components and the lexicographic ordering on the pair.

Before proving that every step decreases the termination measure, we first show two lemmas. These lemmas reduce the task of proving $|C[M]| > |C[N]|$ if $C[M] \rightarrow C[N]$, by an application of the rule $M \rightarrow N$, to proving $|M| > |N|$. We adopt the notation:

$$|x_1 = M_1, \ldots, x_n = M_n|_2 = \bigcup_{i=1}^n (\{\!\{0\}\!\} \cup |M_i|_2) \ .$$

Lemma 7.2. *Given $M, N \in \Lambda\circ$ and a context C.*

(i) *If $|M|_1^0 \geq |N|_1^0$ and $|M|_1^1 \geq |N|_1^1$ then for $j = 0, 1$: $|C[M]|_1^j \geq |C[N]|_1^j$.*

(ii) *If $|M|_1^1 > |N|_1^1$ then $|C_{\mathrm{safe}}[M]|_1^1 > |C_{\mathrm{safe}}[N]|_1^1$.*

Proof.

(i) Simple induction on the context C.

(ii) Follows immediately from the claim that there exists a constant c such that for every term P, $|C_{\mathrm{safe}}[P]|_1^1 = c + |P|_1^1$. We prove the claim by cases on C_{safe}:

- $C_{\mathrm{safe}} \equiv C'$: Simple induction on C'.
- $C_{\mathrm{safe}} \equiv C[\lambda x.C']$: It is not hard to prove that $|C[\lambda x.C'[P]]|_1^1 = |C[\lambda x.x]|_1^1 + |C'[P]|_1^1$. The result then follows from the first case.
- $C_{\mathrm{safe}} \equiv C[C'\ M]$: From the first case and the equation: $|C[C'[P]\ M]|_1^1 = |C[x\ x]|_1^1 + |C'[P]|_1^1 + |M|_1^1$.

- $C_{\text{safe}} \equiv C[M\ C']$: Similar to previous case.

$\square$

LEMMA 7.3. *Given $M, N \in \Lambda\circ$ and a context C. If $|M|_2 > |N|_2$ then $|C[M]|_2 > |C[N]|_2$.*

PROOF. By induction on the context C.

- $C \equiv \square$. Trivial.

- $C \equiv P\ C'$.

$$\begin{array}{rcl} |C[M]|_2 & = & |P\ C'[M]|_2 \\ & = & \mathrm{inc}(|P|_2 \cup |C'[M]|_2) \\ & > & \mathrm{inc}(|P|_2 \cup |C'[N]|_2) \\ & = & |P\ C'[N]|_2 \\ & = & |C[N]|_2 \end{array}$$

- $C \equiv C'\ Q$.

$$|C[M]|_2 = \mathrm{inc}(|C'[M]|_2 \cup |Q|_2) > \mathrm{inc}(|C'[N]|_2 \cup |Q|_2) = |C[N]|_2$$

- $C \equiv \lambda x.C'$.

$$|C[M]|_2 = \mathrm{inc}(|C'[M]|_2) > \mathrm{inc}(|C'[N]|_2) = |C[N]|_2$$

- $C \equiv \langle C' \mid D\rangle$.

$$|C[M]|_2 = |C'[M]|_2 \cup \{\!\{0\}\!\} \cup |D|_2 > |C'[N]|_2 \cup \{\!\{0\}\!\} \cup |D|_2 = |C[N]|_2$$

- $C \equiv \langle P \mid x = C', D\rangle$.

$$|C[M]|_2 = |P|_2 \cup \{\!\{0\}\!\} \cup |C'[M]|_2 \cup \{\!\{0\}\!\} \cup |D|_2 > |P|_2 \cup \{\!\{0\}\!\} \cup |C'[N]|_2 \cup \{\!\{0\}\!\} \cup |D|_2 = |C[N]|_2$$

$\square$

THEOREM 7.4. *$\mathcal{R}_1^{\rightarrow}$ is terminating.*

PROOF. We show that if $C[M] \to C[N]$ then $|C[M]| > |C[N]|$. By cases on the rule being applied.

Naming rule. We obviously have:

$$\begin{array}{rcl} |M\ N|_1^1 & > & |\langle y \mid y = M\ N\rangle|_1^1 \\ |\lambda x.M|_1^1 & > & |\langle y \mid y = \lambda x.M\rangle|_1^1 \end{array}$$

By Lemma 7.2(ii) we then have $|C[M]|_1 > |C[N]|_1$, because C must be a safe context. Hence, $|C[M]| > |C[N]|$.

Other rules. We claim that every other rule satisfies $|M|_1^i \geq |N|_1^i$ and $|M|_2 > |N|_2$. It then follows by Lemma 7.2(i) that $|C[M]|_1 \geq |C[N]|_1$ and it follows by Lemma 7.3 that $|C[M]|_2 > |C[N]|_2$. Hence, $|C[M]| > |C[N]|$.

The claim about the first measure is trivial for all rules. We prove next the claim about the second measure

-

$$\begin{array}{rcl} |\langle M \mid D\rangle\ N|_2 & = & \mathrm{inc}(|\langle M \mid D\rangle|_2 \cup |N|_2) \\ & = & \mathrm{inc}(|M|_2 \cup \{\!\{0\}\!\} \cup |D|_2 \cup |N|_2) \\ & = & \mathrm{inc}(|M|_2 \cup |D|_2 \cup |N|_2) \cup \{\!\{1\}\!\} \\ & > & \mathrm{inc}(|M|_2 \cup |D|_2 \cup |N|_2) \cup \{\!\{0\}\!\} \\ & \geq & \mathrm{inc}(|M|_2 \cup |N|_2) \cup \{\!\{0\}\!\} \cup |D|_2 \\ & = & |M\ N|_2 \cup \{\!\{0\}\!\} \cup |D|_2 \\ & = & |\langle M\ N \mid D\rangle|_2 \end{array}$$

-

$$\begin{array}{rcl} |M\ \langle N \mid D\rangle|_2 & = & \mathrm{inc}(|M|_2 \cup |N|_2 \cup \{\!\{0\}\!\} \cup |D|_2) \\ & > & \mathrm{inc}(|M|_2 \cup |N|_2) \cup \{\!\{0\}\!\} \cup |D|_2 \\ & = & |\langle M\ N \mid D\rangle|_2 \end{array}$$

-

$$\begin{array}{rcl} |\lambda x.\langle M \mid D_1, y = N, D_2\rangle|_2 & = & \mathrm{inc}(|M|_2 \cup \{\!\{0\}\!\} \cup |D_1|_2 \cup \{\!\{0\}\!\} \cup |N|_2 \cup |D_2|_2) \\ & > & \mathrm{inc}(|M|_2 \cup \{\!\{0\}\!\} \cup |D_1|_2) \cup \{\!\{0,0\}\!\} \cup |N|_2 \cup |D_2|_2 \\ & = & |\langle \lambda x.\langle M \mid D_1\rangle \mid y = N, D_2\rangle|_2 \end{array}$$

-

$$\begin{array}{rcl} |\langle M \mid D_1, D_2\rangle|_2 & = & |M|_2 \cup \{\!\{0\}\!\} \cup |D_1|_2 \cup |D_2|_2 \\ & > & |M|_2 \cup \{\!\{0\}\!\} \cup |D_1|_2 \\ & = & |\langle M \mid D_1\rangle|_2 \end{array}$$

-

$$\begin{array}{rcl} |\langle M \mid \rangle|_2 & = & |M| \cup \{\!\{0\}\!\} \\ & > & |M|_2 \end{array}$$

-

$$\begin{array}{rcl} |\langle M \mid x = \langle N \mid D_1\rangle, D_2\rangle|_2 & = & |M|_2 \cup \{\!\{0\}\!\} \cup (|N|_2 \cup \{\!\{0\}\!\} \cup |D_1|_2) \cup \{\!\{0\}\!\} \cup |D_2|_2 \\ & > & |M|_2 \cup \{\!\{0\}\!\} \cup |N|_2 \cup \{\!\{0\}\!\} \cup |D_1|_2 \cup |D_2|_2 \\ & = & |\langle M \mid x = N, D_1, D_2\rangle|_2 \end{array}$$

-

$$\begin{array}{rcl} |\langle\langle M \mid D_1\rangle \mid D_2\rangle|_2 & = & (|M|_2 \cup \{\!\{0\}\!\} \cup |D_1|_2) \cup \{\!\{0\}\!\} \cup |D_2|_2 \\ & > & |M|_2 \cup \{\!\{0\}\!\} \cup |D_1|_2 \cup |D_2|_2 \\ & = & |\langle M \mid D_1, D_2\rangle|_2 \end{array}$$

-

$$\begin{array}{rcl} |\langle M \mid x = y, D\rangle|_2 & = & |M|_2 \cup \{\!\{0\}\!\} \cup \{\!\{0\}\!\} \cup |D|_2 \\ & > & |M|_2 \cup \{\!\{0\}\!\} \cup |D|_2 \\ & = & |M[x := y]|_2 \cup \{\!\{0\}\!\} \cup |D[x := y]|_2 \\ & = & |\langle M[x := y] \mid D[x := y]\rangle|_2 \end{array}$$

□

8. Alternative representational rewriting systems

We now investigate alternative representational rewriting systems obtained by orienting the naming and copying axioms in the opposite direction. When we orient the copy axiom from right to left we obtain the compression rule:

$$M \to N \quad \exists \sigma : \mathcal{V} \to \mathcal{V}, M^\sigma \equiv N, M \not\equiv N \ .$$

The proviso $M \not\equiv N$ is there to ensure termination. When we orient the naming axiom from right to left we obtain the unique substitution rule:

$$\langle x \mid x = M\rangle \to M \ .$$

If we want to replace naming in $\mathcal{R}_0^{\rightarrow}$ with this rule then, to avoid critical pairs with the lift and merge axioms, we extend the unique substitution rule to

$$\begin{array}{rcll} \langle C_{\text{name}}[x] \mid x = M, D\rangle & \to & \langle C_{\text{name}}[M] \mid D\rangle & x \text{ not shared} \\ \langle M \mid y = C_{\text{name}}[x], x = N, D\rangle & \to & \langle M \mid y = C_{\text{name}}[N], D\rangle & y \not\equiv x, x \text{ not shared} \end{array}$$

where x not shared means there exists only once reference to x, and C_{name} is

$$C_{\text{name}} ::= \square \mid C_{\text{name}}\ M \mid M\ C_{\text{name}} \mid \langle C_{\text{name}} \mid D\rangle \mid \langle M \mid x = C_{\text{name}}, D\rangle\ .$$

This restriction on the context avoids substituting under a lambda-abstraction. This needs to be forbidden because it would move some nodes into the scope of a lambda, which is not allowed in $\mathcal{R}_0$. We remove this restriction for $\mathcal{R}_1$ and $\mathcal{R}_2$, due to their completeness with respect to well-formed graphs. Hence, we can replace naming in $\mathcal{R}_1^{\rightarrow}$ and $\mathcal{R}_2^{\rightarrow}$ with the more general rules:

$$\begin{array}{rcll} \langle C[x] \mid x = M, D\rangle & \to & \langle C[M] \mid D\rangle & x \text{ not shared} \\ \langle M \mid y = C[x], x = N, D\rangle & \to & \langle M \mid y = C[N], D\rangle & y \not\equiv x, x \text{ not shared} \end{array}$$

PROPOSITION 8.1.

$$\begin{array}{rll} \mathcal{R}_0 \vdash & \langle C_{\text{name}}[x] \mid x = M\rangle = C_{\text{name}}[M] & x \textit{ not shared} \\ \mathcal{R}_1 \vdash & \langle C[x] \mid x = M\rangle = C[M] & x \textit{ not shared} \end{array}$$

PROOF. The proof uses structural induction on the context $C[.]$

- $C \equiv \square$. Follows from the naming axiom.

- $C \equiv C'\ N$.

$$\begin{array}{rcll} \langle C[x] \mid x = M\rangle & \equiv & \langle C'[x]\ N \mid x = M\rangle & \\ & = & \langle C'[x] \mid x = M\rangle\ N & \text{left lift} \\ & = & C'[M]\ N \equiv C[M] & \text{induction hypothesis} \end{array}$$

- $C \equiv N\ C'$.

$$\begin{array}{rcll} \langle C[x] \mid x = M\rangle & \equiv & \langle N\ C'[x] \mid x = M\rangle & \\ & = & N\ \langle C'[x] \mid x = M\rangle & \text{right lift} \\ & = & N\ C'[M] \equiv C[M] & \text{induction hypothesis} \end{array}$$

- $C \equiv \lambda y.C'$. (This case does not occur for C_{name}.)

$$\begin{array}{rcll} \langle C[x] \mid x = M\rangle & \equiv & \langle \lambda y.C'[x] \mid x = M\rangle & \\ & = & \lambda y.\langle C'[x] \mid x = M\rangle & \text{lambda-lift} \\ & = & \lambda y.C'[M] \equiv C[M] & \text{induction hypothesis} \end{array}$$

- $C \equiv \langle C' \mid D\rangle$.

$$\begin{array}{rcll} \langle C[x] \mid x = M\rangle & \equiv & \langle\langle C'[x] \mid D\rangle \mid x = M\rangle & \\ & = & \langle C'[x] \mid D, x = M\rangle & \text{external merge} \\ & = & \langle\langle C'[x] \mid x = M\rangle \mid D\rangle & \text{external merge} \\ & = & \langle C'[M] \mid D\rangle \equiv C[M] & \text{induction hypothesis} \end{array}$$

- $C \equiv \langle N \mid y = C', D\rangle$.

$$\begin{array}{rcll} \langle C[x] \mid x = M\rangle & \equiv & \langle\langle N \mid y = C'[x], D\rangle \mid x = M\rangle & \\ & = & \langle N \mid y = C'[x], D, x = M\rangle & \text{external merge} \\ & = & \langle N \mid y = \langle C'[x] \mid x = M\rangle, D\rangle & \text{internal merge} \\ & = & \langle N \mid y = C'[M], D\rangle \equiv C[M] & \text{induction hypothesis} \end{array}$$

$\mathcal{R}_0^{\text{base}}$:

$$\begin{array}{rcll}
\langle M \mid D\rangle\ N & \to & \langle M\ N \mid D\rangle & \\
M\ \langle N \mid D\rangle & \to & \langle M\ N \mid D\rangle & \\
\langle M \mid \rangle & \to & M & \\
\langle\langle M \mid D_1\rangle \mid D_2\rangle & \to & \langle M \mid D_1, D_2\rangle & \\
\langle M \mid x = \langle N \mid D_1\rangle, D_2\rangle & \to & \langle M \mid x = N, D_1, D_2\rangle & \\
\langle M \mid x = y, D\rangle & \to & \langle M[x := y] \mid D[x := y]\rangle & x \not\equiv y
\end{array}$$

$\mathcal{R}_0^{\text{name}}$:

$$\begin{array}{rcll}
C_{\text{safe}}[\lambda x.M] & \to & C_{\text{safe}}[\langle y \mid y = \lambda x.M\rangle] & y \text{ a new variable} \\
C_{\text{safe}}[M\ N] & \to & C_{\text{safe}}[\langle y \mid y = M\ N\rangle] & y \text{ a new variable}
\end{array}$$

$\mathcal{R}_0^{\text{unq}}$:

$$\begin{array}{rcll}
\langle C_{\text{name}}[x] \mid x = M, D\rangle & \to & \langle C_{\text{name}}[M] \mid D\rangle & x \text{ not shared} \\
\langle M \mid x = C_{\text{name}}[y], y = N, D\rangle & \to & \langle M \mid x = C_{\text{name}}[N], D\rangle & x \not\equiv y, y \text{ not shared}
\end{array}$$

$\mathcal{R}_1^{\text{base}}$:

$$\begin{array}{rcll}
\langle M \mid D_1, D_2\rangle & \to & \langle M \mid D_1\rangle & D_2 \neq \{\}, D_2 \perp \langle M \mid D_1\rangle \\
\lambda x.\langle M \mid D_1, D_2\rangle & \to & \langle \lambda x.\langle M \mid D_1\rangle \mid D_2\rangle & D_2 \neq \{\}, D_1 \perp D_2, x \notin \text{free}(D_2)
\end{array}$$

$\mathcal{R}_1^{\text{unq}}$:

$$\begin{array}{rcll}
\langle C[x] \mid x = M, D\rangle & \to & \langle C[M] \mid D\rangle & x \text{ not shared} \\
\langle M \mid x = C[y], y = N, D\rangle & \to & \langle M \mid x = C[N], D\rangle & x \not\equiv y, y \text{ not shared}
\end{array}$$

$\mathcal{R}_2^{\text{copy}}$:

$$\begin{array}{rcll}
M & \to & N & \exists \sigma : \mathcal{V} \to \mathcal{V}, N^\sigma \equiv M
\end{array}$$

$\mathcal{R}_2^{\text{comp}}$:

$$\begin{array}{rcll}
M & \to & N & \exists \sigma : \mathcal{V} \to \mathcal{V}, M^\sigma \equiv N, M \not\equiv N
\end{array}$$

system	rules	confluent	terminating
$\mathcal{R}_0^{\rightarrow}$	$\mathcal{R}_0^{\text{base}}, \mathcal{R}_0^{\text{name}}$	yes	yes
$\mathcal{R}_0^{\leftarrow}$	$\mathcal{R}_0^{\text{base}}, \mathcal{R}_0^{\text{unq}}$	yes	yes
$\mathcal{R}_1^{\rightarrow}$	$\mathcal{R}_0^{\rightarrow}, \mathcal{R}_1^{\text{base}}$	yes	yes
$\mathcal{R}_1^{\leftarrow}$	$\mathcal{R}_0^{\text{base}}, \mathcal{R}_1^{\text{unq}}, \mathcal{R}_1^{\text{base}}$	yes	yes
$\mathcal{R}_2^{\rightrightarrows}$	$\mathcal{R}_1^{\rightarrow}, \mathcal{R}_2^{\text{copy}}$	yes	no
$\mathcal{R}_2^{\rightleftarrows}$	$\mathcal{R}_1^{\rightarrow}, \mathcal{R}_2^{\text{comp}}$	?	yes
$\mathcal{R}_2^{\leftrightarrows}$	$\mathcal{R}_1^{\leftarrow}, \mathcal{R}_2^{\text{copy}}$	no	no
$\mathcal{R}_2^{\leftleftarrows}$	$\mathcal{R}_1^{\leftarrow}, \mathcal{R}_2^{\text{comp}}$	no	yes

Table 3. Summary of alternative representational rewriting systems

□

In Table 3 we present how all the different systems combine with each other. We have adopted the following notation: we superscript a representational rewriting system $\mathcal{R}$ with one or two arrow(s). The (top) arrow indicates the orientation of the naming axiom. If present, the lower arrow indicates the orientation of the copy axiom.

REMARK 8.2. Given a graph g and a set W such that $\mathrm{gc}(g, W)$ is well defined we have shown that for every M representing g we could find an N representing $\mathrm{gc}(g, W)$, such that $M \underset{\mathcal{R}_1^{\rightarrow}}{\twoheadrightarrow} N$. For $\mathcal{R}_1^{\leftarrow}$ however such an N does not necessarily exist. For example, in the graph of $\langle x \mid y = y\ (x\ x)\rangle$ we have two application nodes which are garbage. On the graph it is legal to remove only the first of the two. In $\mathcal{R}_1^{\rightarrow}$ we have the following reduction sequence:

$$
\begin{array}{rcl}
\langle x \mid y = y\ (x\ x)\rangle & \underset{\mathcal{R}_1^{\rightarrow}}{\longrightarrow} & \langle x \mid y = y\ \langle z \mid z = x\ x\rangle\rangle \\
& \underset{\mathcal{R}_1^{\rightarrow}}{\twoheadrightarrow} & \langle x \mid y = y\ z, z = x\ x\rangle \\
& \underset{\mathcal{R}_1^{\rightarrow}}{\longrightarrow} & \langle x \mid z = x\ x\rangle\ .
\end{array}
$$

The last term in the sequence properly represents the garbage collection of the single node. The naming step at the beginning is however essential in this rewriting sequence. Hence, in $\mathcal{R}_1^{\leftarrow}$ it is impossible to rewrite to a term representing the graph with the single node removed.

PROPOSITION 8.3.

(i) *The following rewriting systems are confluent:* $\mathcal{R}_0^{\leftarrow}, \mathcal{R}_1^{\leftarrow}$.

(ii) *The following rewriting systems are terminating:* $\mathcal{R}_0^{\leftarrow}, \mathcal{R}_1^{\leftarrow}, \mathcal{R}_2^{\rightleftarrows}, \mathcal{R}_2^{\leftleftarrows}$.

(iii) *Given* $M, N \in \Lambda\circ$. *Then:*

$$
\begin{array}{l}
\mathcal{R}_0 \vdash M = N \textit{ iff } M \underset{\mathcal{R}_0^{\leftarrow}}{\longleftrightarrow} N \\
\mathcal{R}_1 \vdash M = N \textit{ iff } M \underset{\mathcal{R}_1^{\leftarrow}}{\longleftrightarrow} N \\
\mathcal{R}_2 \vdash M = N \textit{ iff } M \underset{\mathcal{R}_2^{\leftrightarrows}}{\longleftrightarrow} N \textit{ iff } M \underset{\mathcal{R}_2^{\rightleftarrows}}{\longleftrightarrow} N \textit{ iff } M \underset{\mathcal{R}_2^{\leftleftarrows}}{\longleftrightarrow} N\ .
\end{array}
$$

PROOF.

(i) From local confluence and the next item.

(ii) To prove termination of $\mathcal{R}_2^{\rightleftarrows}$ we use the same pair of measures as in the proof of termination of $\mathcal{R}_1^{\rightarrow}$ (Theorem 7.4). For the other systems we use a modified version of this pair of measures. The first counts the number of equations, the second measure remains unchanged.

(iii) Completeness is obvious, soundness follows from:

$$
\begin{array}{rcll}
\langle C[x] \mid x = M, D\rangle & = & \langle\langle C[x] \mid x = M\rangle \mid D\rangle & \text{external merge} \\
& = & \langle C[M] \mid D\rangle & \text{Proposition 8.1}
\end{array}
$$

and

$$
\begin{array}{rcll}
\langle M \mid x = C[y], y = N, D\rangle & = & \langle M \mid x = \langle C[y] \mid y = N\rangle, D\rangle & \text{internal merge} \\
& = & \langle M \mid x = C[N], D\rangle & \text{Proposition 8.1}\ .
\end{array}
$$

□

Because in the first-order case the compression rule together with naming yields a confluent system, we conjecture that the same holds for cyclic lambda terms.

Conjecture 8.4. *The rewriting system $\mathcal{R}_2^{\rightleftarrows}$ is confluent.*

Remark 8.5. Confluence is lost for the systems $\mathcal{R}_2^{\leftrightarrows}$ and $\mathcal{R}_2^{\leftleftarrows}$.

$\mathcal{R}_2^{\leftrightarrows}$. The counterexample to confluence is similar to the one presented in [AK96]. Consider the following two reductions:

$$\langle x \mid x = w\ x\rangle \xrightarrow[\text{cp}]{} \langle x \mid x = w\ y, y = w\ x\rangle \xrightarrow[\text{unq}]{} \langle x \mid x = w\ (w\ x)\rangle$$

and

$$\langle x \mid x = w\ x\rangle \xrightarrow[\text{cp}]{} \langle x \mid x = w\ y, y = w\ z, z = w\ y\rangle \xrightarrow[\text{unq}]{}\!\!\!\!\!\rightarrow \langle w\ y \mid y = w\ (w\ y)\rangle\ .$$

It is impossible to rewrite the two right-hand side terms to the same term: Every reduct of the first will have an even number of w's and every reduct of the second an odd number. In [AK96] we pointed out that confluence can be regained by giving a name to the subexpression $w\ x$.

$\mathcal{R}_2^{\leftleftarrows}$. The counterexample consists of the following two reductions:

$$\langle x_1 \mid x_1 = w\ x_2, x_2 = w\ x_3, x_3 = w\ x_4, x_4 = w\ x_5, x_5 = w\ x_6, x_6 = w\ x_1\rangle \xrightarrow[\text{comp}]{}$$

$$\langle x_1 \mid x_1 = w\ x_2, x_2 = w\ x_1\rangle \xrightarrow[\text{unq}]{} \langle x_1 \mid x_1 = w\ (w\ x_1)\rangle$$

and

$$\langle x_1 \mid x_1 = w\ x_2, x_2 = w\ x_3, x_3 = w\ x_4, x_4 = w\ x_5, x_5 = w\ x_6, x_6 = w\ x_1\rangle \xrightarrow[\text{comp}]{}$$

$$\langle x_1 \mid x_1 = w\ x_2, x_2 = w\ x_3, x_3 = w\ x_1\rangle \xrightarrow[\text{unq}]{} \langle x_1 \mid x_1 = w\ (w\ (w\ x_1))\rangle\ .$$

It is impossible to rewrite the two right-hand side terms to the same term because they are in normal form.

In Section 11, we will show that the non-confluence of $\mathcal{R}_2^{\leftrightarrows}$ is not a serious problem, since the system enjoys a new property which guarantees uniqueness of infinite normal forms. These infinite normal forms correspond to the possibly infinite tree unwinding.

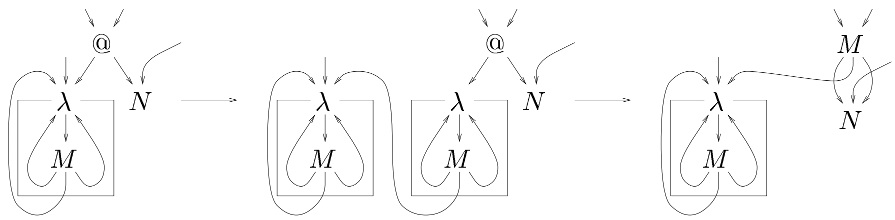


Figure 28. β-reduction on scoped lambda-graphs

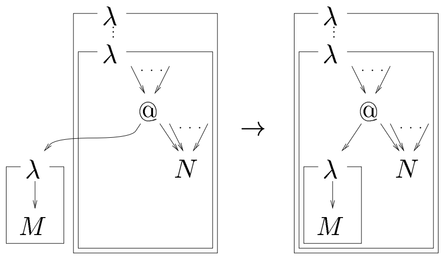


Figure 29. Moving the lambda-node and its scope inside the scope of the application node.

Part II
The computational behavior of cyclic terms

9. The cyclic lambda calculus $\lambda\circ_{\mathrm{name}}$

The representational calculus $\mathcal{R}_2$ makes terms and graphs isomorphic. We now give computational meaning to our graphs and terms. This is done by first introducing β-reduction on scoped graphs. We will then give an axiomatic view of this reduction.

β-reduction on lambda-graphs

A β-redex in a lambda-graph is an application node whose first argument is a lambda-node. The contraction of a β-redex is a two-step process (see Fig. 28). In the first step we check if the reference

Figure 30. Beta-reduction principles for interaction nets and lambda-graphs

to the lambda-node from the application node is unique and if the application node is outside the scope of the lambda-node. If one of these tests fails we copy the lambda-node and its scope in such a way that the test succeeds on the result (see the left step of Fig. 28). We then place the lambda-node (the copy if a copy has been made) and its scope in the same scope of the application node (the original if a copy has been made)(see Fig. 29). The second step is a redirection of pointers consisting of: 1) replacing the application node by an indirection node whose argument is the argument of the lambda-node, 2) replacing the lambda-node by an indirection node whose argument is the former right argument of the application node. Note that all pointers to the indirection node replacing the lambda-node need to be changed from back-pointers to normal pointers. This second step is drawn on the right of Fig. 30. On the left of the same figure we have drawn the β-reduction principle used by interaction nets [Laf90]. There the use of indirection nodes is superfluous because there is exactly one reference to the application node and exactly one back-pointer to the lambda-node.

In the following definition we use the notation $i \oplus V$ for the set $\{i \oplus v \mid v \in V\}$, for $i = 1, 2$.

Definition 9.1. Given a scoped graph $g \equiv (V, L, A, S, r)$ and an application node $v \in V$ such that the first argument w of v is a lambda-node. The contraction of the β-redex v in g, written as $g \xrightarrow[\beta]{} h$, is defined as follows.

(i) We first define a scoped graph $g_1 \equiv (V', L', A', S', r')$ by:

- $V' = \begin{cases} (V \setminus S(w)) \oplus S(w) & \text{if the reference to } w \text{ is unique and } v \notin S(w) \\ V \oplus S(w) & \text{otherwise} \end{cases}$

- $L'(i \oplus u) = L(u), i = 1, 2$

The functions f_1 and f_2 we use in giving the arguments are given for single arguments and extended pointwise to strings of arguments.

- $A'(1 \oplus u) = f_1(A(u))$, if $u \not\equiv v$
- $A'(1 \oplus v) = (2 \oplus w)(f_1(A(v)_2))$, where

$$\begin{array}{rcl} f_1(x) & = & x \\ f_1(v) & = & 1 \oplus v \\ f_1(\overline{v}) & = & \overline{1 \oplus v} \end{array}$$

- $A'(2 \oplus u) = f_2(A(u))$, where

$$\begin{array}{rcll} f_2(x) & = & x & \\ f_2(v) & = & 2 \oplus v & \text{, if } v \in S(w) \setminus w \\ f_2(v) & = & 1 \oplus v & \text{, otherwise} \\ f_2(\overline{v}) & = & \overline{2 \oplus v} & \text{, if } v \in S(w) \\ f_2(\overline{v}) & = & \overline{1 \oplus v} & \text{, otherwise} \end{array}$$

- $S'(2 \oplus u) = 2 \oplus S(u)$
- $S'(1 \oplus u) = 1 \oplus S(u)$, if $v \notin S(u)$
- $S'(1 \oplus u) = S(u) \oplus S(w)$, if $v \in S(u)$

- $r' = 1 \oplus r$, where $1 \oplus x = x$.

(ii) We then update g_1 to obtain scoped graph g_2 in the following way:

- $A'(1 \oplus v)_1 := A'(2 \oplus w)$
- $A'(2 \oplus w) := A'(1 \oplus v)_2$

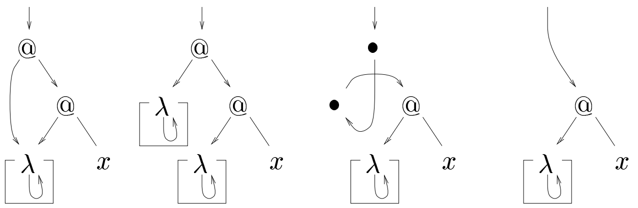


Figure 31. β-reducing $\langle y\ (y\ x) \mid y = \lambda z.z\rangle$

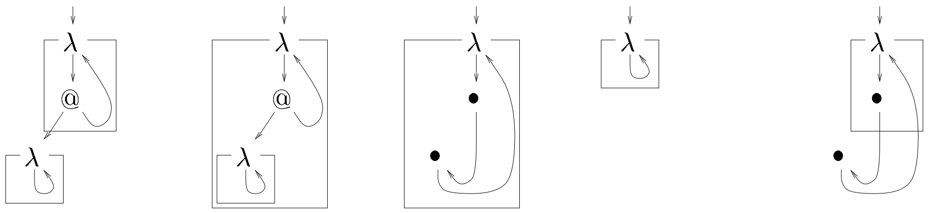


Figure 32. β-reducing $\langle \lambda x.yx \mid y = \lambda x.x\rangle$

- $A'(1 \oplus v) := A'(1 \oplus v)_1$
- $L'(1 \oplus v) := \bullet$
- $L'(2 \oplus w) := \bullet$
- $S'(2 \oplus w) := undefined$
- We replace every argument $\overline{2 \oplus w}$ by $2 \oplus w$

The graph h is then $\mathrm{Sim}(g_2)$.

Example 9.2. In Fig. 31 we present an example of β-reduction. Since we have two references to the lambda-node a copy step is first performed, followed by the redirection and simplification steps. In Fig. 32, since the reference to the lambda-node is unique and the application node is outside the scope of the lambda, we avoid the copying. The first step brings the lambda-node in the same scope as the application node, and, as before, we have the redirection and simplification steps. Note that the first step is necessary, since if we perform the redirection directly in the first graph we obtain

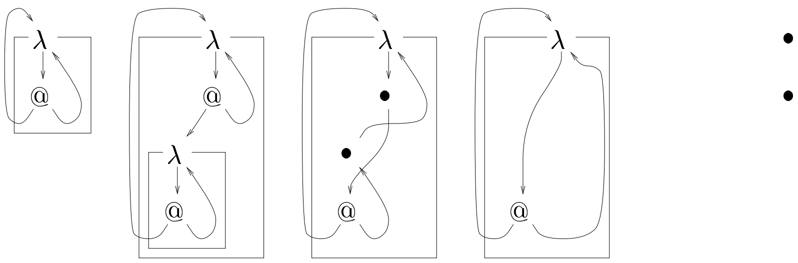


Figure 33. Copying could be necessary with unique references

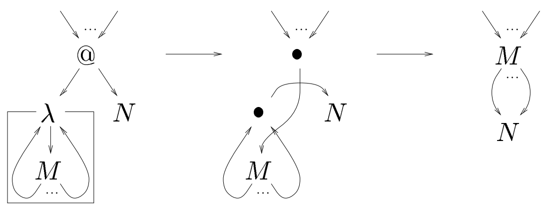


Figure 34. Proof by picture

the ill-formed graph displayed on the extreme right. In this case this ill-formed graph is simplified to a well-formed graph, but this is not true in general. In Fig. 33, we have a unique reference to the lambda-node but still we perform a copy step since the application node and the lambda-node are in the same scope. The graph on the extreme right represents the graph we would have obtained if we did not perform the copy step. Note that graphically the back-pointers to the lambda-nodes change into normal pointers by just replacing the label of the node.

Axiomatization of β-reduction

We now proceed by giving axioms on cyclic terms that describe β-reduction. The first step is described by the axioms introduced so far. For the second step we introduce the following $\beta\circ$-axiom:

$$(\lambda x.M)\ N = \langle M \mid x = N \rangle\ .$$

The rewriting system obtained by orienting the $\beta\circ$-axiom from left to right is denoted by $\beta\circ^{\rightarrow}$. Its reduction relation is denoted by $\xrightarrow[\beta\circ]{}$.

THEOREM 9.3. *Given $M, N \in \Lambda\circ$. If $M \xrightarrow[\beta\circ]{} N$ then $\rho(M) \xrightarrow[\beta]{} \rho(N)$.*

PROOF. See Fig. 34. □

EXAMPLE 9.4. The β-reduction of $\langle x \mid x = \lambda y.x\ y \rangle$ (see Fig. 33, in which we need to draw the root pointer) is described below.

$$\begin{array}{rcll}
M \equiv \langle x \mid x = \lambda y.x\ y \rangle & = & \langle x \mid x = \lambda y.x'\ y, x' = \lambda y'.xy' \rangle & \text{copy} \\
& = & \langle x \mid x = \langle \lambda y.x'\ y \mid x' = \lambda y'.xy' \rangle \rangle & \text{merge} \\
& = & \langle x \mid x = \lambda y.\langle x'\ y \mid x' = \lambda y'.xy' \rangle \rangle & \text{lambda lift} \\
& = & \langle x \mid x = \lambda y.\langle x' \mid x' = \lambda y'.xy' \rangle y \rangle & \text{left lift} \\
& = & \langle x \mid x = \lambda y.(\lambda y'.xy')y \rangle & \text{naming} \\
& = & \langle x \mid x = \lambda y.\langle xy' \mid y' = y \rangle \rangle \equiv N & \beta\circ.
\end{array}$$

In [AK] we have called the subterm xy of $\langle x \mid x = \lambda y.xy \rangle$ an *implicit β-redex* which needs to be made *explicit*, *i.e.,* of the form $(\lambda x.P)Q$, in order to be reduced. In our cyclic calculus, an implicit β-redex can be made explicit by the use of the $\mathcal{R}_2$ representational axioms. However, we would also like to make a redex explicit by applying a representational rewriting system. As shown in Section 8 we have different alternatives. We choose $\mathcal{R}_2^{\leftrightarrows}$ because it contains copying and unique substitution, operations that are sufficient to expose implicit redexes. Referring to the example above, we have

$$\begin{array}{rcl}
\langle x \mid x = \lambda y.x\ y \rangle & \xrightarrow[\text{cp}]{} & \langle x \mid x = \lambda y.x'\ y, x' = \lambda y'.xy' \rangle \\
& \xrightarrow[\text{unq}]{} & \langle x \mid x = \lambda y.\underline{(\lambda y'.xy')y} \rangle\ ,
\end{array}$$

$$
\begin{array}{rcll}
\multicolumn{4}{l}{\mathcal{R}\circ^{\rightarrow}} \\
\langle C[x] \mid x = M, D\rangle & \xrightarrow[\text{es}]{} & \langle C[M] \mid x = M, D\rangle & \\
\langle M \mid x = C[y], y = N, D\rangle & \xrightarrow[\text{is}]{} & \langle M \mid x = C[N], y = N, D\rangle & \\
\langle\langle M \mid D_1\rangle \mid D_2\rangle & \xrightarrow[\text{em}]{} & \langle M \mid D_1, D_2\rangle & \\
\langle M \mid x = \langle N \mid D_1\rangle, D_2\rangle & \xrightarrow[\text{im}]{} & \langle M \mid x = N, D_1, D_2\rangle & \\
\langle M \mid D\rangle\ N & \xrightarrow[\text{lift}]{} & \langle M\ N \mid D\rangle & \\
M\ \langle N \mid D\rangle & \xrightarrow[\text{lift}]{} & \langle M\ N \mid D\rangle & \\
\lambda x.\langle M \mid D_1, D_2\rangle & \xrightarrow[\text{lift}]{} & \langle \lambda x.\langle M \mid D_1\rangle \mid D_2\rangle & D_2 \neq \{\}, D_1 \perp D_2, x \notin \text{free}(D_2) \\
\langle M \mid D_1, D_2\rangle & \xrightarrow[\text{gc}]{} & \langle M \mid D_1\rangle & D_2 \neq \{\}, D_2 \perp \langle M \mid D_1\rangle \\
\langle M \mid\rangle & \xrightarrow[\text{gc}]{} & M & \\
M & \xrightarrow[\text{cp}]{} & N & \exists \sigma : \mathcal{V} \to \mathcal{V}, N^{\sigma} \equiv M \\
\hline
\multicolumn{4}{l}{\beta\circ^{\rightarrow}} \\
(\lambda x.M)\ N & \xrightarrow[\beta\circ]{} & \langle M \mid x = N\rangle &
\end{array}
$$

Table 4. $\lambda\circ^{\rightarrow}_{\text{name}}$: the rewriting systems for the cyclic calculus.

where the underlined term is now ready to be β-reduced. A copy step followed by a unique substitution step captures the inlining transformations discussed in the introduction. This sequence can be more simply described by the following two rules, called external and internal substitution rules,

$$
\begin{array}{rcl}
\langle C[x] \mid x = M, D\rangle & \rightarrow & \langle C[M] \mid x = M, D\rangle \\
\langle M \mid x = C[y], y = N, D\rangle & \rightarrow & \langle M \mid x = C[N], y = N, D\rangle\ .
\end{array}
$$

We admit the above two rules in $\mathcal{R}_2^{\leftrightarrows}$. This means that the variable substitution and unique substitution rules become obsolete. We refer to the obtained system as $\mathcal{R}\circ^{\rightarrow}$ (see Table 4) and to $\mathcal{R}\circ^{\rightarrow}$ combined with $\beta\circ^{\rightarrow}$ as $\lambda\circ^{\rightarrow}_{\text{name}}$. $\xrightarrow[\mathcal{R}\circ]{}$ and $\xrightarrow[\lambda\circ_{\text{name}}]{}$ denote the one-step reductions induced by $\mathcal{R}\circ^{\rightarrow}$ and $\lambda\circ^{\rightarrow}_{\text{name}}$, respectively.

THEOREM 9.5. *Given two scoped graphs g and h such that $g \xrightarrow[\beta]{} h$. If M represents g then there exists an N that represents h such that $M \xrightarrow[\lambda\circ_{\text{name}}]{} \!\!\!\!\rightarrow N$.*

PROOF. Let $g \to g' \to h$, where g' is the result of the first step of β-graph reduction. We claim that we can reduce M to a term M' representing g', such that the occurrence of the redex in M' occurs as a subterm of the form $(\lambda x.P)\ Q$. It is then obvious that rewriting this subterm of M' to $\langle P \mid x = Q\rangle$ yields a term representing h.

To prove the claim we consider the following two cases:

- The redex in g is represented by a subterm of M of the form $\langle \cdots \langle \lambda x.P \mid D_1\rangle \cdots D_n\rangle\ Q$. In this case a sequence of lift steps will expose the redex:

$$\langle \cdots \langle \lambda x.P \mid D_1\rangle \cdots D_n\rangle\ Q \xrightarrow[\text{lift}]{}\!\!\!\!\rightarrow \langle \cdots \langle (\lambda x.P)\ Q \mid D_1\rangle \cdots D_n\rangle\ .$$

  The term on the right-hand side represents g' (which in this case is the same as g).

- The redex in g is a represented by a subterm of M of the form $\langle \cdots \langle x_1 \mid D_{11}\rangle \cdots D_{1n_1}\rangle\ Q$, with M containing the following equations:

$$x_1 = \langle \cdots \langle x_2 \mid D_{21}\rangle \cdots D_{2n_2}\rangle, \cdots, x_n = \langle \cdots \langle \lambda x.P \mid D_{n1}\rangle \cdots D_{nn_n}\rangle\ .$$

In this case we reduce the subterm by lift to $\langle \cdots \langle x_1\ Q \mid D_{11} \rangle \cdots D_{1n_1} \rangle$ and we reduce the equations by internal merge to

$$x_1 = x_2, D_{11}, \cdots, D_{2n_2}, \cdots, x_n = \lambda x.P, D_{n1}, \cdots, D_{nn_n} \ .$$

We then substitute variables until we have the subterm $x_n\ Q$. The term we obtain in this way still represents g. We then use substitution to rewrite the subterm to $(\lambda x.P)\ Q$. This takes care of moving the lambda term into the right scope. If we were forced to copy we are done. If we should not have copied then the nodes coming from the equation $x_n = \lambda x.P$ are now garbage. Like in the proof of Theorem 5.7, we proceed by rewriting the term to application lift and merge normal form. It is then possible to rewrite using garbage collection to a term representing the graph with the node belonging to $x_n = \lambda x.P$ removed. Because of the special form of the set of garbage to be removed, an entire scope, and because the lambda abstraction corresponding to this scope already has a name we do not need naming.

□

We can then conclude that $\lambda\circ^{\rightarrow}_{\text{name}}$ is complete with respect to lambda-graph reduction and unwinding.

We present next our cyclic call-by-name calculus, which for simplicity contains the more general forms of the lambda lift and the garbage collection axioms. We do not include the naming axiom, since the inclusion of the substitution axioms makes it derivable.

Definition 9.6. The call-by-name cyclic calculus ($\lambda\circ_{\text{name}}$) has the following axioms.

$$\begin{array}{llll}
\multicolumn{4}{l}{\beta\circ:} \\
(\lambda x.M)N & = & \langle M \mid x = N \rangle & \\
\multicolumn{4}{l}{\textit{Substitution:}} \\
\langle C[x] \mid x = M, D \rangle & = & \langle C[M] \mid x = M, D \rangle & \\
\langle N \mid x = C[x_1], x_1 = M, D \rangle & = & \langle N \mid x = C[M], x_1 = M, D \rangle & \\
\multicolumn{4}{l}{\textit{Lift:}} \\
\langle M \mid D \rangle N & = & \langle MN \mid D \rangle & \\
M \langle N \mid D \rangle & = & \langle MN \mid D \rangle & \\
\lambda x.\langle M \mid D, D' \rangle & = & \langle \langle \lambda x.M \mid D \rangle \mid D' \rangle & D \perp D' \text{ and } x \text{ not free in } D' \\
\multicolumn{4}{l}{\textit{Merge:}} \\
\langle M \mid x = \langle N \mid D \rangle, D_1 \rangle & = & \langle M \mid x = N, D, D_1 \rangle & \\
\langle \langle M \mid D \rangle \mid D' \rangle & = & \langle M \mid D, D' \rangle & \\
\multicolumn{4}{l}{\textit{Garbage collection:}} \\
\langle M \mid D, D' \rangle & = & \langle M \mid D \rangle & D' \perp \langle M \mid D \rangle \\
\langle M \mid \ \rangle & = & M & \\
\multicolumn{4}{l}{\textit{Copying:}} \\
M & = & N & \exists \sigma : \mathcal{V} \to \mathcal{V}, N^{\sigma} \equiv M
\end{array}$$

$\lambda\circ^{\rightarrow}_{\text{name}}$ (given in Table 4) constitutes the call-by-name cyclic rewriting system.

From now we will omit the subscript 'name' if it is clear from the context that we intend the call-by-name cyclic lambda calculus.

Proposition 9.7. *Given $M, N \in \Lambda\circ$. $\lambda\circ \vdash M = N$ iff $M \xleftrightarrow[\lambda\circ]{} N$.*

The main difference between $\lambda\circ_{\text{name}}$ and the $\lambda\phi$-calculi [AK94, AK] involves to the substitution and merge operations. Here, these operations may occur in any context, whereas in [AK94, AK] they cannot occur on a cycle. For example, the $\lambda\phi$-calculi allow the following substitution step

$$\langle x \mid x = \lambda z.y(Sz), y = \lambda w.y(Sw) \rangle \to \langle x \mid x = \lambda z.(\lambda w.y(Sw))(Sz), y = \lambda w.y(Sw) \rangle \ ,$$

but disallow the following one

$$\langle x \mid x = \lambda z.y(Sz), y = \lambda w.x(Sw)\rangle \to \langle x \mid x = \lambda z.(\lambda w.x(Sw))(Sz), y = \lambda w.x(Sw)\rangle \ .$$

The first rewriting is an example of an acyclic substitution, that is, we substitute the definition of y in the definition of x, where x and y do not lie on the same cyclic plane. Instead, in the second rewriting, x and y lie on the same cyclic plane. This is an example of a cyclic substitution. Cyclic substitution is the cause of non-confluence, as shown by the following example (see Fig. 35, in which we have omitted the garbage):

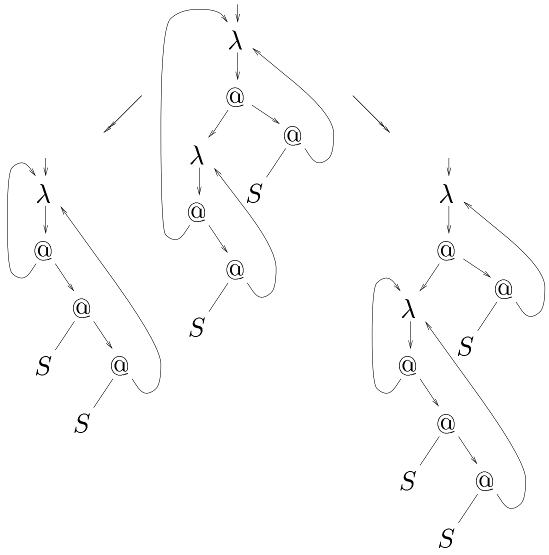


Figure 35. Failure of confluence of $\lambda\circ$

EXAMPLE 9.8.

$$\begin{array}{rcll} M & \equiv & \langle x \mid x = \lambda z.y(Sz), y = \lambda w.x(Sw)\rangle & \\ & \to & \langle x \mid x = \lambda z.(\lambda w.x(Sw))(Sz), y = \lambda w.x(Sw)\rangle & \\ & \twoheadrightarrow & \langle x \mid x = \lambda z.x(S(Sz)), y = \lambda w.x(Sw)\rangle & (*) \end{array}$$

$$\begin{array}{rcll} M & \equiv & \langle x \mid x = \lambda z.y(Sz), y = \lambda w.x(Sw)\rangle & \\ & \to & \langle x \mid x = \lambda z.y(Sz), y = \lambda w.(\lambda z.y(Sz))(Sw)\rangle & \\ & \twoheadrightarrow & \langle x \mid x = \lambda z.y(Sz), y = \lambda w.y(S(Sw))\rangle & (**) \end{array}$$

The terms (*) and (**) (which correspond to the bottom-left and bottom-right graphs, respectively, of Fig. 35) have no common reduct, since in the term (*) an even number of $S's$ is reachable from the root x , while the term (**) will contain an odd number. This 'out-of-synch' phenomenon is also observed by reducing (in at most ω steps) the infinite terms that arise by unwinding the cyclic graphs. This counterexample is more subtle than the one presented in Remark 8.5, since in here there are no operations (that are sound with respect to the infinitary lambda calculus) that can be introduced to regain confluence. By disallowing the substitutions for x and y the counterexample disappears. In [AK] the only way of making the two implicit redexes $y(Sz)$ and $x(Sw)$ explicit is by using the operation of copying:

$$\begin{array}{l} \langle x \mid x = \lambda z.y(Sz), y = \lambda w.x(Sw)\rangle \xrightarrow[\text{cp}]{} \\ \langle x \mid x = \lambda z.y(Sz), y = \lambda w.x'(Sw), x' = \lambda z.y'(Sz), y' = \lambda w.x'(Sw)\rangle \twoheadrightarrow \\ \langle x \mid x = \lambda z.\underline{(\lambda w.x'(Sw))(Sz)}, y = \lambda w.\underline{(\lambda z.y'(Sz))(Sw)}, x' = \lambda z.y'(Sz), y' = \lambda w.x'(Sw)\rangle \ . \end{array}$$

Now, substitutions for y and x' are allowed thus exposing the underlined redexes. However, new implicit redexes are created.

Moreover, in the $\lambda\phi$-calculi, the merge rules had the proviso that only acyclic letrec's could be merged. For example, the following rewriting is allowed:

$$\langle x \mid x = \langle \lambda z.y(Sz) \mid y = \lambda w.y(Sw)\rangle\rangle \to \langle x \mid x = \lambda z.y(Sz), y = \lambda w.y(Sw)\rangle \ ,$$

but not the following one:

$$\langle x \mid x = \langle \lambda z.y(Sz) \mid y = \lambda w.x(Sw)\rangle\rangle \to \langle x \mid x = \lambda z.y(Sz), y = \lambda w.x(Sw)\rangle \ .$$

If the above step were allowed then confluence would have been lost, since the acyclic substitution for y is turned into a cyclic substitution once the internal letrec is removed, as shown in the diagram below.

$$\begin{tikzcd} \langle x \mid x = \langle \lambda z.y(Sz) \mid y = \lambda w.x(Sw)\rangle\rangle \arrow[r] \arrow[d] & \langle x \mid x = \lambda z.y(Sz), y = \lambda w.x(Sw)\rangle \\ \langle x \mid x = \langle \lambda z.(\lambda w.x(Sw))(Sz) \mid y = \lambda w.x(Sw)\rangle\rangle & \end{tikzcd}$$

In summary, in [AK94, AK], the focus was on finding a confluent calculus that could express cyclic lambda graph rewriting. Instead, we do not take confluence as the guiding factor in designing the calculus. Thus, we do not restrict the calculus, but introduce a new way of proving the consistency of the calculus. More specifically, we introduce an approximate notion of confluence - *confluence up to information content*. This notion allows us to abstract away syntactic details.

10. Approximate notion of confluence

Once cycles are admitted it seems natural to consider infinite normal forms instead of normal forms. We thus introduce a new property, confluence up to a quasi order, which guarantees uniqueness of infinite normal forms. In the rest of this section we work with abstract rewriting systems, since we do not need the extra structure terms have to define the necessary notions.

10.1. Confluence up to a quasi order

We start by introducing a few notions about abstract rewriting systems where the set of objects also has a quasi order defined on it.

DEFINITION 10.1. An ordered abstract rewriting system (ARS) is a structure $(A, \to, \preceq)$, where $(A, \to)$ is an ARS and $(A, \preceq)$ is a quasi order.

Given an ordered ARS $(A, \to, \preceq)$. Then

- $\to$ is monotonic with respect to $\preceq$ if

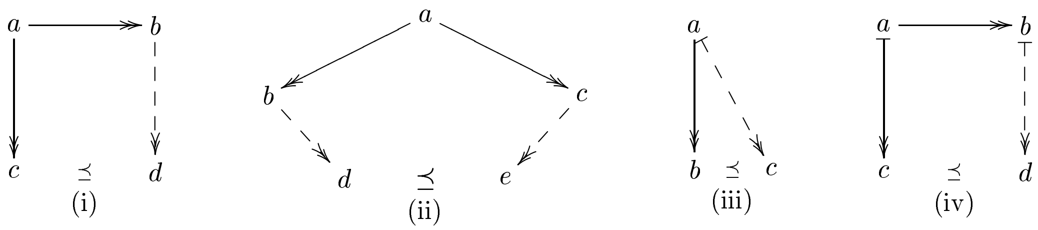
$$\forall a, b \in A : a \to b \Rightarrow a \preceq b \ .$$

- $\to$ is confluent up to $\preceq$ if

$$\forall a, b, c \in A : a \twoheadrightarrow b, a \twoheadrightarrow c \Rightarrow \exists d \in A : b \twoheadrightarrow d, c \preceq d \ .$$

- $\to$ is weakly confluent up to $\preceq$ if:

$$\forall a, b, c \in A : a \twoheadrightarrow b, a \twoheadrightarrow c \Rightarrow \exists d, e \in A : b \twoheadrightarrow d, c \twoheadrightarrow e, d \preceq e \ .$$



(i)	$\to$ is confluent up to $\preceq$
(ii)	$\to$ is weakly confluent up to $\preceq$
(iii)	$\mapsto$ is complete for $\to$ up to $\preceq$
(iv)	$\mapsto$ commutes with $\to$ up to $\preceq$

Figure 36. Pictorial definitions

- We denote by $\overset{\preceq}{\longleftrightarrow}$ the reflexive transitive closure of the relation $((\leftarrow \cup \rightarrow) \cap \preceq)$.
- We define $\simeq = \preceq \cap \succeq$.

See Fig. 36(i) and (ii) for a pictorial definition of confluence and weak confluence up to a quasi order (in short, (weak) confluence up to). The relation between weak confluence and confluence is expressed in the following proposition.

Proposition 10.2. *Given an ordered ARS $(A, \to, \preceq)$. If $\to$ is monotonic with respect to $\preceq$ and weakly confluent up to $\preceq$ then $\to$ is confluent up to $\preceq$.*

Proof. Elementary. $\square$

It is obvious that given an ordered ARS $(A, \to, \preceq)$ we may conclude that $\to$ is confluent if $\to$ is confluent up to $\preceq$ and $\preceq \subseteq \twoheadrightarrow$. Moreover, we have:

Proposition 10.3. *Given an ordered ARS $(A, \to, \preceq)$. If $\to$ is confluent and monotonic with respect to $\preceq$ then $\to$ is confluent up to $\preceq$.*

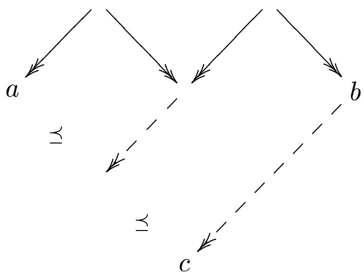
Proof. Trivial. $\square$

Like for normal confluence we have the following proposition:

Proposition 10.4. *Given an ordered ARS $(A, \to, \preceq)$. If $\to$ is confluent up to $\preceq$ then*

$$\forall a, b \in A : a \leftrightarrow b \Rightarrow \exists c \in A : b \twoheadrightarrow c, a \preceq c \ .$$

Proof. Pictorial proof by example:



$\square$

Next, we give an analysis of confluence up to in terms of some simpler properties. We begin with some definitions.

DEFINITION 10.5. Given an ordered ARS $(A, \rightarrow, \preceq)$, and another reduction relation $\mapsto \subseteq \rightarrow$. Then
- $\mapsto$ is complete for $\rightarrow$ up to $\preceq$ if

$$\forall a, b \in A : a \twoheadrightarrow b \Rightarrow \exists c \in A : a \mapsto\!\!\!\twoheadrightarrow c, b \preceq c \ .$$

- We denote by $\circ\!\!\rightarrow$ the reduction relation $\rightarrow \setminus \mapsto$.
- A $\circ\!\!\rightarrow$-conversion is denoted $\overset{\circ}{\leftrightarrow}$.
- $\mapsto$ commutes with $\rightarrow$ up to $\preceq$ if

$$\forall a, b, c \in A : a \twoheadrightarrow b, a \mapsto\!\!\!\twoheadrightarrow c \Rightarrow \exists d \in A : b \mapsto\!\!\!\twoheadrightarrow d, c \preceq d \ .$$

See Fig.36(iii) and (iv) for the pictorial equivalents of these definitions.

LEMMA 10.6. *Given an ordered ARS $(A, \rightarrow, \preceq)$. Let $\mapsto \subseteq \rightarrow$ such that*

(i) $\mapsto$ *is complete for* $\rightarrow$ *up to* $\preceq$*;*

(ii) $\mapsto$ *commutes with* $\rightarrow$ *up to* $\preceq$*.*

Then $\rightarrow$ *is confluent up to* $\preceq$*.*

PROOF. In diagrams:

$$\begin{tikzcd}
a \arrow[rrrr, two heads] \arrow[d, two heads] \arrow[rrd, dashed, mapsto, two heads] \arrow[rd, phantom, "\text{(i)}"] & & & \arrow[d, phantom, "\text{(ii)}"] & a_2 \arrow[d, dashed, mapsto, two heads] \\
a_1 & \preceq & a_4 & \preceq & a_3
\end{tikzcd}$$

□

Less restrictive notions of confluence, such as confluence modulo an equivalence relation have already been proposed [Hue80, DJ90]. We relate these notions to confluence up to.

DEFINITION 10.7. Given an ARS $(A, \rightarrow)$ and an equivalence relation $\sim$ on A. Then
- reduction modulo (written as $\underset{\sim}{\rightarrow}$) is defined by:

$$a \underset{\sim}{\rightarrow} b, \text{ if } \exists a', b' \in A : \ a \sim a' \rightarrow b' \sim b \ .$$

- $\rightarrow$ is weakly confluent modulo $\sim$ if

$$\forall a, a', b' \in A : \ (a \twoheadrightarrow a', a \twoheadrightarrow b' \Rightarrow \exists a'', b'' : \ a' \twoheadrightarrow a'', b' \twoheadrightarrow b'', a'' \sim b'') \ .$$

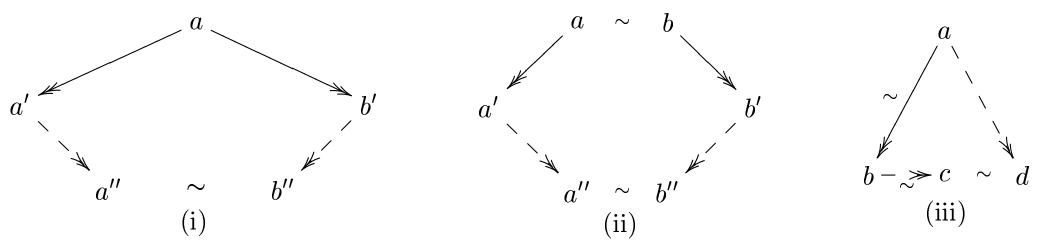
- $\rightarrow$ is confluent modulo $\sim$ if

$$\forall a, a', b, b' \in A : \ (a \sim b, a \twoheadrightarrow a', b \twoheadrightarrow b' \Rightarrow \exists a'', b'' : \ a' \twoheadrightarrow a'', b' \twoheadrightarrow b'', a'' \sim b'') \ .$$

- $\rightarrow$ is complete for $\underset{\sim}{\rightarrow}$ modulo $\sim$ if

$$\forall a, b \in A : \ (a \underset{\sim}{\twoheadrightarrow} b \Rightarrow \exists c, d \in A : \ b \underset{\sim}{\twoheadrightarrow} c, c \sim d, a \twoheadrightarrow d) \ .$$

See Fig. 37 for the pictorial equivalents.

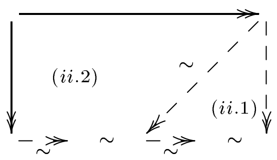


(i) $\to$ is weakly confluent modulo $\sim$
(ii) $\to$ is confluent modulo $\sim$
(iii) $\to$ is complete for $\underset{\sim}{\twoheadrightarrow}$ modulo $\sim$

Figure 37. Pictorial definitions

Proposition 10.8. *Given an ordered ARS $(A, \to, \preceq)$. Define $a \sim b$ if $a \preceq b$ and $b \preceq a$. Then*
(i) $\to$ *is weakly confluent up to $\preceq$ if $\to$ is (weakly) confluent modulo $\sim$.*
(ii) $\to$ *is confluent up to $\preceq$ if*
(ii.1) $\to$ *is complete for $\underset{\sim}{\twoheadrightarrow}$, and*
(ii.2) $\underset{\sim}{\twoheadrightarrow}$ *is confluent modulo $\sim$, and*
(ii.3) $\to$ *is monotonic with respect to $\preceq$.*

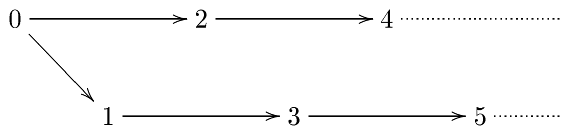
Proof. Point (ii) follows from the following diagram and point (ii.3).



□

Given an ordered ARS $(A, \to, \preceq)$. Let $\sim$ be an equivalence relation such that $\sim \subseteq (\preceq \cap \succeq)$. Then, confluence modulo $\sim$ implies confluence up to $\preceq$. There are however cases where we do have confluence up to $\preceq$ and not confluence modulo $\sim$, as the following example shows:

Example 10.9. Consider the following ARS:

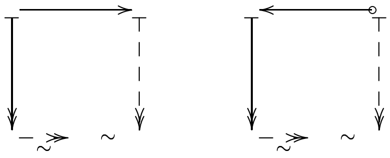


This ARS is not confluent, but it is obviously confluent up to $\leq$ (the natural order on natural numbers). It is immediately obvious that this ARS cannot be confluent modulo $\sim$.

10.2. A technique to prove confluence up to

According to Lemma 10.6 it is sufficient to find a standard reduction $\mapsto \subseteq \to$ that is complete and commutative up to a quasi order to prove that $\to$ is confluent up to the quasi order. Next, we present sufficient conditions that guarantee that these two properties hold.

LEMMA 10.10. *Given an ordered ARS $(A, \to, \preceq)$, a relation $\mapsto \subseteq \to$ and an equivalence $\sim \subseteq \simeq$. If the following two diagrams hold*



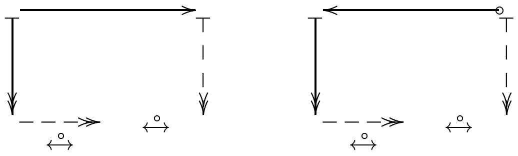
then

(i) $\to$ *commutes with* $\mapsto$ *up to* $\preceq$, *and*

(ii) $\mapsto$ *is complete up to* $\preceq$.

PROOF. (i) Follows from the left diagram. (ii) Follows from the right diagram. □

The proofs of confluence up to in the rest of this paper will also use the fact that $\circ\!\!\to \subseteq \simeq$ and apply the lemma above with $\sim = \overset{\circ}{\leftrightarrow}$. Thus, in each case we prove a lemma stating the following diagrams:

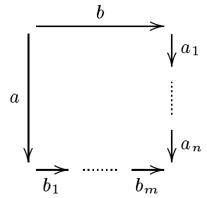


In each case, to prove this lemma, called *reduction lemma*, we use a technique called decreasing diagram technique [vO94]. This technique derives commutativity from local commutativity. It consists of associating a label to each reduction step and giving a well-founded order on these labels. If all local diagrams turn out to be of a specific kind, namely *decreasing*, then commutativity is guaranteed.

DEFINITION 10.11. Let $|.|$ be the measure from strings of labels to multisets of labels defined by:

$$|a_1 \dots a_n| = \{\!\{ a_i | \text{ there is no } j < i \text{ with } a_j > a_i \}\!\} \ .$$

Then, the diagram



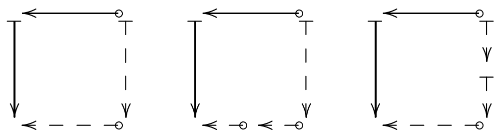
is *decreasing* if $\{\!\{a, b\}\!\} \geq |ab_1 \dots b_m|$ and $\{\!\{a, b\}\!\} \geq |ba_1 \dots a_n|$.

THEOREM 10.12. *If two labeled reduction systems are locally commutative and all local diagrams are decreasing with respect to a well founded order on labels then the systems are commutative.*

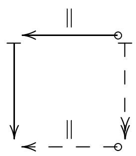
PROOF. See [vO94]. □

We remark that we are free to give any label to any occurrence of a step. It is especially useful to be able to give different labels to the same step occurring horizontally and vertically. However, once we give a label to an occurrence of a step then it is fixed.

The decreasing diagrams technique sometimes fails if there is duplication in both the horizontal and vertical direction, *e.g.,* there is no possible labeling that makes the following diagrams all decreasing:



It is often possible to solve this problem by introducing a form of parallel reduction in, for example, the horizontal direction. With respect to parallel reduction the three diagrams should then collapse into the single diagram



which can be made decreasing by ordering the parallel reduction larger than the standard reduction.

10.3. Infinite normal form

We define the infinite normal form of a term as the maximum information that can be obtained by reducing that term. It may seem strange that the notion of infinite normal form depends on the notion of information content, but all notions of infinitary reduction depend on the specific way of defining the limit construction. We model the information content of a term as a function from the set of objects of an ARS to a partial order. This function and the partial order induce a quasi order on the elements of the ARS. The intuition of information content demands that the rewrite relation is monotonic with respect to the induced order. Formally:

Definition 10.13. The structure $\langle(A, \rightarrow, \preceq), \omega, (B, \leq)\rangle$ is called an ARS with ordered information content if:
- $(A, \rightarrow, \preceq)$ is an ordered ARS
- $\rightarrow$ is monotonic with respect to $\preceq$
- $(B, \leq)$ is a partially order set
- ω is a function $A \rightarrow B$
- $a \preceq b$ iff $\omega(a) \leq \omega(b)$.

We often refer to $\omega(a)$ as the information content of a.

If it is obvious which structure $\mathcal{A} \equiv \langle(A, \rightarrow, \preceq), \omega, (B, \leq)\rangle$ is meant then we say that $\mathcal{A}$ is confluent up to (information content) instead of saying that $\rightarrow$ is confluent up to $\preceq$.

Definition 10.14. Given a partial order $(B, \leq)$. We define the downward closure of a set $C \subseteq B$, denoted by $\downarrow C$, as

$$\{b \in B \mid b \leq a \in C\} \ .$$

Definition 10.15. Given an ARS $\mathcal{A}$ with ordered information content $\langle(A, \rightarrow, \preceq), \omega, (B, \leq)\rangle$.
- The infinite normal form of an $a \in A$ is defined as:

$$\mathrm{Inf}(a) = \downarrow \{\omega(b) \mid a \twoheadrightarrow b\} \ .$$

- $\mathcal{A}$ has unique infinite normal forms if

$$\forall a, b \in A : a \rightarrow b \Rightarrow \mathrm{Inf}(a) = \mathrm{Inf}(b) \ .$$

Next, we show that confluence up to guarantees that the infinite normal form is an ideal. Given a partial order $(B, \leq)$, a subset I of B is an ideal iff (i) I is non-empty, (ii) I is directed: $\forall a, b \in I, \exists c \in I, a \leq c$ and $b \leq c$, (iii) I is downward closed: $\forall c \in I$, if $\exists d \in B, d \leq c$ then $d \in I$.

PROPOSITION 10.16. *Given an ARS with ordered information content $\langle (A, \rightarrow, \preceq), \omega, (B, \leq) \rangle$. If $\rightarrow$ is confluent up to $\preceq$ then* $\mathrm{Inf}(a)$ *is an ideal.*

PROOF. $\mathrm{Inf}(a)$ is non-empty and downward closed by definition; it is directed from confluence up to and monotonicity. $\square$

REMARK 10.17. As pointed out in the above proposition, we do not require $\rightarrow$ to be confluent for the infinite normal form to be an ideal. We require instead monotonicity (which is embedded in the definition of an ARS with ordered information content) and confluence up to.

Confluence up to $\preceq$ is sufficient but not necessary for the infinite normal form to be an ideal. For example, if we have $a \rightarrow b$, $a \rightarrow c$ with $a \prec b \prec c$ then $\mathrm{Inf}(a)$ is still an ideal, but the rewriting system is not confluent up to $\preceq$. However, confluence up to $\preceq$ is a sufficient and necessary condition for the uniqueness of infinite normal forms, as expressed in the next lemma.

LEMMA 10.18. *Given an ARS $\mathcal{A}$ with ordered information content $\langle (A, \rightarrow, \preceq), \omega, (B, \leq) \rangle$. $\rightarrow$ is confluent up to $\preceq$ iff $\mathcal{A}$ has the unique infinite normal form property.*

PROOF. $(\Rightarrow)$ Trivially $\mathrm{Inf}(b) \subseteq \mathrm{Inf}(a)$. Let $a' \in \mathrm{Inf}(a)$ be given. There exists an a'' such that $a \twoheadrightarrow a''$ with $a' \leq \omega(a'')$. By confluence up to $\preceq$ there exists b' such that $b \twoheadrightarrow b'$ with $a'' \preceq b'$ and therefore $a' \leq \omega(b')$ so $a' \in \mathrm{Inf}(b)$. $(\Leftarrow)$ Trivial. $\square$

LEMMA 10.19. *Given an ARS with ordered information content $\langle (A, \rightarrow, \preceq), \omega, (B, \leq) \rangle$. Let $(A, \leq_A)$ be an ARS. If $(A, \leq_A)$ commutes with $(A, \rightarrow)$, and $\leq_A$ is monotonic with respect to $\preceq$ then*

$$a \leq_A a' \Longrightarrow \mathrm{Inf}(a) \subseteq \mathrm{Inf}(a') \ .$$

PROOF. Suppose $a \twoheadrightarrow b$ then (assuming w.l.o.g. that $\leq_A$ is transitive) by commutativity we have that there exists a $b' \in A$ such that $a' \twoheadrightarrow b'$ and $b \leq_A b'$. Because $\leq_A$ is monotonic with respect to $\preceq$ we have that $b \preceq b'$. $\square$

In the following two sections we are going to give two applications of the use of confluence up to information content. We start by studying $\mathcal{R}\circ^{\rightarrow}$ (given in Table 4, and not to be confused with $\mathcal{R}_0$ given in the previous part) and then we return to $\lambda\circ^{\rightarrow}_{\mathrm{name}}$. In both cases we will use the technique described in Section 10.2.

11. Tree unwinding as an infinite normal form

In this section we consider the rewriting system $\mathcal{R}\circ^{\rightarrow}$, as defined in Table 4. We first define the information content of a term. The notion of information content is similar to the *approximate normal form* of Wadsworth [Wad71], also called *direct approximation* by Lévy [Lév78], but considers substitution redexes rather than β-redexes. Thus, we obtain a form of unwinding of the term rather than a Böhm tree. As described in [Lév78], the information content is obtained by first replacing the redexes by Ω, and then sending compatible redexes to Ω. A compatible redex is not a redex, but could become so by replacing Ω with some other term. For example, in lambda calculus ΩM is a compatible redex because it could become a β-redex by replacing Ω with a lambda abstraction. In our setting, we do not send all redexes and compatible redexes to Ω, since that would mean sending every term that contains an Ω to Ω. In fact, suppose a term N is of the form $C[\Omega]$, then by induction on the context

N could be Ω or one of the following $M\ C'[\Omega], C'[\Omega]\ M, \lambda x.C'[\Omega], \langle C'[\Omega] \mid D\rangle, \langle M \mid x = C'[\Omega], D\rangle$. By induction hypothesis we would obtain:

$$M\ \Omega, \Omega\ M, \lambda x.\Omega, \langle \Omega \mid D\rangle, \langle M \mid x = \Omega, D\rangle\ .$$

Now, the first three terms are compatible lift redexes, the others compatible merge redexes. Hence, the information content of N is Ω.

Even though, in general, giving every term the same information content makes the system an ARS with ordered information content that is confluent up information content, it does not tell us anything useful because every term is equated.

In deciding what is the information content of a term, we first divide the reduction rules in two subsets: the rules that are administrative in nature and those that do the real work. Since in here the final goal is to compute the possible infinite unwinding of a term, the only rule that corresponds to an increase in information content is the external substitution rule. However, we still need to be careful. Consider the reduction

$$\langle x \mid x = \lambda y.x\rangle \to \langle \lambda y.x \mid x = \lambda y.x\rangle \to \langle \lambda y.\lambda y.x \mid x = \lambda y.x\rangle \to \cdots$$

If we send all external substitution redexes to Ω then every term in the sequence would be sent to Ω. Instead, we want the above reduction to correspond to the following increasing chain:

$$\Omega \le \lambda y.\Omega \le \lambda y.\lambda y.\Omega \le \cdots$$

This can be accomplished by sending to Ω all the occurrences of the variables that could be replaced in an external substitution step, and then removing all the inaccessible equations from the environment. This leads to the following definition.

Definition 11.1. Given $M \in \Lambda\circ$. The $\mathcal{R}\circ^{\rightarrow}$-information content of M is the normal form of M with respect to the rule:

$$\langle M \mid x_1 = M_1, \cdots, x_n = M_n\rangle \xrightarrow[\text{strip}]{} M[x_1 := \Omega, \cdots, x_n := \Omega]\ .$$

We denote this normal form by $\text{strip}(M)$.

The strip function maps cyclic terms to plain lambda calculus terms that include the constant Ω. We still denote this set of lambda terms by Λ. We order these lambda terms and also cyclic lambda terms with the order $\le_\Omega$, which is generated by the axiom $\Omega \le_\Omega M$, for every term M. We can also describe the order $\le_\Omega$ by the single rule rewriting system $M \xrightarrow[\Omega]{} \Omega$, *i.e.,* $M \le_\Omega N$ iff $N \twoheadrightarrow_{\Omega} M$. Given a set of (cyclic) terms S, $(S, \le_\Omega)$ is a partial order.

We now have a set of terms $\Lambda\circ$ (which includes Ω as a constant), a rewriting relation induced by the rewriting system $\mathcal{R}\circ^{\rightarrow}$ (written as $\xrightarrow[\mathcal{R}\circ]{}$), a function that computes information content (strip) and an order to compare information contents ($\le_\Omega$). Together they form an ARS with ordered information content:

Proposition 11.2. $\langle(\Lambda\circ, \xrightarrow[\mathcal{R}\circ]{}, \preceq_{\mathcal{R}\circ}), \text{strip}, (\text{strip}(\Lambda\circ), \le_\Omega)\rangle$ *is an ARS with ordered information content.*

Proof. Trivial. □

To prove confluence up to information content for $\mathcal{R}\circ^{\rightarrow}$, we use the technique of Section 10.2. We start by introducing the notion of standard reduction, which is complete with respect to information content. The idea behind standard reduction is that we take a simple subset of the rewriting system, such that only standard steps can increase information. Common features of all notions of standard reduction defined in this paper are that a standard redex is never duplicated or destroyed by any other redex and that only a standard redex can create a new standard redex.

DEFINITION 11.3. Given $M, N \in \Lambda\circ$. M standard rewrites to N (written as $M \underset{\mathcal{R}\circ}{\longmapsto} N$) if:

$$M \equiv E[\langle E[x] \mid x = M, D\rangle] \underset{\mathcal{R}\circ}{\longmapsto} E[\langle E[M] \mid x = M, D\rangle] \equiv N \ ,$$

where

$$E ::= \square \mid E\ M \mid M\ E \mid \lambda x.E \mid \langle E \mid D\rangle \ .$$

A non-standard step cannot change the information content:

LEMMA 11.4. *Given $M, N \in \Lambda\circ$. If $M \circ\!\!\underset{\mathcal{R}\circ}{\longrightarrow} N$ then M and N have the same information,* i.e., *$strip(M) \equiv strip(N)$.*

PROOF. For all redexes except external substitution the result is trivial. If we have a non-standard external substitution step

$$C_1[\langle C_2[x] \mid x = M, D\rangle] \circ\!\!\rightarrow C_1[\langle C_2[M] \mid x = M, D\rangle]$$

then either C_1 or C_2 is not an evaluation context. That is, either C_1 or C_2 is of the form $C'[\langle P \mid y = C'', D'\rangle]$. For both cases we obviously have that the information content does not change. $\square$

Before studying the interaction between standard reduction and normal reduction, we make the following observation.

PROPOSITION 11.5. $\langle C[\langle M \mid D\rangle] \mid x = \langle M \mid D\rangle, D'\rangle \underset{\mathcal{R}\circ}{\longleftrightarrow} \langle C[M] \mid x = M, D, D'\rangle$.

PROOF. By an induction similar to the proof of Lemma 8.1 we can prove that

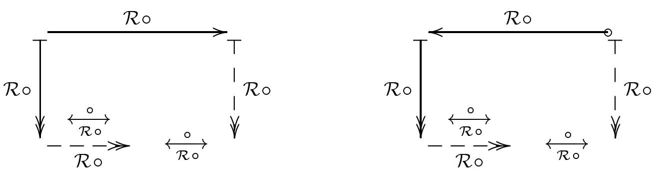
$$C[\langle M \mid D\rangle] \underset{\mathcal{R}\circ}{\longleftrightarrow} \langle C[M] \mid D\rangle \ .$$

We then have

$$\begin{array}{lcl} \langle\langle C[M] \mid D\rangle \mid x = \langle M \mid D\rangle, D'\rangle & \underset{\text{im}}{\longrightarrow} & \langle\langle C[M] \mid D\rangle \mid x = M, D, D'\rangle \\ & \underset{\text{em}}{\longrightarrow} & \langle C[M] \mid D, x = M, D, D'\rangle \\ & \underset{\text{cp}}{\longleftarrow} & \langle C[M] \mid x = M, D, D'\rangle \end{array} \ .$$

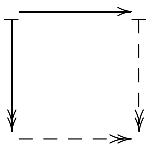
$\square$

LEMMA 11.6. (REDUCTION LEMMA) *We have the following two diagrams:*

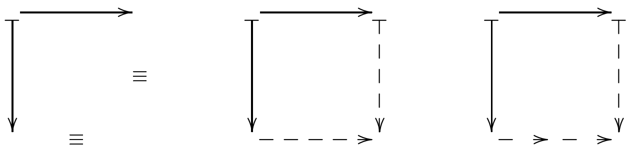


PROOF. We reason by cases on the top step. For simplicity we omit the subscript $\mathcal{R}\circ$ from the reduction relation.

- *Left diagram for internal and external substitution.* The result follows from the following diagram:



We can prove this diagram by tiling with the elementary diagrams below:

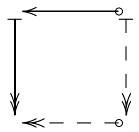


These elementary diagrams cover all cases of interaction between a standard step and a substitution step: The top step could be disjoint from the standard step (middle diagram), could be the same (left diagram) or could be nested inside the term substituted by the standard step (right diagram). This last situation is depicted below:

$$\begin{tikzcd} E[\langle E[x] \mid x = C[y], y = M, D\rangle] \arrow[r] \arrow[d] & E[\langle E[x] \mid x = C[M], y = M, D\rangle] \\ E[\langle E[C[y]] \mid x = C[y], y = M, D\rangle] & \end{tikzcd}$$

Even if $x \equiv y$ we can close this diagram by reducing to $E[\langle E[C[C[x]]] \mid x = C[C[x]], D\rangle]$.

- *Right diagram for internal and external substitution.* The result follows from the fact that a non-standard reduction does not increase information content (Lemma 11.4) and the following claim:



(11.1)

Before we can prove this claim we need to introduce some auxiliary notions. First, we must distinguish between two kinds of substitution rules: cyclic substitution (cs) and unnested substitution (us). Second, we need to define complete developments of disjoint sets of cyclic and unnested substitutions.

A cyclic substitution is an internal substitution of the form

$$\langle M \mid x = C[x], D\rangle \to \langle M \mid x = C[C[x]], D\rangle \ .$$

Two cyclic substitution steps are disjoint if the involved equations are syntactically disjoint, *e.g.*, in $\langle x \mid x = \underline{x}\underline{y}, y = \underline{y}\rangle$ the underlined variables refer to disjoint cyclic redexes. Instead, in $\langle x \mid x = \underline{x}\underline{x}, y = \langle C[\underline{y}] \mid z = C_1[\underline{z}]\rangle\rangle$ the two underlined variables x refer to two non-disjoint cyclic redexes, the same holds for the underlined variables y and z. A parallel cyclic substitution step ($\|$cs) is the reduction of a set of disjoint cyclic substitution redexes.

An unnested substitution step is a substitution step (either external or internal) of the form

$$\langle M \mid x = C[y], y = N, D\rangle \to \langle M \mid x = C[N], y = N, D\rangle \ ,$$

where $x \not\equiv y$. Two unnested substitution redexes are disjoint if the redexes do not occur in the terms we substitute, *e.g.*, in $\langle M \mid x = C[\underline{y}], y = C[\underline{z}], z = P\rangle$ the underlined variables y and z denote two unnested substitution redexes that are not disjoint since redex z occurs in the term we substitute for y. In $\langle \underline{y} \mid y = z, w = \underline{x}\ \underline{x}, x = z\rangle$ the underlined variables x and y form a set of three disjoint unnested substitution redexes. A parallel unnested substitution step ($\|$us) is the reduction of a set of disjoint unnested substitution steps. Every single substitution step is either a cyclic substitution step or an unnested substitution step.

The claimed diagram 11.1 then follows from the following diagrams:

$$
\begin{tikzcd}
{} \arrow[d] & {} \arrow[l, "\|\mathrm{cs}"'] \arrow[d, dashed] \\
{} & {} \arrow[l, dashed, "\|\mathrm{cs}"]
\end{tikzcd}
\quad
\begin{tikzcd}
{} \arrow[dd] & {} \arrow[l, "\|\mathrm{cs}"'] \arrow[d, dashed] \\
 & {} \arrow[d, dashed] \\
{} & {} \arrow[l, dashed, "\|\mathrm{cs}"]
\end{tikzcd}
\quad
\begin{tikzcd}
{} \arrow[d] & & {} \arrow[ll, "\|\mathrm{cs}"'] \arrow[d, dashed] \\
{} & {} \arrow[l, dashed, "\|\mathrm{cs}"] & {} \arrow[l, dashed, "\|\mathrm{us}"]
\end{tikzcd}
\quad
\begin{tikzcd}
{} \arrow[d] & {} \arrow[l, "\|\mathrm{us}"'] \arrow[d, dashed, two heads] \\
{} & {} \arrow[l, dashed, "\|\mathrm{us}"]
\end{tikzcd}
\tag{11.2}
$$

These diagrams follow from analyzing the interaction between parallel cyclic substitution, parallel unnested substitution and standard substitution:

- The relation between a parallel cyclic substitution and a standard substitution step can be one of the following:
  - The redexes are disjoint. Trivially we fall under the first diagram of 11.2.
  - The standard redex replaces x with M and some of the cyclic redexes occur inside M. Because the replacement of x is a standard step (*i.e.,* it never occurs in the right-hand side of an equation) it cannot happen that the copy of M, made by the standard step occurs inside the N of a cyclic substitution redex $y = N$. That is, the following rewriting would not happen

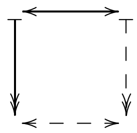
    $$\langle y \mid y = \langle z \mid z = C[\underline{y}]\rangle\rangle \to \langle y \mid y = \langle C[\underline{y}] \mid z = C[\underline{y}]\rangle\rangle \ .$$

    Note that the descendants of the cyclic redex y with respect to the above rewriting are not disjoint. However, the descendants of cyclic redexes with respect to a standard reduction are disjoint. Therefore, they can be contracted in a single parallel step. So again we fall under the first diagram of 11.2.
  - We have a term of the from $C_1[\langle C_2[x^1] \mid x = C_3[x^2], D\rangle]$, where the x superscripted with 1 is the variable replaced by contracting the standard redex and the x superscripted with 2 is a variable replaced by one of the cyclic steps. This means that to close we have to do the standard substitution step, another substitution step and then the parallel cyclic substitution step. This falls under the second or third diagram of 11.2, depending on the extra step being standard or not. *I.e.,*

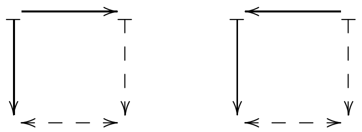
    $$
    \begin{tikzcd}
    \langle x \mid x = \lambda y.(\lambda y.x)\rangle \arrow[dd] & \langle x \mid x = \lambda y.x\rangle \arrow[l] \arrow[d, dashed] \\
     & \langle \lambda y.x \mid x = \lambda y.x\rangle \arrow[d, dashed] \\
    \langle \lambda y.(\lambda y.x) \mid x = \lambda y.(\lambda y.x)\rangle & \langle \lambda y.(\lambda y.x) \mid x = \lambda y.x\rangle \arrow[l, dashed]
    \end{tikzcd}
    $$

- Given a parallel unnested substitution step consisting of only non-standard steps and a standard step we are in one of the following cases (see the last diagram of 11.2):
  - The standard step replaces a variable x with a term M and some of the unnested steps replace variables inside M. The descendants of the unnested redexes with respect to the standard reduction are still disjoint redexes, and thus we may close the diagram by first doing the standard ones and then in one parallel step the non-standard ones.
  - The standard step can be added to the set of disjoint unnested redexes and we close like in the previous case.

- *Both diagrams for the remaining rules.* The result follows from the fact that none of these steps can increase information content (Lemma 11.4) and from the following claim:



where we do not have substitutions in the horizontal direction. This claim follows from the following two elementary diagrams:



For all rules, except internal merge, these diagrams follow from the fact that there are no critical pairs, and that the rules in the horizontal direction cannot duplicate a standard redex. A critical pair between internal merge and standard substitution is shown below:

$$\begin{tikzcd} \langle C[x] \mid x = \langle M \mid D\rangle, D'\rangle \arrow[r] \arrow[d, mapsto] & \langle C[x] \mid x = M, D, D'\rangle \arrow[d, mapsto] \\ \langle C[\langle M \mid D\rangle] \mid x = \langle M \mid D\rangle, D'\rangle \arrow[r, leftrightarrow, dashed] & \langle C[M] \mid x = M, D, D'\rangle \end{tikzcd}$$

The bottom conversion follows from Proposition 11.5.

□

LEMMA 11.7. *Given* $\langle(\Lambda\circ, \xrightarrow[\mathcal{R}\circ]{}, \preceq_{\mathcal{R}\circ}), \mathrm{strip}, (\mathrm{strip}(\Lambda\circ), \leq_\Omega), \xrightarrow[\mathcal{R}\circ]{}\rangle$.

(i) $\underset{\mathcal{R}\circ}{\longmapsto}$ *is complete up to* $\preceq_{\mathcal{R}\circ}$.

(ii) $\underset{\mathcal{R}\circ}{\longmapsto}$ *commutes with* $\xrightarrow[\mathcal{R}\circ]{}$ *up to* $\preceq_{\mathcal{R}\circ}$.

PROOF. From Lemmas 11.6 and 10.10. □

THEOREM 11.8. *Given* $\langle(\Lambda\circ, \xrightarrow[\mathcal{R}\circ]{}, \preceq_{\mathcal{R}\circ}), \mathrm{strip}, (\mathrm{strip}(\Lambda), \leq_\Omega)\rangle$. $\xrightarrow[\mathcal{R}\circ]{}$ *is confluent up to* $\preceq_{\mathcal{R}\circ}$.

PROOF. Follows from Lemmas 11.7 and 10.6. □

COROLLARY 11.9. $\langle(\Lambda\circ, \xrightarrow[\mathcal{R}\circ]{}, \preceq_{\mathcal{R}\circ}), \mathrm{strip}, (\mathrm{strip}(\Lambda), \leq_\Omega)\rangle$ *has the unique normal form property.*

PROOF. From the above theorem and Lemma 10.18. □

In Section 15, we will define unwinding of cyclic terms to the terms of the infinitary lambda calculus, and show that the unwinding of any cyclic term M is the least upper bound of the infinite normal form computed with respect to $\langle(\Lambda\circ, \xrightarrow[\mathcal{R}\circ]{}, \preceq_{\mathcal{R}\circ}), \mathrm{strip}, (\mathrm{strip}(\Lambda), \leq_\Omega)\rangle$ (written as $\mathrm{Inf}_{\mathcal{R}\circ\rightarrow}(M)$).

$$
\begin{array}{lll}
\beta\circ: & & \\
(\lambda x.M)N & \xrightarrow[\text{eval}]{} & \langle M \mid x = N\rangle \\
\textit{External substitution:} & & \\
\langle C[x] \mid x = M, D\rangle & \xrightarrow[\text{eval}]{} & \langle C[M] \mid x = M, D\rangle \\
\textit{Lift:} & & \\
\langle M \mid D\rangle N & \xrightarrow[\text{eval}]{} & \langle MN \mid D\rangle
\end{array}
$$

Table 5. Evaluation calculus: $\lambda\circ_{\text{eval}}$

12. Basic properties of the cyclic lambda calculus

We now turn to the rewriting system $\lambda\circ^{\rightarrow}$. As in the previous section, we show that $\lambda\circ^{\rightarrow}$ is an abstract reduction system with ordered information that is confluent up to information content. We will prove the latter by defining a notion of standard reduction that is complete and commutative up to information content.

As for $\mathcal{R}\circ^{\rightarrow}$, in computing the information content of a term we do not send all redexes to Ω. $\beta\circ$ and the occurrences of variables that correspond to external substitution redexes are sent to Ω. As in lambda calculus, ΩM is also sent to Ω. The inaccessible equations are then removed.

Definition 12.1. Given $M, N \in \Lambda\circ$. The information content of M is given by the function ω, which given M returns the normal form of M with respect to the following rules (also called ω-rules):

$$
\begin{array}{rcllr}
(\lambda x.M)N & \xrightarrow[\omega]{} & \Omega & & \beta\omega \\
\langle C[x] \mid x = M, D\rangle & \xrightarrow[\omega]{} & \langle C[\Omega] \mid x = M, D\rangle & & es\omega \\
\Omega M & \xrightarrow[\omega]{} & \Omega & & @\omega \\
\langle M \mid D\rangle & \xrightarrow[\omega]{} & M & D \perp M & \text{gc}\omega
\end{array}
$$

We define $M \preceq_{\text{name}} N$ if $\omega(M) \leq_{\Omega} \omega(N)$.

The ω-function is well defined due to the termination and confluence of the ω-rules (due to Newman's lemma). Termination of the ω-rules follows from counting the number of non-Ω symbols in every term. Local confluence follows from a simple check.
Examples: $\omega(\langle\lambda x.yz \mid y = I\rangle) = \lambda x.\Omega$, $\omega(\langle x \mid x = x\rangle) = \Omega$, $\omega(\langle xy \mid y = I\rangle x) = (x\Omega)x$, and $\omega(\langle xx \mid x = I\rangle) = \Omega$. Note that even though $\langle xy \mid y = I\rangle x$ is a lift redex, its information content is not Ω.

Proposition 12.2.

(i) *Given* $(\Lambda\circ, \leq_{\Omega}, \preceq_{\text{name}})$. $\leq_{\Omega}$ *is monotonic with respect to* $\preceq_{\text{name}}$.

(ii) *Given* $(\Lambda\circ, \xrightarrow[\lambda\circ]{}, \preceq_{\text{name}})$. $\xrightarrow[\lambda\circ]{}$ *is monotonic with respect to* $\preceq_{\text{name}}$.

Proof.

(i) Let us write $N \xrightarrow[\Omega]{}\!\!\!\!\rightarrow M$ and $N \xrightarrow[\omega]{}\!\!\!\!\rightarrow \omega(N)$. If we can show that $\xrightarrow[\Omega]{}\!\!\!\!\rightarrow$ commutes with $\xrightarrow[\omega]{}\!\!\!\!\rightarrow$ then there exists an M' such that $\omega(N) \xrightarrow[\Omega]{}\!\!\!\!\rightarrow M'$ and $M \xrightarrow[\omega]{}\!\!\!\!\rightarrow M'$. From $M' \xrightarrow[\omega]{}\!\!\!\!\rightarrow \omega(M)$ it then follows that $\omega(M) \leq_{\Omega} \omega(N)$. Commutativity follows from the following diagrams:

$$
\begin{tikzcd}
\cdot \arrow[r, "\Omega"] \arrow[d, "\beta\omega"'] & \cdot \arrow[d, dashed, "{\beta\omega,@\omega,\equiv}"] \\
\cdot \arrow[r, dashed, "{\Omega,\equiv}"'] & \cdot
\end{tikzcd}
\quad
\begin{tikzcd}
\cdot \arrow[r, "\Omega"] \arrow[d, "\text{es}\omega"'] & \cdot \arrow[d, dashed, "{\text{es}\omega,\equiv}"] \\
\cdot \arrow[r, dashed, "{\Omega,\equiv}"'] & \cdot
\end{tikzcd}
\quad
\begin{tikzcd}
\cdot \arrow[r, "\Omega"] \arrow[d, "@\omega"'] & \cdot \arrow[d, dashed, "{@\omega,\equiv}"] \\
\cdot \arrow[r, dashed, "{\Omega,\equiv}"'] & \cdot
\end{tikzcd}
\quad
\begin{tikzcd}
\cdot \arrow[r, "\Omega"] \arrow[d, "\text{gc}\omega"'] & \cdot \arrow[d, dashed, "{\text{gc}\omega,\equiv}"] \\
\cdot \arrow[r, dashed, "{\Omega,\equiv}"'] & \cdot
\end{tikzcd}
$$

where a comma between labels means or.

(ii) If the reduction is non-standard then the result is obvious. Otherwise, let $M \equiv C[R] \underset{C}{\mapsto} [R'] \equiv N$. If R is a lift redex then $M \simeq_{\text{name}} N$. For R a $\beta\circ$-redex or an external substitution redex we have $\omega(M) = \omega(C[\Omega])$ and $C[\Omega] \leq_\Omega N$, the result then follows from the first item.

□

PROPOSITION 12.3. $\langle(\Lambda\circ, \underset{\lambda\circ}{\longrightarrow}, \preceq_{\text{name}}), \omega, (\omega(\Lambda\circ), \leq_\Omega)\rangle$ *is an ARS with ordered information.*

PROOF. Follows from the previous proposition. □

Next, we present a standard reduction strategy that is complete with respect to information content. We restrict the system to the evaluation system of Table 5. We need $\beta\circ$ and external substitution because those two rules potentially increase information content. We need lift to expose $\beta\circ$-redexes that are implicit. We then restrict the standard redex so that it cannot occur in the internal part D of a construct $\langle M \mid D\rangle$. We also disallow reduction of redexes that could be moved into an environment by contracting a $\beta\circ$-redex. This guarantees confluence of the standard reduction.

We illustrate the basic idea behind the formal definition of standard reduction through an example. Let us consider the term $\lambda x.(\langle x \mid y = y\rangle(\langle \lambda w.w \mid y = y\rangle z))$. We start by looking for a standard redex at the outermost position. Since we find a lambda, we look inside it and find a lift redex, which we could neglect since the redex is not obstructing a $\beta\circ$-redex. Assuming we do neglect it, we next look at the left argument of the application. Since variable x is a lambda bound variable, it can never become a redex, so we start looking in the right-hand side of the application. We again find a lift redex. This time we must reduce it because it is obstructing a $\beta\circ$-redex. This informal discussion points out how we split lift redexes into two categories: those redexes that (will in the future) obstruct a $\beta\circ$-redex (*e.g.,* $\langle(\lambda x.M) \mid D\rangle N$ and $\langle\langle y \mid D\rangle N \mid y = \lambda x.M\rangle$), and those that never will (*e.g.,* $\langle y \mid D\rangle M$). The redexes in the first category must be reduced and the other ones can be delayed.

DEFINITION 12.4. Given $M, N \in \Lambda\circ$. M standard rewrites to N (written as $M \underset{\lambda\circ}{\mapsto} N$) if:

$$M \equiv E[R] \underset{\lambda\circ}{\longrightarrow} E[R'] \equiv N \ ,$$

where R and R' stand for a $\lambda\circ_{\text{eval}}$-redex and its contractum, and E is defined as follows:

$$\begin{array}{lcl} E & ::= & \lambda x.E \mid \langle E \mid D\rangle \mid App[y, M_1, \cdots M_{i-1}, E, M_{i+1}, \cdots, M_n] \mid App[\Box, M_1, \cdots, M_n] \\ App & ::= & \Box \mid App\Box \mid \langle App \mid D\rangle \end{array}$$

where the y must be bound by a lambda or free in the final expression.

EXAMPLE 12.5. $\lambda x.(\langle x \mid y = y\rangle(\langle \lambda w.w \mid y = y\rangle z))$ is partitioned as $E_1[(\langle x \mid y = y\rangle(\langle \lambda w.w \mid y = y\rangle z))]$, where E_1 is $\lambda x.\Box$, or as $E_2[\langle \lambda w.w \mid y = y\rangle z]$, where E_2 is $\lambda x.\langle x \mid y = y\rangle\Box$. $\langle x \mid x = y\rangle y$ is partitioned as $E_3[x]$, where E_3 is $\langle\Box \mid x = y\rangle y$. The redex II in $\langle x \mid x = y\rangle(II)$ is not a standard redex, since $\langle x \mid x = y\rangle\Box \neq E[\Box]$. To make it standard, an external substitution step must first take place.

PROPOSITION 12.6. *Standard reduction is confluent.*

PROOF. Suppose the term M can be factored into two different evaluation contexts and redexes: $E_1[R_1]$ and $E_2[R_2]$. We will use structural induction on the context E_1 to show that the two redexes commute in precisely one step:

- $E_1 \equiv \lambda x.E_1'$ or $E_1 \equiv \langle E_1' \mid D\rangle$: The context E_2 must be of the same form and we can use the induction hypothesis.

- $E_1 \equiv App[x, M_1, \cdots, E_1', \cdots, M_m]$: Now E_2 must be of the form $App[\Box, N_1, \cdots, N_n]$ or of the form $App[x, N_1, \cdots, E_2', \cdots, N_n]$. In the first case, R_2 is a lift-redex that commutes in one step with R_1. In the second case, let i be the index where E_1' occurs and let j be the index where E_2' occurs. If $i = j$ we use the induction hypothesis. If $i \neq j$ then the redexes are disjoint and commute in one step.
- $E_1 \equiv App[\Box, M_1, \cdots, M_m]$: The context E_2 must be of the form $App[\Box, N_1, \cdots, N_n]$ or of the form $App[x, N_1, \cdots, E_2', \cdots, N_n]$. In the first case, we know that R_1 or R_2 is a lift-redex and that they commute in one step. In the second case, we know that R_1 is a lift-redex that commutes in one step with R_2.

$\Box$

Proposition 12.7. *Given $M, N \in \Lambda\circ$. If $M \underset{\lambda\circ}{\circ\!\!\longrightarrow} N$ then $M \simeq_{\text{name}} N$.*

Proof. All of the rules, except $\beta\circ$ and external substitution, only modify the environment or where the environment occurs. Since all environments will eventually become garbage and then be removed, we have that $M \simeq_{\text{name}} N$ in those cases. To deal with $\beta\circ$ and external substitution we first describe each context $C[\Box]$ as follows:

$$\begin{array}{rcl} C & ::= & App[\Box, M, \cdots, M] \mid \lambda x.C \mid \langle C \mid D\rangle \mid \langle M \mid x = C, D\rangle \mid App[x, M, \cdots, C, \cdots, M] \\ & \mid & App[\lambda x.M, M, \cdots, C, \cdots, M] \mid App[\lambda x.C, M_1, \cdots, M_n] \\ App & ::= & \Box \mid App\Box \mid \langle App \mid D\rangle \end{array}$$

where in the last production of the first clause $n \geq 1$.

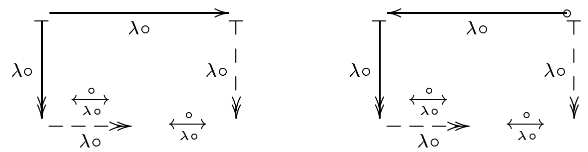
Since the redex reduced in the reduction from M to N is non-standard it means that M is of the form $C[C']$, where C' is one of the following:

- $C' \equiv \langle M \mid x = C'', D\rangle$
- $C' \equiv App[x, M, \cdots, C'', \cdots, M]$, where x is an external substitution redex
- $C' \equiv App[\lambda x.M, M, \cdots, C'', \cdots, M]$
- $C' \equiv App[\lambda x.C'', M_1, \cdots, M_n]$, where $n \geq 1$.

In the first case, the redex only modifies the environment, so the information content does not change because the ω-rules throw away all environments. In the other cases, the ω-rules will send $C[C'[P]]$ to $C[\Omega]$, for any P. Therefore, the information content of M and N is the same. $\Box$

Next, we study the interaction of standard reduction with full reduction.

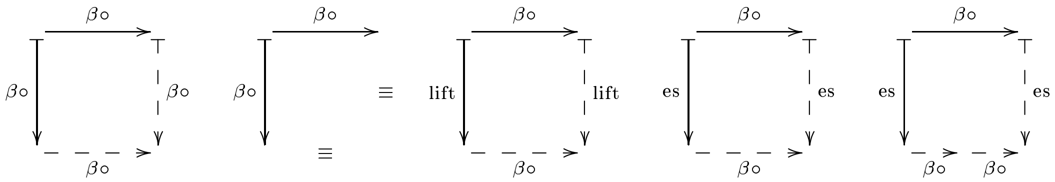
Lemma 12.8. (Reduction lemma) *We have the following two diagrams:*



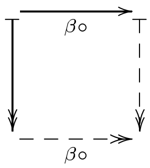
Moreover, in the left diagram the number of $\beta\circ$/es-steps in the right standard reduction is not greater than the number of $\beta\circ$/es-steps in the left standard reduction. The number of $\beta\circ$/es-steps in the right standard reduction is smaller if a descendant of the top $\beta\circ$, es or is-redex is contracted in the standard reduction.

Proof. We will prove these diagrams by cases on the top step.

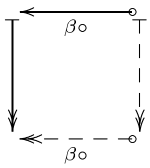
- *Left diagram for $\beta\circ$.* Given any $\beta\circ$-redex and a standard $\beta\circ$-redex we have that these redexes commute in at most one step by descendants. A standard lift redex will commute in exactly one step with any given $\beta\circ$-redex. A standard substitution step could duplicate a $\beta\circ$-redex if the redex is inside the term that is substituted. Moreover, the descendant of a standard redex along a $\beta\circ$-redex is going to be a standard redex again. Hence, we have the following diagrams:



From the above diagrams we derive the following:

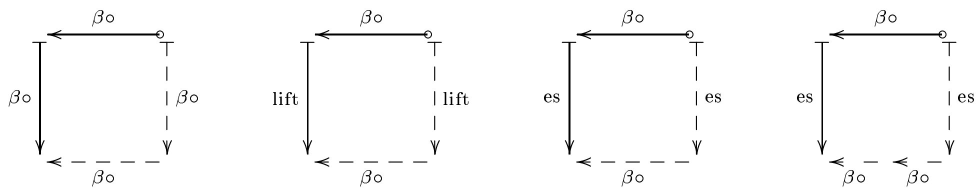


- *Right diagram for $\beta\circ$.* The result follows from the fact that a non-standard step does not increase information (Proposition 12.7) and the following claim:

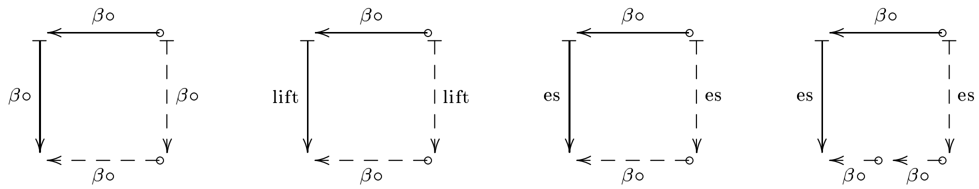


(12.1)

Due to the non-standardness of the top step we are able to find a redex on the right-hand side whose descendant is the given step. We can then reuse the local diagrams from the previous case to obtain:



So far however we do not know if the bottom steps are standard or non-standard. Since we need non-standard steps at the bottom to tile with these local diagrams, we further analyze the diagrams and get:



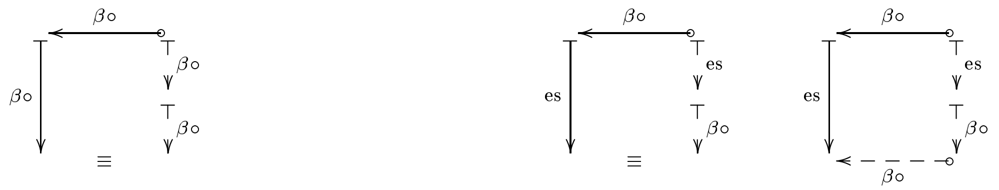
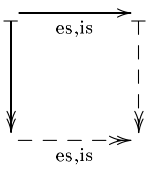
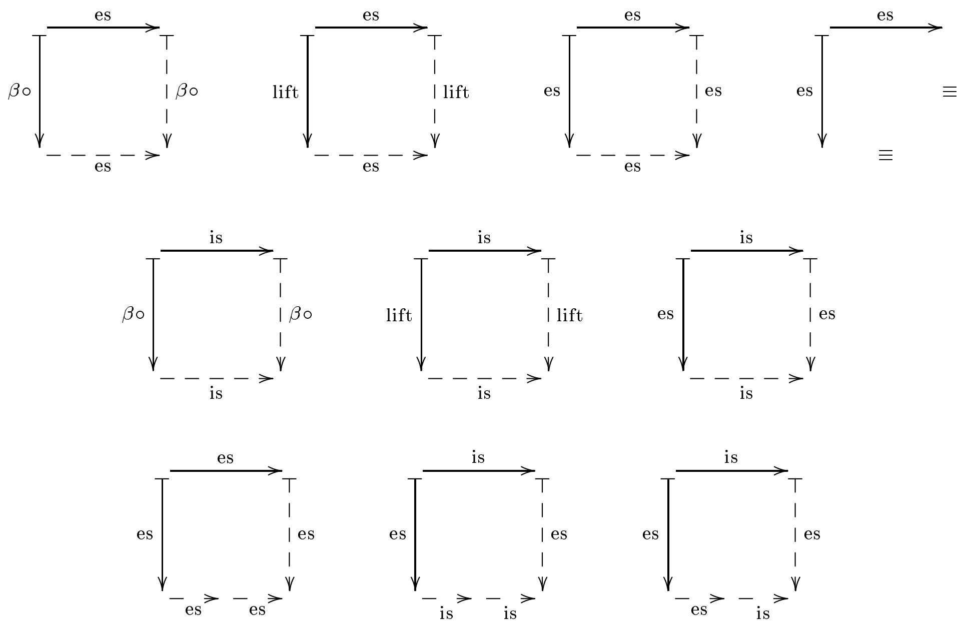


Diagram 12.1 follows by applying the decreasing diagrams theorem (Theorem 10.12) with the ordering: $\downarrow_{\text{es}} > \leftarrow_{\beta\circ} > \downarrow_{\beta\circ}, \downarrow_{\text{lift}}$.

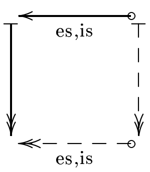
- *Left diagram for internal and external substitution.* The result follows from the following claim:



This claim follows from the following diagrams:



- *Right diagram for internal and external substitution.* The result follows from the fact that a non-standard step does not increase information (Proposition 12.7) and the following claim:



(12.2)

This claim follows from the following four diagrams

$$
\begin{tikzcd}
\bullet \arrow[d, "\mathrm{es}"'] & \bullet \arrow[l, "\mathrm{is,es}"'] \arrow[d, dashed, "\mathrm{es}"] \\
\bullet & \bullet \arrow[l, dashed, "\mathrm{is}\cup\mathrm{es}"]
\end{tikzcd}
\quad
\begin{tikzcd}
\bullet \arrow[d, "\beta\circ"'] & \bullet \arrow[l, "\mathrm{is,es}"'] \arrow[d, dashed, "\beta\circ"] \\
\bullet & \bullet \arrow[l, dashed, "\mathrm{is,es}"]
\end{tikzcd}
\quad
\begin{tikzcd}
\bullet \arrow[dd, "\beta\circ"'] & & \bullet \arrow[ll, "\mathrm{es}"'] \arrow[d, "\beta\circ"] \\
& & \bullet \arrow[d, "\mathrm{es}"] \\
\bullet & \equiv & \bullet
\end{tikzcd}
\quad
\begin{tikzcd}
\bullet \arrow[d, "\mathrm{lift}"'] & \bullet \arrow[l, "\mathrm{is,es}"'] \arrow[d, dashed, "\mathrm{lift}"] \\
\bullet & \bullet \arrow[l, dashed, "\mathrm{is,es}"]
\end{tikzcd}
\tag{12.3}
$$

The first diagram of 12.3 can be proven by applying the proof strategy of the corresponding case in the proof of Lemma 11.6. There we defined the notions of parallel unnested and cyclic substitution steps and proved the diagrams in 11.2. We can prove the diagrams in 11.2 for the current version of standard substitution by the same arguments as were used in that proof. These diagrams may be combined to prove the first diagram of 12.3.

The other diagrams of 12.3 reflect that we can reorder any standard $\beta\circ$/lift step and any substitution step. They also reflect that as a result of the reordering the substitution step can only become standard if the other step is a $\beta\circ$ step.

By tiling we can then prove diagram 12.2. Note that it is vital that the standard reduction on the left of the first diagram of 12.3 is a multi-step reduction. Without it we would not be able to show termination of the tiling procedure.

- *Both diagrams at the same time for all other rules.* In all diagrams for this case we have that substitution and $\beta\circ$ do not occur in the horizontal direction. The result follows from the fact that only $\beta\circ$ and the substitution rules can change the information content and the following claim:

$$
\begin{tikzcd}
\bullet \arrow[r, leftrightarrow] \arrow[d] & \bullet \arrow[d, dashed] \\
\bullet \arrow[r, dashed, leftrightarrow] & \bullet
\end{tikzcd}
\tag{12.4}
$$

To prove this claim we first introduce the extended β-rule:

$$\langle \cdots \langle \lambda x.M \mid D_1\rangle \cdots D_n\rangle\ N \xrightarrow[\beta^*]{} \langle \cdots \langle\langle M \mid x = N\rangle \mid D_1\rangle \cdots D_n\rangle\ .$$

We also consider a different standard reduction consisting of β^* and external substitution in an evaluation context. We denote this reduction by $\mapsto$

The claim 12.4 trivially follows from the following diagrams:

$$
\begin{tikzcd}
\bullet \arrow[r] \arrow[d] & \bullet \arrow[d, dashed] \\
\bullet \arrow[r, dashed, leftrightarrow] & \bullet
\end{tikzcd}
\quad
\begin{tikzcd}
\bullet \arrow[d] & \bullet \arrow[l] \arrow[d, dashed] \\
\bullet \arrow[r, dashed, leftrightarrow] & \bullet
\end{tikzcd}
\quad
\begin{tikzcd}
& \bullet \arrow[dl] \arrow[dr, dashed] & \\
\bullet \arrow[rr, dashed, leftrightarrow] & & \bullet
\end{tikzcd}
\quad
\begin{tikzcd}
& \bullet \arrow[dl] \arrow[dr, dashed] & \\
\bullet \arrow[rr, dashed, leftrightarrow] & & \bullet
\end{tikzcd}
\tag{12.5}
$$

We will first prove the first two diagrams. Apart from the situation where the top and left step commute by descendants we have the following critical pairs:

-

$$
\begin{tikzcd}
\langle \cdots \langle \lambda x.M \mid D_1\rangle \cdots D_n\rangle N \arrow[r] \arrow[d] & \langle\langle \cdots \langle \lambda x.M \mid D_1\rangle \cdots D_{n-1}\rangle N \mid D_n\rangle \arrow[d] \\
\langle \cdots \langle\langle M \mid x = N\rangle \mid D_1\rangle \cdots D_n\rangle \arrow[r, dashed, leftrightarrow] & \langle \cdots \langle\langle M \mid x = N\rangle \mid D_1\rangle \cdots D_n\rangle
\end{tikzcd}
$$

-

$$\begin{tikzcd}
\langle \cdots \langle \lambda x.M \mid D_1 \rangle \cdots D_n \rangle \langle N \mid D \rangle \arrow[r] \arrow[d] & \langle \langle \cdots \langle \lambda x.M \mid D_1 \rangle \cdots D_n \rangle N \mid D \rangle \arrow[d] \\
\langle \cdots \langle \langle M \mid x = \langle N \mid D \rangle \rangle \mid D_1 \rangle \cdots D_n \rangle \arrow[r, dashed, leftrightarrow] & \langle \langle \cdots \langle \langle M \mid x = N \rangle \mid D_1 \rangle \cdots D_n \rangle \mid D \rangle
\end{tikzcd}$$

-

$$\begin{tikzcd}
\langle \cdots \langle \lambda x.\langle M \mid D_1, D_2 \rangle \mid D_3 \rangle \cdots D_n \rangle N \arrow[r] \arrow[d] & \langle \cdots \langle \lambda x.\langle M \mid D_1 \rangle \mid D_2 \rangle \cdots D_n \rangle N \arrow[d] \\
\langle \cdots \langle \langle \langle M \mid D_1, D_2 \rangle \mid x = N \mid D_3 \rangle \cdots D_n \rangle \arrow[r, dashed, leftrightarrow] & \langle \cdots \langle \langle \langle M \mid D_1 \rangle \mid x = N \mid D_2 \rangle \cdots D_n \rangle
\end{tikzcd}$$

-

$$\begin{tikzcd}
\langle \cdots \langle \langle \langle \cdots \langle \lambda x.M \mid D_1 \rangle \cdots D_n \rangle \mid \rangle \mid D'_1 \rangle \cdots D'_{n'} \rangle N \arrow[r] \arrow[d] & \langle \cdots \langle \langle \cdots \langle \lambda x.M \mid D_1 \rangle \cdots D_n \rangle \mid D'_1 \rangle \cdots D'_{n'} \rangle N \arrow[d] \\
\langle \cdots \langle \langle \langle \cdots \langle \langle M \mid x = N \rangle \mid D_1 \rangle \cdots D_n \rangle \mid \rangle \mid D'_1 \rangle \cdots D'_{n'} \rangle \arrow[r, dashed, leftrightarrow] & \langle \cdots \langle \langle \cdots \langle \langle M \mid x = N \rangle \mid D_1 \rangle \cdots D_n \rangle \mid D'_1 \rangle \cdots D'_{n'} \rangle
\end{tikzcd}$$

-

$$\begin{tikzcd}
\langle \cdots \langle \lambda x.M \mid D_1 \rangle \cdots D_i \rangle \mid D_{i+1} \rangle \cdots D_n \rangle \ N \arrow[r] \arrow[d] & \langle \cdots \langle \lambda x.M \mid D_1 \rangle \cdots D_i, D_{i+1} \rangle \cdots D_n \rangle \ N \arrow[d] \\
\langle \cdots \langle \langle M \mid x = N \rangle \mid D_1 \rangle \cdots D_i \rangle \mid D_{i+1} \rangle \cdots D_n \rangle \arrow[r, dashed, leftrightarrow] & \langle \cdots \langle \langle M \mid x = N \rangle \mid D_1 \rangle \cdots D_i, D_{i+1} \rangle \cdots D_n \rangle
\end{tikzcd}$$

-

$$\begin{tikzcd}
\langle C[x] \mid x = \langle M \mid D \rangle, D' \rangle \arrow[r] \arrow[d] & \langle C[x] \mid x = M, D, D' \rangle \arrow[d] \\
\langle C[\langle M \mid D \rangle] \mid x = \langle M \mid D \rangle, D' \rangle \arrow[r, dashed, leftrightarrow] & \langle C[M] \mid x = M, D, D' \rangle
\end{tikzcd}$$

The bottom conversion follows from Proposition 11.5.

The third diagram of 12.5 is constructed as follows. Given a standard reduction we build a staircase: If the step is also a modified standard step we draw it in the downward direction, otherwise we draw it in the left direction. We then fill in the staircase with the first two diagrams of 12.5. For example if we have

$$\xrightarrow[\beta\circ]{} \xrightarrow[\text{lift}]{} \xrightarrow[\text{es}]{} \xrightarrow[\text{lift}]{} \xrightarrow[\beta\circ]{}$$

then we construct:

$$\begin{tikzcd}
 & & \cdot \arrow[d, "\beta\circ"'] \\
 & \cdot \arrow[d, "\text{es}"'] & \cdot \arrow[l, "\text{lift}"] \arrow[d, dashed] \\
\cdot \arrow[d, "\beta\circ"'] & \cdot \arrow[l, "\text{lift}"] \arrow[r, dashed, leftrightarrow] \arrow[d, dashed] & \cdot \arrow[d, dashed] \\
\cdot \arrow[r, dashed, leftrightarrow] & \cdot \arrow[r, dashed, leftrightarrow] & \cdot
\end{tikzcd}$$

The fourth diagram follows from the simple fact that if $M \mapsto N$ then $M \twoheadrightarrow N$.

We now prove the moreover part of the statement. The result that in the left diagram the number of $\beta\circ$/es-steps in the right standard reduction is not greater than the number of $\beta\circ$/es-steps in the left standard reduction follows immediately from the earlier diagrams. The fact that the number of $\beta\circ$/es-steps in the right standard reduction is smaller if a descendant of the top $\beta\circ$, es or is-redex is contracted in the standard reduction, can be proven by induction on the number of steps. $\square$

LEMMA 12.9. *Given* $\langle(\Lambda\circ, \underset{\lambda\circ}{\longrightarrow}, \preceq_{\text{name}}), \omega, (\omega(\Lambda\circ), \leq_\Omega)\rangle$.

(i) $\underset{\lambda\circ}{\longmapsto}$ *is complete up to* $\preceq_{\text{name}}$.

(ii) $\underset{\lambda\circ}{\longmapsto}$ *commutes with* $\rightarrow$ *up to* $\preceq_{\text{name}}$.

PROOF. From Lemmas 12.8 10.10. $\square$

THEOREM 12.10. *Given* $\langle(\Lambda\circ, \underset{\lambda\circ}{\longrightarrow}, \preceq_{\text{name}}), \omega, (\omega(\Lambda\circ), \leq_\Omega)\rangle$. $\underset{\lambda\circ}{\longrightarrow}$ *is confluent up to* $\preceq_{\text{name}}$.

PROOF. Follows from Lemmas 12.9 and 10.6. $\square$

REMARK 12.11. Note that the counterexample of confluence given in Example 9.8 does not constitute a counterexample of confluence up to information content, since both terms (*) and (**) contain the same amount of information, namely, Ω. The term M has a well defined infinite normal form, which is $\lambda z.\Omega$.

Confluence and standardization up to information content allow us to relate the equational reasoning to the operational behavior.

COROLLARY 12.12. *Let an answer A be a $\Lambda\circ$ term of the form*

$$A ::= \lambda x.M \mid \langle A \mid D\rangle \mid xM_1 \cdots M_n \ .$$

Given a term $M \in \Lambda\circ$ and an answer A. If $\lambda\circ \vdash M = A$ then there exists an answer A' such that $M \underset{\lambda\circ_{\text{eval}}}{\twoheadrightarrow} A'$ by doing leftmost outermost steps.

PROOF. By Propositions 10.4 and 9.7 we know that there exists M_1 such that:

$$M \underset{\lambda\circ}{\twoheadrightarrow} M_1 \text{ and } A \preceq_{\text{name}} M_1 \ .$$

It then follows that there exists an answer A_1 such that $M_1 \underset{\text{lift}}{\twoheadrightarrow} A_1$. From completeness up to information content of $\underset{\lambda\circ}{\longmapsto\!\!\!\!\twoheadrightarrow}$ (Lemma 12.9(i)) it follows that there is a term M_2 with $A_1 \preceq_{\text{name}} M_2$ such that $M \underset{\lambda\circ}{\longmapsto\!\!\!\!\twoheadrightarrow} M_2$. This M_2 is either of the form $\langle\langle\cdots\langle\lambda x.M \mid D_1\rangle \mid \cdots\rangle \mid D_n\rangle$ or it is of the form $App[x, M_1, \ldots, M_n]$. This last term is not necessarily an answer so we do all standard lift redexes to normal form to obtain an answer.

So we have that there exists a standard reduction from M to an answer. Every leftmost outermost step is a standard step, but not the other way round. So we will show by induction on the length of this standard reduction that there is a leftmost outermost reduction to an answer.

Given $M \underset{\lambda\circ}{\longmapsto\!\!\!\!\twoheadrightarrow} A$. If M is already an answer then we are done, otherwise M must be either of the form $App[R, M_1, \ldots, M_n]$, where R is the redex that is contracted in the first step, or of the form $App[x, M_1, \ldots, M_n]$. In the first case, we do leftmost outermost redexes until we have done R (The leftmost outermost redex is either R or a lift redex and since lift reduction terminates eventually we must do R.) If we commute this leftmost outermost reduction (which is still a standard reduction) with the standard reduction we get a shorter standard reduction that ends in an answer and by induction hypothesis we are done. (Remember from the proof of Lemma 12.6 that standard steps commute in one step.) In the second case, we can just leftmost outermost reduce the term to lift normal form to obtain an answer. $\square$

13. Semantics of the cyclic lambda calculus

We have shown that $\xrightarrow[\lambda\circ]{}$ is confluent up to information content. Hence, by Lemma 10.18 the infinite normal form of any cyclic term M, written as $\mathrm{Inf}_{\lambda\circ}(M)$, is well defined and unique. Next, we want to show that the infinite normal form defines a congruence, *i.e.,* $\mathrm{Inf}_{\lambda\circ}(M) = \mathrm{Inf}_{\lambda\circ}(N) \implies \mathrm{Inf}_{\lambda\circ}(C[M]) = \mathrm{Inf}_{\lambda\circ}(C[N])$, for all contexts C. To that end, let us first show that the infinite normal forms of a cyclic term computed with respect to $\xrightarrow[\lambda\circ]{}$, $\underset{\lambda\circ}{\longmapsto}$ (written as $\mathrm{Inf}_{\mathrm{std}}$) and $\xrightarrow[\mathrm{eval}]{}$ (written as $\mathrm{Inf}_{\mathrm{eval}}$) are the same. By the infinite normal form of a cyclic term computed with respect to $\xrightarrow[R]{}$, we mean the infinite normal form computed with respect to $\langle(\Lambda\circ, \xrightarrow[R]{}, \preceq_{\mathrm{name}}), \omega, (\omega(\Lambda\circ), \leq_\Omega)\rangle$.

Proposition 13.1. *Given a term* $M \in \Lambda\circ$. $\mathrm{Inf}_{\mathrm{std}}(M) = \mathrm{Inf}_{\mathrm{eval}}(M) = \mathrm{Inf}_{\lambda\circ}(M)$.

Proof. Because $\mapsto\!\!\!\twoheadrightarrow \subseteq \underset{\mathrm{eval}}{\twoheadrightarrow} \subseteq \underset{\lambda\circ}{\twoheadrightarrow}$ we have:

$$\mathrm{Inf}_{\mathrm{std}}(M) \subseteq \mathrm{Inf}_{\mathrm{eval}}(M) \subseteq \mathrm{Inf}_{\lambda\circ}(M) \ .$$

Because $\underset{\lambda\circ}{\longmapsto}$ is complete for $\xrightarrow[\lambda\circ]{}$ up to information content we have that

$$\mathrm{Inf}_{\lambda\circ}(M) \subseteq \mathrm{Inf}_{\mathrm{std}}(M) \ .$$

□

This result allows us to prove that the infinite normal form computed with respect to full reduction (*i.e.,* $\xrightarrow[\lambda\circ]{}$) is a congruence, by showing congruence of the infinite normal form computed with respect to the evaluation calculus. Congruence with respect to the evaluation calculus is easier to prove, since that calculus is confluent by using the complete development method.

We follow the proof strategy described in [Lév78]. We first prove monotonicity:

Lemma 13.2. *Given* $M, N \in \Lambda\circ$. *If* $M \leq_\Omega N$ *then* $\mathrm{Inf}_{\mathrm{eval}}(M) \subseteq \mathrm{Inf}_{\mathrm{eval}}(N)$.

Proof. From Proposition 12.2(i), $\leq_\Omega$ is monotonic with respect to $\preceq_{\mathrm{name}}$. We claim that $(\Lambda\circ, \leq_\Omega)$ and $(\Lambda\circ, \xrightarrow[\mathrm{eval}]{})$ commute so by Lemma 10.19 we are done. The claimed commutativity follows from the following diagrams:

$$\begin{tikzcd} \cdot \arrow[d, "\beta\circ"'] & \cdot \arrow[l, "\Omega"'] \arrow[d, dashed, "\beta\circ"] \\ \cdot & \cdot \arrow[l, dashed, "\Omega"] \end{tikzcd} \qquad \begin{tikzcd} \cdot \arrow[d, "\mathrm{lift}"'] & \cdot \arrow[l, "\Omega"'] \arrow[d, dashed, "\mathrm{lift}"] \\ \cdot & \cdot \arrow[l, dashed, "\Omega"] \end{tikzcd} \qquad \begin{tikzcd} \cdot \arrow[d, "\mathrm{es}"'] & \cdot \arrow[l, "\Omega"'] \arrow[d, dashed, "\mathrm{es}"] \\ \cdot & \cdot \arrow[l, dashed, "\Omega"] \end{tikzcd} \qquad \begin{tikzcd} \cdot \arrow[d, "\mathrm{es}"'] & & \cdot \arrow[ll, "\Omega"'] \arrow[d, dashed, "\mathrm{es}"] \\ \cdot & \cdot \arrow[l, dashed, "\Omega"] & \cdot \arrow[l, dashed, "\Omega"] \end{tikzcd}$$

□

Remark 13.3. The use of the evaluation calculus in the previous proof was necessary, because $\xrightarrow[\lambda\circ]{}$ does not commute with $\leq_\Omega$. See for example the following two critical pairs:

$$\begin{array}{ccc} \langle \Omega \mid x = z\ x\rangle & \xrightarrow{\mathrm{gc}} & \langle \Omega \mid\rangle \\ \leq_\Omega & & \\ \langle x \mid x = z\ x\rangle & & ? \end{array} \qquad \begin{array}{ccc} \lambda x.\langle y \mid y = \Omega\rangle & \xrightarrow{\mathrm{lift}} & \langle \lambda x.\langle y \mid\rangle \mid y = \Omega\rangle \\ \leq_\Omega & & \\ \lambda x.\langle y \mid y = x\ x\rangle & & ? \end{array}$$

Proposition 13.4. *Given a term* $M \in \Lambda\circ$ *and a context* C.

(i) *If* $C[M] \underset{\text{eval}}{\twoheadrightarrow} N$ *then there exists an* M_1 *such that* $M \underset{\text{eval}}{\twoheadrightarrow} M_1$, $C[M_1] \underset{\text{eval}}{\twoheadrightarrow} P$ *without reducing any redex inside* M_1 *(written as* $\underset{M_1}{\not\twoheadrightarrow}$*) and* $N \underset{\text{eval}}{\twoheadrightarrow} P$.

(ii) *If* $C[M] \overset{\text{eval}}{\underset{M}{\not\twoheadrightarrow}} N$ *then* $C[\omega(M)] \underset{\text{eval}}{\twoheadrightarrow} N_1$ *with* $N \simeq_{\text{name}} N_1$.

(iii) $\mathrm{Inf}_{\text{eval}}(C[\omega(M)]) \subseteq \mathrm{Inf}_{\text{eval}}(C[M])$.

PROOF.

(i) Follows from the confluence of λ_{eval} by complete developments.

(ii) The computation of information content can be captured by three rules:

$$\begin{array}{rcrcl}
\beta\omega & : & (\lambda x.M)\ N & \rightarrow & \Omega \\
\mathrm{s}\omega & : & \langle M \mid x_1 = N_1, \cdots, x_n = N_n \rangle & \rightarrow & M[x_1 := \Omega, \cdots, x_n := \Omega] \\
@\omega & : & \Omega\ M & \rightarrow & \Omega
\end{array}$$

The result follows from the following three diagrams:

$$\begin{tikzcd}
C[M] \arrow[r, two heads, "\text{eval}", "\not M"'] \arrow[d, "\|\beta\omega"'] & N \arrow[d, dashed, "\|\beta\omega"] \\
C[M_1] \arrow[r, dashed, two heads, "\text{eval}", "\not M_1"'] & N'
\end{tikzcd}
\qquad
\begin{tikzcd}
C[M] \arrow[r, two heads, "\text{eval}", "\not M"'] \arrow[d, "\|\mathrm{s}\omega"'] & N \arrow[d, dashed, "\|\mathrm{s}\omega"] \\
C[M_1] \arrow[r, dashed, two heads, "\text{eval}", "\not M_1"'] & N'
\end{tikzcd}
\qquad
\begin{tikzcd}
C[M] \arrow[r, two heads, "\text{eval}", "\not M"'] \arrow[d, "\|@\omega"'] & N \arrow[d, dashed, "\|@\omega"] \\
C[M_1] \arrow[r, dashed, two heads, "\text{eval}", "\not M_1"'] & N'
\end{tikzcd}$$

where $\|\ r$ means a complete development of redexes of type r.

(iii) If $M \underset{\omega}{\rightarrow} N$ then either $N \leq_\Omega M$ or the ω-step is a garbage collection step. The result then follows from monotonicity of $\mathrm{Inf}_{\text{eval}}$ with respect to $\leq_\Omega$ (Lemma 13.2) and uniqueness of the infinite normal form.

□

REMARK 13.5. In general ω-reduction does not commute with reduction:

$$\begin{tikzcd}
(\lambda x.y)z \arrow[r] \arrow[d] & \langle y|x = z\rangle \\
\Omega & ?
\end{tikzcd}$$

THEOREM 13.6. *Given* $M, N \in \Lambda\circ$ *and a context* C. *Then*

$$\mathrm{Inf}_{\lambda\circ}(M) = \mathrm{Inf}_{\lambda\circ}(N) \Longrightarrow \mathrm{Inf}_{\lambda\circ}(C[M]) = \mathrm{Inf}_{\lambda\circ}(C[N])\ .$$

PROOF. Same as in [Lév78]. □

14. The cyclic lambda calculus and the traditional lambda calculus

As it was done in [Ari96] for the first-order case, we can use the model to relate $\lambda\circ$ to the traditional lambda calculus. We show that cycles can be explained in terms of their expansions, which are finite lambda calculus terms. To define expansion, we introduce the notation $M \xrightarrow[\text{GK(es)}]{}^n N$, which denotes n steps of the Gross-Knuth strategy applied to the external substitution redexes occurring in M (*i.e.*, all external substitution redexes are performed). If M does not contain any external substitution redexes we still write $M \xrightarrow[\text{GK(es)}]{} N$. Recalling the definition of the strip function (Definition 11.1) we define:

DEFINITION 14.1. Given $M \in \Lambda\circ$.

(i) The n^{th} unfolding of M, written as M_n, is given by $M \xrightarrow[\text{GK(es)}]{}^n M_n$.

(ii) The n^{th} expansion of M, written as M^n, is the term $\text{strip}(M_n)$.

We will build towards the main result with two lemmas. The first lemma relates suitable $\beta\circ$/es-reduction sequences on graphs to β-reduction sequences on lambda calculus terms. The second lemma relates each piece of information derivable from a cyclic term to information derivable from an expansion of that term.

LEMMA 14.2. *Given $M, M_1, M_2 \in \Lambda\circ$. If $M \twoheadrightarrow_{\beta\circ} M_1 \twoheadrightarrow_{\text{es}} M_2$ such that $M_1 \twoheadrightarrow_{\text{es}} M_2$ is a complete development of all the external substitution redexes created by the $\beta\circ$-steps then* $\text{strip}(M) \twoheadrightarrow_{\beta} \text{strip}(M_2)$.

PROOF. We distinguish cases by the number of $\beta\circ$-steps n:

$n = 1$. Given

$$P \equiv C[(\lambda x.M)N] \underset{\beta\circ}{\rightarrow} C[\langle M \mid x = N\rangle \underset{\text{es}}{\twoheadrightarrow} C[\langle M[x := N] \mid x = N\rangle \equiv Q \ .$$

We will prove by structural induction on $C[\Box]$ that either $\text{strip}(P) \underset{\beta}{\rightarrow} \text{strip}(Q)$ or $\text{strip}(P) \equiv \text{strip}(Q)$.

- $C \equiv \Box$: Working from both sides we get

$$\text{strip}((\lambda x.M)N) \equiv (\lambda x.\text{strip}(M))\ \text{strip}(N)$$

and

$$\begin{aligned} \text{strip}(\langle M[x := N] \mid x = N\rangle) &\equiv \text{strip}(M[x := N]) \\ &\equiv \text{strip}(M[x := \text{strip}(N)]) \\ &\equiv \text{strip}(M)[x := \text{strip}(N)] \ . \end{aligned}$$

It then follows that

$$\text{strip}((\lambda x.M)N) \underset{\beta}{\rightarrow} \text{strip}(\langle M[x := N] \mid x = N\rangle) \ .$$

- $C \equiv \lambda y.C'$: Follows from the fact that $\text{strip}(\lambda z.M') \equiv \lambda z.\text{strip}(M')$ and the induction hypothesis.
- $C \equiv C'\ M$ and $C \equiv M\ C'$: Follow from the fact that $\text{strip}(M'\ N') \equiv \text{strip}(M')\ \text{strip}(N')$ and the induction hypothesis.
- $C \equiv \langle C' \mid x_1 = N_1, \cdots, x_n = N_n\rangle$: Follows from the fact that

$$\begin{aligned} \text{strip}(\langle M' \mid x_1 = N_1, \cdots, x_n = N_n\rangle) &\equiv \text{strip}(M'[x_1 := \Omega, \cdots, x_n := \Omega]) \\ &\equiv \text{strip}(M')[x_1 := \Omega, \cdots, x_n := \Omega] \end{aligned}$$

and the induction hypothesis.
- $C \equiv \langle M' \mid y = C', x_1 = N_1, \cdots, x_n = N_n\rangle$: We have the following simple situation:

$$\text{strip}(P) \equiv \text{strip}(Q) \equiv \text{strip}(M')[x_1 := \Omega, \cdots, x_n := \Omega] \ .$$

$n > 1$. We will reorder the $\underset{\beta\circ}{\twoheadrightarrow}\underset{\text{es}}{\twoheadrightarrow}$ subsequence into an equivalent $(\underset{\beta\circ}{\rightarrow}\ \underset{\text{es}}{\twoheadrightarrow})^+$ sequence. We use induction on the number of $\beta\circ$-steps. If we are given a single $\beta\circ$-step we are done. If we are given more than one, there must be an outermost $\beta\circ$-redex. We reorder the $\beta\circ$-sequence, preserving it's length in such a way that this step is at the end. Let $M \underset{\beta\circ}{\twoheadrightarrow} M' \underset{\beta\circ}{\rightarrow} M_1$, we then construct the es-sequence contracting all created es-redexes in M_1 and we commute this sequence with the last $\beta\circ$-step and the original es-sequence. What we obtain is:

$$
\begin{tikzcd}
{} \arrow[r, two heads, "\beta\circ"] & {} \arrow[r, "\beta\circ"] \arrow[d, two heads, "\mathrm{es}"'] & {} \arrow[r, two heads, "\mathrm{es}"] \arrow[d, two heads, "\mathrm{es}"'] & {} \arrow[d, phantom, "\equiv"] \\
 & {} \arrow[r, dashed, "\beta\circ"'] & {} \arrow[r, dashed, two heads, "\mathrm{es}"'] & {} .
\end{tikzcd}
$$

The single $\beta\circ$-step at the bottom follows the fact that all the es-steps on the left occur disjoint from the $\beta\circ$-redex or inside it.

□

LEMMA 14.3. *Given $M, N \in \Lambda\circ$. If $M \underset{\lambda\circ}{\twoheadrightarrow} N$ then there exists an n and $P \in \Lambda$ such that $M^n \underset{\beta}{\twoheadrightarrow} P$ and $N \preceq_{\mathrm{name}} P$.*

PROOF. Without loss of generality $M \underset{\lambda\circ}{\mapsto\!\!\!\twoheadrightarrow} N$. Let us first find the n.

We divide the es-redexes contracted in a reduction sequence into two categories: those created by a $\beta\circ$-step and those created by es-steps or already present in the first term of the sequence. We remind the reader that $\beta\circ$ created es-redexes cannot create new es-redexes. Let $N \equiv N_0$. Let the construction have proceeded up to $M_i \underset{\lambda\circ}{\mapsto\!\!\!\twoheadrightarrow} N_i$. If the sequence from M_i to N_i contracts only $\beta\circ$-created substitution redexes then we take $n \equiv i$, otherwise by Lemma 12.8 we can find an N_{i+1} such that $M_{i+1} \mapsto\!\!\!\twoheadrightarrow N_{i+1}$ and $N_i \preceq_{\mathrm{name}} N_{i+1}$. Moreover, the fact that the sequence $M_i \underset{\lambda\circ}{\mapsto\!\!\!\twoheadrightarrow} N_i$ does at least one es-redex present in M_i implies that there are less $\beta\circ$/es-steps in $M_{i+1} \mapsto\!\!\!\twoheadrightarrow N_{i+1}$ than in $M_i \underset{\lambda\circ}{\mapsto\!\!\!\twoheadrightarrow} N_i$. Therefore the construction terminates and we have found an n such that $M_n \mapsto\!\!\!\twoheadrightarrow N_n$ by doing only es-steps created by $\beta\circ$-steps.

Let us now find term P. We show that starting with the reduction $M_n \twoheadrightarrow N_n$ we can find a reduction of the form

$$M_n \underset{\mathrm{lift}}{\twoheadrightarrow} \underset{\beta\circ}{\twoheadrightarrow} \underset{\mathrm{es}}{\twoheadrightarrow} \underset{\mathrm{lift}}{\twoheadrightarrow} \underset{\beta\circ}{\twoheadrightarrow} \underset{\mathrm{es}}{\twoheadrightarrow} \underset{\mathrm{lift}}{\twoheadrightarrow} \ldots Q \ ,$$

where the external substitution multi-step sequences following the multi-step $\beta\circ$-steps do precisely the created redexes and where $N_n \preceq_{\mathrm{name}} Q$. To prove this we group the reduction sequence from M_n to N_n into a sequence of $\beta\circ$, lift and es multi-step reductions. We assume these multi-step reductions consist of at least one redex. We proceed by induction on the number of these multi-steps. The head multi-step can be a lift multi-step or a $\beta\circ$ multi-step. In case of a lift reduction we apply the induction hypothesis to the tail. In case of $\beta\circ$-reduction we construct a new sequence by reducing all newly created es-redexes to normal form. This reduction is finite. Then we commute this es-reduction with the tail. Observe that the head multi-step of the new tail cannot be a non-empty es-reduction. By induction we are done. In a diagram:

$$
\begin{tikzcd}
{} \arrow[r, two heads, "\beta\circ"] & {} \arrow[r, two heads] \arrow[d, two heads, "\mathrm{es}"'] & {} \arrow[r, two heads] \arrow[d, two heads, "\mathrm{es}"'] & {} \arrow[d, two heads, "\mathrm{es}"'] & \cdots & {} \arrow[d, two heads, "\mathrm{es}"'] \\
 & {} \arrow[r, two heads] & {} \arrow[r, two heads] & {} & \cdots & {}
\end{tikzcd}
$$

The result then follows from Lemma 14.2 and the fact that if $M \underset{\mathrm{lift}}{\rightarrow} N$ then $\mathrm{strip}(M) = \mathrm{strip}(N)$.
□

Notation: $\mathrm{Inf}_\lambda(M)$ denotes the infinite normal form of a lambda term M computed with respect to $\langle(\Lambda, \underset{\beta}{\rightarrow}, \preceq_{\mathrm{name}}), \omega, (\omega(\Lambda), \leq_\Omega)\rangle$ (as described in [Lév78]).

THEOREM 14.4. *Given* $M \in \Lambda\circ$. *Then* $\mathrm{Inf}(M) = \bigcup\{\mathrm{Inf}_\lambda(M^i) \mid i \geq 0\}$.

PROOF.

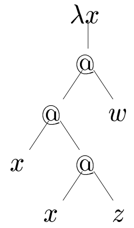
"$\supseteq$" Let i be given. Because $\twoheadrightarrow_\beta \subseteq \twoheadrightarrow_{\lambda\circ}$ we have that $\mathrm{Inf}_\lambda(M^i) \subseteq \mathrm{Inf}(M^i)$. The result then follows from monotonicity with respect to the Ω-ordering (Lemma 13.2) and uniqueness of the infinite normal form.

"$\subseteq$" Follows from Lemma 14.3.

$\square$

15. The cyclic lambda calculus and the infinitary lambda calculi

In this section we study the relation of $\lambda\circ$ with the infinitary lambda calculi of Kennaway *et al.* [KKSdV95]. To disambiguate a reduction in $\lambda\circ$ from a reduction in the infinitary lambda calculus we will sometimes subscript the latter with λ^∞. Reductions in λ^∞ can be of transfinite length. Reductions consisting of either finitely many or at most ω steps are written as $\twoheadrightarrow^{\leq\omega}$. Kennaway *et al.* [KKSdV95] define infinite reductions as ***strongly convergent*** reductions, *i.e.,* reductions in which the depth of the contracted redex tends to infinity. Based on different depth measures, they developed eight different infinitary calculi. More precisely, given the three contexts: $\lambda x.[\ ]$, $[\ ]t$, and $t[\ ]$, the different depth measures result from assigning different depths to the hole. A depth measure of $0**$, $*0*$, and $**0$, where the symbol $*$ stands for either 0 or 1, signifies that the depth measure does not increase by going through a lambda-node, the left branch, and right branch of an application node, respectively. For example, in the term $\lambda x.x(xz)w$ which is displayed below:



the depth of z according to 001, 101 and 111-calculi is $0+0+1+1 = 2$, $1+0+1+1 = 3$ and $1+1+1+1 = 4$, respectively. Among the eight calculi, there are three, namely the 001, 101 and 111-calculi, that are confluent if the calculi admit next to the β-rule the following $\perp$-rule:

$$s \xrightarrow[\perp]{} \Omega \text{ if } s \text{ is 0-active} \quad ,$$

where 0-active means that s always has a redex at depth 0. The meaning of a term is then its unique infinitary normal form.

We define the unwinding of cyclic terms to infinitary terms as follows:

DEFINITION 15.1. Given a term $M \in \Lambda\circ$, the (possibly infinite) tree unwinding of M, written as M^∞, is defined as

$$M^\infty = \lim_{n\to\infty} M^n \quad .$$

Λ^∞ denotes the set $\{M^\infty \mid M \in \Lambda^\infty\}$.

Note that both the 001 and 101-calculi do not contain all terms that could arise from unwinding cyclic terms, *e.g.,* $\langle x \mid x = \lambda z.x\rangle^\infty$ is illegal in 001 (but legal in 101) and $\langle x \mid x = xz\rangle^\infty$ is illegal in 001 and 101. The reason is that these terms cannot be reached with strongly convergent reductions starting from finite lambda calculus terms. To relate $\lambda\circ$ to these calculi, we could define a different notion of unwinding or an equivalence relation on $\lambda\circ$ terms in such a way that the canonical representative of each class corresponds to a 001 or a 101-term. However, this would complicate our system with further rules. The only calculus that relates to $\lambda\circ$ is the 111-calculus, with respect to which we prove soundness and a weak notion of completeness, namely, if $M^\infty \twoheadrightarrow_{\lambda^\infty} s$ then each finite prefix of s can be obtained in $\lambda\circ$ by reducing M. For the completeness result to hold the evaluation calculus (see Table 5) suffices.

We have now introduced three notions of unwinding:

- The unwinding of a graph to an infinite tree.
- The $\mathcal{R}\circ^{\rightarrow}$ infinite normal form of a cyclic term.
- The unwinding of a cyclic term to an infinitary term.

We will now show that these notions are the same, in the sense that they yield isomorphic objects. The relationship between the first and last notion established in the following proposition is useful in establishing the soundness of $\lambda\circ$ with respect to λ^∞.

Proposition 15.2. *Given a term M in $\Lambda\circ$. Then M^∞ is isomorphic to $\rho(M)^u$.*

Proof. The set of infinite trees is a complete metric space when we define $d(g,h) = 2^{-|p|}$, where p is the shortest path exposing a difference between g and h. We have that M^∞ is the limit of $M^1, M^2, \cdots$. We also have that $\rho(M)^u = \rho(M_1)^u = \rho(M_2)^u \cdots$. From the fact that $d(\rho(M_i)^u, M^i) \xrightarrow[i\to\infty]{} 0$ it then follows that $\rho(M)^u \equiv M^\infty$. $\square$

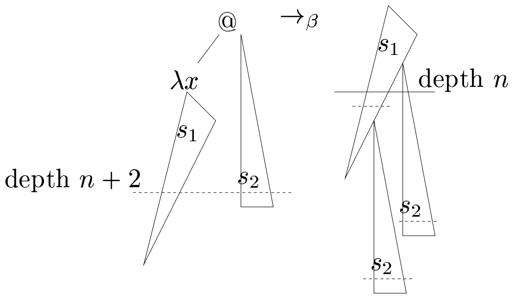
Proposition 15.3. *Given $M \in \Lambda\circ$. We have $\bigsqcup(\mathrm{Inf}_{\mathcal{R}\circ^{\rightarrow}}(M)) = M^\infty$.*

Proof. From the fact that for each i we have that $M^i \in \mathrm{Inf}_{\mathcal{R}\circ^{\rightarrow}}(M)$ it follows that $M^\infty \leq_\Omega \bigsqcup(\mathrm{Inf}_{\mathcal{R}\circ^{\rightarrow}}(M))$. From the fact that if $M \mapsto\!\!\!\twoheadrightarrow_{\mathcal{R}\circ} N$ then there exists an i such that $\mathrm{strip}(N) \leq_\Omega M^i$ it follows that $\bigsqcup(\mathrm{Inf}_{\mathcal{R}\circ^{\rightarrow}}(M)) \leq_\Omega M^\infty$ $\square$

We continue by relating one β-step in the infinitary calculus to a reduction sequence in the cyclic calculus. Let us first show that rewriting is continuous. Notation: If t is a tree, t^n denotes the finite tree truncated at depth n. $s \xrightarrow[\beta]{r} t$ denotes a reduction in the infinitary lambda calculus of redex located at path r.

Lemma 15.4. *Given an ascending sequence $(s_i)_{i=1}^\infty \subseteq \Lambda^\infty$ with limit s. If $s \xrightarrow[\beta]{r} t$ then there exists an n such that $t = \lim\ \{t_i \mid s_i \xrightarrow[\beta]{r} t_i\}_{i=n}^\infty$.*

Proof. Let p be the depth at which the redex contracted in the step $s \xrightarrow[\beta]{r} t$ occurs, *i.e.*, the length of path r. By definition of limit there exists n from which onward the sequence s_i does no longer change at a depth less than $p+2$. For this n it is obvious that at least the application and the lambda of the redex in s are exposed and therefore can be contracted. The existence of the limit and equality of the limit to t is best shown using *proof by picture*. Assume that $s \equiv (\lambda x.s_1)s_2$. In the following picture we show what happens when you take the prefix of depth $p+2$ in the left-hand side and transport it to the right-hand side and compare it to the prefix of depth p.



It follows from the picture that when $s' \leq_\Omega s'' \leq_\Omega s$ and $s' \xrightarrow[\lambda^\infty]{r} t'$ and $s'' \xrightarrow[\lambda^\infty]{r} t''$ then $t' \leq_\Omega t'' \leq_\Omega t$. Moreover, we have:

$$\forall n \exists m_0 \forall m \geq m_0 : t^n \leq_\Omega t_m \ .$$

From these two observations and the fact that $t_i \leq_\Omega t$ the final result follows. □

LEMMA 15.5. *Given a term $M \in \Lambda\circ$. If $M^\infty \xrightarrow[\beta]{} t$ then $M \xrightarrow[\lambda_{\mathrm{eval}}]{} \!\!\!\!\to N$ such that $N^\infty \equiv t$.*

PROOF. Let r denote the redex contracted in the step from M to t. Due to continuity of the infinitary lambda reduction (Lemma 15.4) there exists an n such that :

$$t = \lim\{t_i \mid M^i \xrightarrow[\beta]{r} t_i\}_{i=n}^{\infty} \ .$$

By an analysis of stripping, the following diagrams holds:

$$\begin{tikzcd}
M \arrow[d, "\mathrm{GK(es)}"'] & & & & \\
M_n \arrow[r, dashed, "\mathrm{lift}"] \arrow[dr, "\mathrm{strip}"'] & {} \arrow[r, dashed, "\beta\circ", "\lambda\circ"'] & {} \arrow[r, dashed, "\mathrm{es}"] & N \arrow[dl, dashed, "\mathrm{strip}"] \\
& M^n \arrow[r, "\beta", "\lambda^\infty"'] & t_n &
\end{tikzcd}$$

Now we need to show that for the N defined by the diagram above

$$\lim_{m\to\infty} N^m = \lim\{t_i \mid M^i \xrightarrow[\beta]{r} t_i\}_{i=n}^{\infty} \ .$$

By induction on i we show the following:

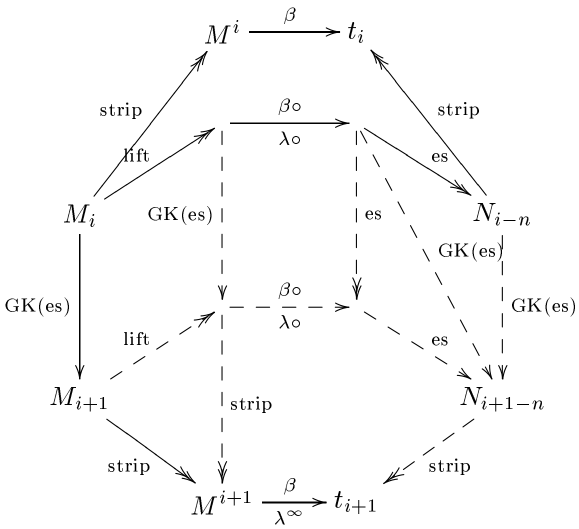
$$\forall i \geq n, t_i \equiv N^{i-n} \ .$$

$i = n$. Trivial.

$i \geq n$. Let us assume it for i and we prove it for $i+1$. That is, we want to prove

$$t_{i+1} \equiv N^{i+1-n} \ .$$

We have the following diagram:



The solid arrows are known. To prove the existence of the dashed arrows let us remark that external substitution and lifting commute without destroying or creating external substitution redexes. β-reduction commutes also without destroying redexes, but it may create some new redexes. Doing the descendants of the redexes contracted from M_i to M_{i+1} plus doing the newly created redexes is a GK step. Since doing the newly created redexes in M_i does not create new ones, the resulting steps from M_i to M_{i+1} form a GK step. The M_{i+1} created this way strips to t_{i+1} because a lift step does not change the stripped term and because doing a β followed by the newly created substitution redexes followed by stripping is the same as stripping first and then doing a β.

□

Since reductions in λ^∞ consist also of $\perp$-reductions, before proving the main result we need a further lemma.

LEMMA 15.6. *Given $s, t \in \Lambda^\infty$. If $s \twoheadrightarrow_{\beta\perp} t$ then there exists a $t' \in \Lambda^\infty$ such that $s \twoheadrightarrow_{\beta} t'$ and $t \leq_\Omega t'$.*

PROOF. Let us first observe that if we have a β-reduction step from p to q and we replace some Ω's in p by some term to obtain p' we can still do a single β-step to obtain q' and this q' can be obtained from q by replacing some Ω's by some term. Pictorially:

$$\begin{tikzcd} {} \arrow[r, "\beta"] \arrow[d, phantom, "\leq_\Omega"] & {} \arrow[d, phantom, "\leq_\Omega"] \\ {} \arrow[r, dashed, "\beta"'] & {} \end{tikzcd}$$

Note that it is possible that we may have to replace infinitely many Ω's. With the above observation in hand we will proceed with a *proof by example*. Suppose we have a sequence of β and $\perp$-steps:

$$\xrightarrow[\beta]{}\xrightarrow[\perp]{}\xrightarrow[\beta]{}\xrightarrow[\beta]{}\xrightarrow[\perp]{}\xrightarrow[\beta]{} \quad .$$

We can put this sequence into two dimensions by doing β from left to right and $\perp$ from bottom to top. The next step is to tile the diagram with the elementary diagram the observation provides:

$$\begin{tikzcd}
& & & \cdot \arrow[r, "\beta"] & \cdot \\
& \cdot \arrow[r, "\beta"] & \cdot \arrow[r, "\beta"] & \cdot \arrow[u, "\perp"] \arrow[r, dashed, "\beta"'] \arrow[ur, phantom, "\leq_\Omega"] & \cdot \\
\cdot \arrow[r, "\beta"] & \cdot \arrow[u, "\perp"] \arrow[r, dashed, "\beta"'] \arrow[ur, phantom, "\leq_\Omega"] & \cdot \arrow[r, dashed, "\beta"'] \arrow[ur, phantom, "\leq_\Omega"] & \cdot \arrow[r, dashed, "\beta"'] \arrow[ur, phantom, "\leq_\Omega"] & \cdot
\end{tikzcd}$$

□

THEOREM 15.7. *Given a term $M \in \Lambda\circ$. If $M^\infty \underset{\lambda\infty}{\twoheadrightarrow} t$ then for all finite $s \leq_\Omega t$, $M \underset{\lambda\circ}{\twoheadrightarrow} N, s \leq_\Omega N^\infty$.*

PROOF. From the compression lemma of the infinitary lambda calculus we get:

$$M^\infty \underset{\lambda\infty}{\twoheadrightarrow}^{\leq\omega} t \ \ .$$

Let's now study the reduction sequence:

$$M^\infty \equiv t_0 \underset{\lambda\infty}{\longrightarrow} t_1 \underset{\lambda\infty}{\longrightarrow} t_2 \underset{\lambda\infty}{\longrightarrow} \ldots t \ \ .$$

By definition of infinite reduction sequence for all finite $s \leq_\Omega t$, there exists an i such that for all $j \geq i$, $t_j \geq s$. The result then follows from the previous two lemmas. □

We can now prove that $\lambda\circ$ is sound with respect to the 111-infinitary lambda calculus.

THEOREM 15.8. *Given $M, N \in \Lambda\circ$. Then*

$$M \underset{\lambda\circ}{\twoheadrightarrow} N \Longrightarrow M^\infty \underset{\beta\perp}{\twoheadrightarrow}^{\leq\omega} N^\infty \ \ .$$

PROOF. Once we have proven the result for a single step the result for multiple steps follows from the compression lemma of the infinitary lambda calculus.

If $M \to N$ by any step other than $\beta\circ$ then by Theorem 6.12 M and N represent graphs that have the same unwinding. Therefore by Proposition 15.2 $M^\infty \equiv N^\infty$. If $M \underset{\beta\circ}{\longrightarrow} N$ then we will construct a t such that $M^\infty \underset{\beta\perp}{\twoheadrightarrow}^{\leq\omega} t$ and $t \equiv N^\infty$.

To define t we denote by $\underline{M}$ the term M with the lambda of the redex contracted in the step from M to N underlined. If there are no underlined redexes in $\underline{M}^\infty$ then we define $t \equiv \underline{M}^\infty$, this may happen if the $\beta\circ$-redex is garbage collected during the unwinding. If $\underline{M}^\infty$ contains an infinite number of underlined redexes it means that the $\beta\circ$-redex in M lies on a cycle. That is, it must have a name associated to it, say x, *i.e.,* M contains an equation of the form $x = (\underline{\lambda} y.P)Q$. If $P \equiv x$ or $P \equiv y$ and $Q \equiv x$, then it is not the case that $M^\infty \underset{\beta}{\twoheadrightarrow}^{\leq\omega} N^\infty$ because the depth of the contracted redex at each step remains 0. Consider the following two examples:

$$\langle x \mid x = (\underline{\lambda} y.y)x\rangle \underset{\beta\circ}{\longrightarrow} \langle x \mid x = \langle y \mid y = x\rangle\rangle$$

and

$$\langle x \mid x = (\underline{\lambda} y.x)0\rangle \underset{\beta\circ}{\longrightarrow} \langle x \mid x = \langle x \mid y = 0\rangle\rangle \ \ .$$

However, the following holds:

$$C[(\lambda y.P)Q]^\infty \underset{\perp}{\twoheadrightarrow}^{\leq\omega} C[\Omega]^\infty \equiv C[\langle P \mid y = Q\rangle]^\infty \ \ .$$

In the other cases we define t by doing a complete development of all the underlined β-redexes in $\underline{M}^\infty$, which is now well defined.

The next step is to show that $t \equiv N^\infty$.

$N^\infty \leq_\Omega t$.

We show that $\forall N'$, with $N \xrightarrow[\text{es}]{}\!\!\!\!\rightarrow N'$, $\mathrm{strip}(N') \leq_\Omega t$. We first construct M' as follows: Let M_0 be M and let N_0 be N'. If all of the external substitution redexes in the sequence $M_i \twoheadrightarrow N_i$ are created by a $\beta\circ$-step in that sequence then there is no M_{i+1}. Otherwise we take all es-redexes in M_i that have a descendant contracted in the sequence to N_i. The result of doing a complete development of these redexes we denote by M_{i+1} and we denote the common reduct by descendants by N_{i+1}. The sequence sequence of M_i's constructed in this way must be finite and we denote the last element M_n by M'. The sequence from M_n to N_n starts with several $\beta\circ$-steps. By N'' we denote the unique result of contracting in N_n all the es-redexes that these $\beta\circ$-steps created and that are still present in N_n.

From Lemma 14.2 it follows that $\mathrm{strip}(M') \twoheadrightarrow_{\beta} \mathrm{strip}(N'')$. If we follow the underlining we must conclude that all of the redexes contracted in the reduction from $\mathrm{strip}(M') \twoheadrightarrow_{\beta} \mathrm{strip}(N'')$ are underlined. This means that $M^\infty \twoheadrightarrow_{\beta} t'$ such that $\mathrm{strip}(N'') \leq_\Omega t'$ and $t' \twoheadrightarrow_{\beta} t$. From the fact that $\mathrm{strip}(N'')$ does not contain any underlining and the fact that if the argument of an underlined $\beta\circ$-redex shows up in a stripped term the underlined lambda itself must also be present it follows that also $\mathrm{strip}(N'') \leq_\Omega t$. (The second condition is necessary to rule out a situation like $\Omega 1 \leq_\Omega (\lambda y.0)1$, where $(\lambda y.0)1 \rightarrow_{\beta} 0$ but $\Omega 1 \not\leq_\Omega 0$.) Because $N' \twoheadrightarrow_{\text{es}} N''$ we also have that $\mathrm{strip}(N') \leq_\Omega \mathrm{strip}(N'')$ and therefore $\mathrm{strip}(N') \leq_\Omega t$.

$t \leq_\Omega N^\infty$.

Let $M^\infty \equiv t_0 \to t_1 \to t_2 \to \cdots t$. For all $n \geq 0$, by definition of limit, there is an m such that for all $i \geq m$ we have that $(t_i)^n \equiv t^n$. By Lemma 15.5 we have a reduction $M \twoheadrightarrow_{\lambda\circ} M''$ with $t_m \equiv M''^\infty$. We can unwind this M'' to a term M' that satisfies $t^n \leq_\Omega \mathrm{strip}(M')$. Let N' be the common reduct of M' and N. We have that $M' \twoheadrightarrow_{\beta\circ} N'$ by doing underlined $\beta\circ$-redexes and $N \twoheadrightarrow_{\text{es,lift}} N'$. This follows from the fact that the reduction from M^∞ to t_m only contracts underlined redexes, implying that the sequence from M to M' only does underlined $\beta\circ$, es and lift redexes. Since t^n contains no underlining, the redexes contracted from M' to N' can be reachable only by paths longer than n steps. This means that $t^n \leq_\Omega \mathrm{strip}(N')$. It also follows that $t^n \leq_\Omega N'^\infty$. Because es and lift steps preserve the unwinding we have that $N^\infty \equiv N'^\infty$ so $t^n \leq_\Omega N^\infty$.

□

The semantics of $\lambda\circ$ as provided by the 111-calculus differs from the term model developed in the previous section in that it distinguishes between ΩM and Ω. Since we distinguish $\lambda x.\Omega$ from Ω and equate ΩM to Ω, our model corresponds to the infinitary normal form in the 101-calculus extended with 111-terms. The relationship between our model and the 111-normal form (denoted $\mathrm{nf}(t)$, for every t) is given by the following proposition:

Proposition 15.9. *Given $M \in \Lambda\circ$. Then $\bigsqcup(\mathrm{Inf}(M)) \leq_\Omega \mathrm{nf}(M^\infty)$.*

Proof. We have to show that for every $a \in \mathrm{Inf}(M)$ we have that $a \leq_\Omega \mathrm{nf}(M^\infty)$. Suppose $M \twoheadrightarrow N$. We have that $\omega(N) \leq_\Omega \mathrm{strip}(N)$. Because $\mathrm{strip}(N) \in \mathrm{Inf}_{\mathcal{R}_2}(N)$ it follows that $\omega(N) \leq_\Omega \bigsqcup(\mathrm{Inf}_{\mathcal{R}_2}(N))$. By Proposition 15.3 it then follows that $\omega(N) \leq_\Omega N^\infty$. Because of soundness of $\lambda\circ$ with respect to the infinitary lambda calculus (Theorem 15.8) we have that $M^\infty \twoheadrightarrow^{\leq\omega} N^\infty$. Because of the confluence of the infinitary lambda calculus this means that $\mathrm{nf}(M^\infty) \equiv \mathrm{nf}(N^\infty)$. It is easy to show from $N^\infty \twoheadrightarrow^{\leq\omega} \mathrm{nf}(N^\infty) \equiv \mathrm{nf}(M^\infty)$ that $\omega(N) \leq_\Omega \mathrm{nf}(M^\infty)$. □

16. The cyclic lambda calculus and an explicit substitution calculus

Our cyclic calculus does not follow the tradition of the explicit substitution calculi [Les94, Ros92, ACCL91], since the substitution is not a local operation. In this section we introduce a cyclic explicit substitution calculus and show that the associated rewriting system produces the same infinite normal form.

DEFINITION 16.1. The cyclic explicit calculus ($\lambda\circ_{\text{explicit}}$) has the following axioms.

$$\begin{array}{rcll}
\multicolumn{4}{l}{\beta\circ\textit{:}} \\
(\lambda x.M)\ N & = & \langle M \mid x = N\rangle & \\
\multicolumn{4}{l}{\textit{Substitution:}} \\
\langle x \mid x = M, D\rangle & = & \langle M \mid x = M, D\rangle & \\
\langle x \mid D\rangle & = & x & D \perp x \\
\multicolumn{4}{l}{\textit{Distribution:}} \\
\langle M\ N \mid D\rangle & = & \langle M \mid D\rangle\ \langle N \mid D\rangle & \\
\langle \lambda x.M \mid D\rangle & = & \lambda x.\langle M \mid D\rangle & x \text{ not free in } D
\end{array}$$

The rewriting relation associated with the calculus is obtained by orienting all axioms from left to right and is denoted by $\xrightarrow[\text{explicit}]{}$.

Next, we are going to show that $\lambda\circ_{\text{explicit}}$ defines an abstract reduction system with ordered information content. In computing the information content, instead of sending explicit substitution redexes to Ω, we replace the entire letrec by Ω. This is so since the presence of a letrec stands for work to do.

PROPOSITION 16.2. $\langle(\Lambda\circ, \xrightarrow[\text{explicit}]{}, \preceq_{\text{explicit}}), \omega_{\text{explicit}}, (\omega_{\text{explicit}}(\Lambda\circ), \leq_\Omega)\rangle$ *is an ARS with ordered information content, where $\omega_{explicit}$ returns the normal form of a term with respect to the following rules:*

$$\begin{array}{rcl}
\langle M \mid D\rangle & \xrightarrow[\omega_{\text{explicit}}]{} & \Omega \\
(\lambda x.M)\ N & \xrightarrow[\omega_{\text{explicit}}]{} & \Omega \\
\Omega\ M & \xrightarrow[\omega_{\text{explicit}}]{} & \Omega
\end{array}$$

and $M \preceq_{\text{explicit}} N$ if $\omega_{\text{explicit}}(M) \leq_\Omega \omega_{\text{explicit}}(N)$.

PROOF. All requirements except monotonicity of $\xrightarrow[\text{explicit}]{}$ with respect to $\preceq_{\text{explicit}}$ are easy. Let $M \xrightarrow[\text{explicit}]{} N$ be an instance of a rewriting rule. We have that $\omega_{\text{explicit}}(M) = \Omega$ and so $M \preceq_{\text{explicit}} N$. By a simple induction on the context C one can prove that if $M \preceq_{\text{explicit}} N$ then $C[M] \preceq_{\text{explicit}} C[N]$. $\square$

The rewriting system $\xrightarrow[\text{explicit}]{}$ is confluent on the set of acyclic terms, but not necessarily on cyclic terms. For example, we may rewrite $\langle(\lambda x.\langle y \mid y = y\ x\ z\rangle)\ 0 \mid z = 1\rangle$ to both $\langle\langle\langle y \mid y = y\ x\ z\rangle \mid x = 0\rangle \mid z = 1\rangle$ and $\langle\lambda x.\langle y \mid y = y\ x\ z\rangle \mid z = 1\rangle\ \langle 0 \mid z = 1\rangle$, which have no common reduct. However, the call-by-name and explicit substitution infinite normal forms are the same. To prove this we develop the necessary machinery to show that for every cyclic term M, $\text{Inf}_{\lambda\circ}(M) = \text{Inf}_{\text{explicit}}(M)$.

PROPOSITION 16.3. $\omega_{\text{explicit}}(\Lambda\circ) = \omega(\Lambda\circ)$.

PROOF.

"$\subseteq$" From the fact that for every $M \in \Lambda\circ$ we have that $\omega(\omega_{\text{explicit}}(M)) = \omega_{\text{explicit}}(M)$ it follows that

$$\omega(\omega_{\text{explicit}}(\Lambda\circ)) = \omega_{\text{explicit}}(\Lambda\circ) \ .$$

From the fact that $\omega_{\text{explicit}}(\Lambda\circ) \subseteq \Lambda\circ$ it follows that

$$\omega(\omega_{\text{explicit}}(\Lambda\circ)) \subseteq \omega(\Lambda\circ) \ .$$

Hence

$$\omega_{\text{explicit}}(\Lambda\circ) \subseteq \omega(\Lambda\circ) \ .$$

"$\supseteq$" Symmetric argument.

□

PROPOSITION 16.4. *Given $M, N \in \Lambda\circ$. If $\lambda\circ_{\text{explicit}} \vdash M = N$ then $\lambda\circ \vdash M = N$.*

PROOF. Trivial except for distribution over an application:

$$\begin{array}{rcll} \langle M \mid D\rangle\langle N \mid D\rangle & \equiv & \langle M \mid D\rangle\langle N' \mid D'\rangle & \alpha\text{-conversion} \\ & = & \langle M\langle N' \mid D'\rangle \mid D\rangle & \text{lift} \\ & = & \langle\langle M \ N' \mid D'\rangle \mid D\rangle & \text{lift} \\ & = & \langle M \ N' \mid D', D\rangle & \text{merge} \\ & = & \langle M \ N \mid D\rangle & \text{copy} \end{array}$$

□

COROLLARY 16.5. *Given $M, N \in \Lambda\circ$. If $M \underset{\text{explicit}}{\twoheadrightarrow} N$ then $\mathrm{Inf}_{\lambda\circ}(M) = \mathrm{Inf}_{\lambda\circ}(N)$.*

PROOF. Trivial from the previous proposition and the uniqueness of call-by-name infinite normal forms. □

LEMMA 16.6. *The subsystem containing the distribution rules and the garbage collecting rule (second substitution rule) is terminating.*

PROOF. See [AK]. □

We denote the distribution/garbage collection normal form of any term M by $\mathrm{nf}_{\text{dgc}}(M)$.

PROPOSITION 16.7. *Given $M \in \Lambda\circ$.*

(i) $\omega_{\text{explicit}}(M) \leq_\Omega \omega(M)$

(ii) $\omega_{\text{explicit}}(\mathrm{nf}_{\text{dgc}}(M)) = \omega(\mathrm{nf}_{\text{dgc}}(M))$

(iii) $\omega(M) = \omega(\mathrm{nf}_{\text{dgc}}(M))$

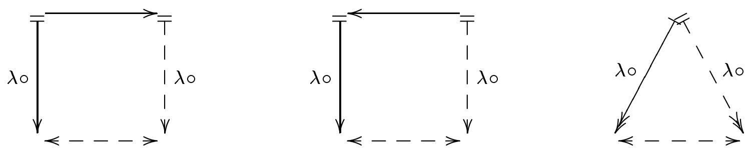
PROOF. Trivial. □

LEMMA 16.8. *Given $M, N \in \Lambda\circ$. If $M \underset{\lambda\circ}{\mapsto\!\!\!\twoheadrightarrow} N$ then $M \underset{\text{explicit}}{\twoheadrightarrow} N'$ with $\omega(N) = \omega(N')$.*

PROOF. Let $\leftrightarrow$ denote convertibility with respect to lift, merge, garbage collection and copying. The result then follows from the following diagram:

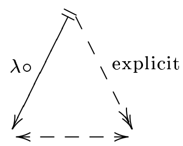
$$\begin{tikzcd} & \cdot \arrow[dl, two heads, "\lambda\circ"'] \arrow[dr, dashed, two heads, "\text{explicit}"] & \\ \cdot \arrow[rr, dashed, leftrightarrow] & & \cdot \end{tikzcd} \tag{16.1}$$

As in the proof of the last case of Lemma 12.8, we use the extended $\beta\circ$ rule β^* and the modified standard reduction. We recall from equation 12.5 the following diagrams:



(16.2)

Diagram 16.1 follows from these diagrams plus the diagram below:



We prove this diagram by case analysis on the modified standard step:

- If the standard step is a β^* we have:

$$\begin{tikzcd}[column sep=small] & \langle \cdots \langle \lambda x.M \mid D_1\rangle \cdots D_n\rangle\ N \arrow[dl, "\beta^*"'] \arrow[dr, dashed, twoheadrightarrow, "\text{explicit}"] & \\ \langle \cdots \langle\langle M \mid x = N\rangle \mid D_1\rangle \cdots D_n\rangle \arrow[rr, dashed, leftrightarrow] & & \langle\langle \cdots \langle M \mid D_1\rangle \cdots D_n\rangle \mid x = N\rangle \end{tikzcd}$$

  The explicit reduction consists of n lambda distribution steps followed by a $\beta\circ$ step and the bottom conversion consists of $2n$ merge steps.

- If the standard step is a substitution we have:

$$\begin{tikzcd} M \arrow[d, phantom, "\equiv"] \arrow[dr, dashed, twoheadrightarrow, "\text{explicit}"] & & \\ \underline{M} \arrow[d, "\text{es}"'] \arrow[r, dashed, leftrightarrow] & \mathrm{nf}_{\mathrm{dgc}}(M) \arrow[d, dashed, "\text{es}"'] \arrow[dr, dashed, twoheadrightarrow, "\text{explicit}"] & \\ {} \arrow[r, dashed, leftrightarrow] & {} \arrow[r, phantom, "\equiv"] & {} \end{tikzcd}$$

  The top triangle follows from the fact that for every term M we have that $M \leftrightarrow \mathrm{nf}_{\mathrm{dgc}}(M)$ (see the proof of Lemma 16.4 to convince ourselves that no other rules other than lift, merge and copy are used). The square follows from the left two diagrams of 16.2. The bottom triangle follows from the fact that if $\mathrm{nf}_{\mathrm{dgc}}(M) \underset{\text{es}}{\mapsto} M'$ then we must have that $M \equiv C[\langle x \mid x = N, D\rangle]$ and $M' \equiv C[\langle N \mid x = N, D\rangle]$, which is also a legal step in the explicit substitution calculus.

□

THEOREM 16.9. *Given a term* $M \in \Lambda\circ$. $\mathrm{Inf}_{\text{explicit}}(M) = \mathrm{Inf}_{\lambda\circ}(M)$.

PROOF.

$\subseteq$ Given $a \in \mathrm{Inf}_{\text{explicit}}(M)$. By definition there exists an N such that $M \xrightarrow[\text{explicit}]{} \!\!\!\!\rightarrow N$ and $a \leq_\Omega \omega_{\text{explicit}}(N)$. By Proposition 16.7(i) we have that $\omega_{\text{explicit}}(N) \leq_\Omega \omega(N)$, hence $a \leq_\Omega \omega(N)$. By Proposition 16.3 it means that a is in ω-normal form, hence $a \in \mathrm{Inf}_{\lambda\circ}(N)$. The result then follows from Corollary 16.5.

$\supseteq$ Given $a \in \mathrm{Inf}_{\lambda\circ}(M)$. By definition there exists an N such that $M \mapsto\!\!\!\twoheadrightarrow N$ and $a \leq_\Omega \omega(N)$. By Lemma 16.8 we can find an N' such that $M \xrightarrow[\text{explicit}]{}\!\!\!\twoheadrightarrow N'$ with $\omega(N) = \omega(N')$. Using Proposition 16.7(ii,iii) we may conclude that $\omega_{\text{explicit}}(\mathrm{nf}_{\text{dgc}}(N')) = \omega(N')$, hence $a \leq_\Omega \omega_{\text{explicit}}(\mathrm{nf}_{\text{dgc}}(N'))$. Since by Proposition 16.3 a is in ω_{explicit}-normal form we conclude that $a \in \mathrm{Inf}_{\text{explicit}}(M)$.

$\square$

PROPOSITION 16.10. *Given* $\langle(\Lambda\circ, \xrightarrow[\text{explicit}]{}, \preceq_{\text{explicit}}), \omega_{\text{explicit}}, (\omega_{\text{explicit}}(\Lambda\circ), \leq_\Omega)\rangle$. $\xrightarrow[\text{explicit}]{}$ *is confluent up to* $\preceq_{\text{explicit}}$.

PROOF. From Corollary 16.5 and the previous theorem it follows that the explicit substitution calculus has unique infinite normal forms. Hence, by Lemma 10.18 it follows that $\xrightarrow[\text{explicit}]{}$ is confluent up to $\preceq_{\text{explicit}}$. $\square$

In practice, explicit substitution calculi tend to have more axioms about substitutions. For example, they have (a suitable restriction of)

$$\langle\langle M \mid x_1 = M_1, \cdots, x_n = M_n\rangle \mid D\rangle = \langle M \mid x_1 = \langle M_1 \mid D\rangle, \cdots, x_n = \langle M_n \mid D\rangle\rangle\ .$$

All those additional axioms preserve the tree unwinding, thus we can derive them in $\lambda\circ$. It then follows easily that adding those axioms to our explicit substitution calculus does not change the infinite normal form of a term.

Part III
The sharing calculi: strictness vs non-strictness

17. The cyclic sharing calculus $\lambda\circ_{\mathrm{share}}$

So far we have developed a precise connection between terms of the lambda calculus extended with `letrec` and the class of well-formed cyclic lambda-graphs. On the set of cyclic terms we have defined an axiom system (called the representational system) that equates terms that represent the same cyclic graph and terms that give rise to the same possibly infinite tree. The representational axiom system combined with a notion of β-reduction constitutes our axiomatization of cyclic lambda structures. A drawback of this cyclic lambda calculus is that it does not support sharing adequately, since it allows reductions that duplicate work. For example, in the following reduction:

$$\langle MxNx \mid x = (\lambda y.y)(\lambda y.y)\rangle \xrightarrow[\lambda\circ]{} \!\!\!\!\!\rightarrow \langle M((\lambda y.y)(\lambda y.y))N((\lambda y.y)(\lambda y.y)) \mid x = (\lambda y.y)(\lambda y.y)\rangle \ ,$$

the $\beta\circ$-redex has been duplicated. Current implementations of functional languages such as Haskell [HPJW+92] and Id [Nik91] do not allow these kinds of reductions. Therefore, we develop a variant of the cyclic calculus that takes sharing into consideration. We emphasize that we are only interested in capturing the sharing present in current implementations (lazy and lenient) of non-strict languages. We do not study the sharing present in optimal (in the sense of Lévy [Lév78]) implementations of lambda calculus. Since the emphasis is on sharing and not on a specific reduction strategy, we call the calculus the sharing calculus ($\lambda\circ_{\mathrm{share}}$), because call-by-need normally implies lazy evaluation.

The sharing calculus is obtained by restricting the operations that cause duplication, such as substitution and copying, so that only values are duplicated, where a value is either a variable or a lambda-abstraction. The restriction on the copy axiom forbids the following equalities:

$$\begin{array}{l}\langle x \mid x = x\ x\rangle = \langle x \mid x = y\ y, y = x\ x\rangle \\ \langle x\ x \mid x = (\lambda z.z\ z)\ (\lambda z.z\ z)\rangle = \langle x\ y \mid x = (\lambda z.z\ z)\ (\lambda z.z\ z), y = (\lambda z.z\ z)\ (\lambda z.z\ z)\rangle \ ,\end{array}$$

because it would require x and y, which are not bound to values to be identified by the variable mapping. The following are legal instances of the copy axiom:

$$\begin{array}{l}\langle x \mid x = \lambda z.\langle u \mid u = x\ x\rangle\rangle = \langle x \mid x = \lambda z.\langle u \mid u = y\ y\rangle, y = \lambda z.\langle v \mid v = x\ x\rangle\rangle \\ \langle x\ x \mid x = x\rangle = \langle x\ y \mid x = x, y = y\rangle \ .\end{array}$$

Also, the lambda-lift axiom has to be restricted to lift values only, since lifting unevaluated expressions out of a lambda-abstraction has an impact on the amount of sharing captured.

We add the two following syntactic clauses of values and value declarations

$$\begin{array}{rcl} V & ::= & x \mid \lambda x.M \\ VD & ::= & x_1 = V_1, \ldots, x_n = V_n \end{array}$$

to the ones of Definition 3.1.

Definition 17.1. The cyclic sharing calculus ($\lambda\circ_{\mathrm{share}}$) has the following axioms.

$(\lambda x.M)N$	$\xrightarrow[\beta\circ]{}$	$\langle M \mid x = N\rangle$	
$\langle C[x] \mid x = V, D\rangle$	$\xrightarrow[\mathrm{es_V}]{}$	$\langle C[V] \mid x = V, D\rangle$	
$\langle M \mid x = C[x_1], x_1 = V, D\rangle$	$\xrightarrow[\mathrm{is_V}]{}$	$\langle M \mid x = C[V], x_1 = V, D\rangle$	
$\langle M \mid D\rangle N$	$\xrightarrow[\mathrm{lift}]{}$	$\langle MN \mid D\rangle$	
$M\langle N \mid D\rangle$	$\xrightarrow[\mathrm{lift}]{}$	$\langle MN \mid D\rangle$	
$\lambda x.\langle M \mid D, VD\rangle$	$\xrightarrow[\mathrm{lift}]{}$	$\langle \lambda x.\langle M \mid D\rangle \mid VD\rangle$	$D \perp VD, VD \neq \{\}$ and x not free in VD
$\langle\langle M \mid D\rangle \mid D'\rangle$	$\xrightarrow[\mathrm{em}]{}$	$\langle M \mid D, D'\rangle$	
$\langle M \mid x = \langle N \mid D\rangle, D_1\rangle$	$\xrightarrow[\mathrm{im}]{}$	$\langle M \mid x = N, D, D_1\rangle$	
$\langle M \mid D, D'\rangle$	$\xrightarrow[\mathrm{gc}]{}$	$\langle M \mid D\rangle$	$\{\} \neq D', D' \perp \langle M \mid D\rangle$
$\langle M \mid \rangle$	$\xrightarrow[\mathrm{gc}_\square]{}$	M	
M	$\xrightarrow[\mathrm{cp_V}]{}$	N	$\exists\sigma : \mathcal{V} \to \mathcal{V}, N^\sigma \equiv M$ and $\forall x \not\equiv x', \sigma(x) \equiv \sigma(x'), \sigma(x)$ bound to a value in M
$C_{\mathrm{safe}}[M\ N]$	$\xrightarrow[\mathrm{name}]{}$	$C_{\mathrm{safe}}[\langle x \mid x = M\ N\rangle]$	x a new variable

C_{safe} is given by:

$$\begin{array}{rcl} C_{\mathrm{safe}} & ::= & C' \mid C[\lambda x.C'] \mid C[C'\ M] \mid C[M\ C'] \ . \\ C' & ::= & \square \mid \langle C' \mid D\rangle \end{array}$$

Table 6. The rewrite relation for the sharing calculus $\xrightarrow[\lambda\circ_{\mathrm{share}}]{}$

$\beta\circ$:

$$(\lambda x.M)N = \langle M \mid x = N\rangle$$

Substitution:

$$\langle C[x] \mid x = V, D\rangle = \langle C[V] \mid x = V, D\rangle$$
$$\langle M \mid x = C[x_1], x_1 = V, D\rangle = \langle M \mid x = C[V], x_1 = V, D\rangle$$

Lift:

$$\langle M \mid D\rangle N = \langle MN \mid D\rangle$$
$$M\langle N \mid D\rangle = \langle MN \mid D\rangle$$
$$\lambda x.\langle M \mid D, VD\rangle = \langle \lambda x.\langle M \mid D\rangle \mid VD\rangle \qquad D \perp VD \text{ and } x \text{ not free in } VD$$

Merge:

$$\langle M \mid x = \langle N \mid D\rangle, D_1\rangle = \langle M \mid x = N, D, D_1\rangle$$
$$\langle\langle M \mid D\rangle \mid D'\rangle = \langle M \mid D, D'\rangle$$

Garbage collection:

$$\langle M \mid D, D'\rangle = \langle M \mid D\rangle \qquad D' \perp \langle M \mid D\rangle$$
$$\langle M \mid \rangle = M$$

Copying:

$$M = N \qquad \exists\sigma : \mathcal{V} \to \mathcal{V}, N^\sigma \equiv M \text{ and } \forall x \not\equiv x', \sigma(x) \equiv \sigma(x'), \sigma(x) \text{ bound to a value in } M$$

Naming:

$$M = \langle x \mid x = M\rangle \qquad x \text{ a new variable}$$

The rewrite relation $\xrightarrow[\lambda\circ_{\mathrm{share}}]{}$ associated with the calculus is given in Table 6.

Note that the sharing calculus adds the naming axiom, since it is no longer derivable. However, in the rewriting system naming is only included for applications and is restricted to safe contexts.

$\lambda\circ_{\mathrm{share}}$ extends the cyclic calculus (λ_{need}) presented in [AFM+95, AF97], since reductions may occur when they are not needed, *e.g.,* λ_{need} disallows the following reduction:

$$\langle x \mid x = \lambda y.wx, w = \lambda z.z\rangle \twoheadrightarrow \langle x \mid x = \lambda y.(\lambda z.z)x\rangle \twoheadrightarrow \langle x \mid x = \lambda y.x\rangle \ .$$

Moreover, in [AFM$^+$95, AF97], the soundness and completeness of λ_{need} with respect to traditional lambda calculus were limited to the acyclic case.

The rewriting system and the calculus identify the same terms. This is formalized in the following proposition.

PROPOSITION 17.2. *Given $M, N \in \Lambda\circ$. Then $M \xleftrightarrow[\lambda\circ_{\mathrm{share}}]{} N$ iff $\lambda\circ_{\mathrm{share}} \vdash M = N$.*

PROOF. All rules and axioms are in 1-1 correspondence, except naming. The non-trivial part is showing that if x is a new variable then $C[M] \xleftrightarrow[\lambda\circ_{\mathrm{value}}]{} C[\langle x \mid x = M\rangle]$. We can do this by a case analysis on M and C, as in the proof of Proposition 4.6. $\square$

17.1. Soundness and completeness of $\lambda\circ_{\mathrm{share}}$ with respect to $\lambda\circ_{\mathrm{name}}$

Soundness of $\lambda\circ_{\mathrm{share}}$ follows from the fact that the sharing theory is a subset of the call-by-name theory. In [AFM$^+$95, AF97], we proved the completeness of (acyclic) λ_{need} with respect to lambda calculus using a simple invariant:

$$M \xrightarrow[name]{} \!\!\!\!\!\rightarrow N \Rightarrow \exists P, N', M \xrightarrow[need]{} \!\!\!\!\!\rightarrow P, N \xrightarrow[name]{} \!\!\!\!\!\rightarrow N', N' \leq P \ .$$

The ordering $\leq$ was a syntactic ordering capturing the amount of sharing in a term. To show completeness of $\lambda\circ_{\mathrm{share}}$ this invariant is too strong. We need to compare information content. Intuitively, we want to say that if M reduces to N in $\lambda\circ_{\mathrm{name}}$, then the information contained in N can be obtained by reducing M in $\lambda\circ_{\mathrm{share}}$. However, this does not hold. Consider the reduction $\langle x \mid x = yy\rangle \xrightarrow[\lambda\circ_{\mathrm{name}}]{} \!\!\!\!\!\rightarrow yy$. Since yy is stable information, we would expect to get that information in $\lambda\circ_{\mathrm{share}}$. But this information is not reachable in $\lambda\circ_{\mathrm{share}}$ since yy is not a value. This shows that if we want to compare $\lambda\circ$ and $\lambda\circ_{\mathrm{share}}$, we need a new notion of information content for the sharing calculus which we call *printable information.* This notion, as opposed to the call-by-name information content, can be infinite. Consider the term $\langle x \mid x = \lambda y.yx\rangle$. Its information content is Ω, whereas its printing value is the sequence $\lambda y.y\Omega$,$\lambda y.y(\lambda y.y\Omega)$, $\lambda y.y(\lambda y.y(\lambda y.y\Omega)), \cdots$. Both the information content and the printable information of $\langle x \mid x = x(\lambda y.y)\rangle$ are Ω.

DEFINITION 17.3. Given $M \in \Lambda\circ$.

(i) The printable information of M, written as $\mathrm{print}(M)$, is

$$\downarrow \{\omega(M_1) \mid M \xrightarrow[\mathrm{es}]{} \!\!\!\!\!\rightarrow M_1\} \ ,$$

where the downward closure is with respect to $\omega(\Lambda\circ)$ and $\leq_\Omega$.

(ii) $M \preceq_{\mathrm{share}} N$ if $\mathrm{print}(M) \subseteq \mathrm{print}(N)$.

(iii) The infinite normal form of M in the sharing calculus is defined as:

$$\mathrm{Inf}_{\mathrm{share}}(M) = \bigcup \{\mathrm{print}(M_1) \mid M \xrightarrow[\mathrm{share}]{} \!\!\!\!\!\rightarrow M_1\} \ .$$

Our invariant becomes: each *finite information* obtained by reducing a term M in $\lambda\circ_{\mathrm{name}}$ can be obtained by reducing M in $\lambda\circ_{\mathrm{share}}$ and then printing the result.

LEMMA 17.4. *Given $M \in \Lambda\circ$. If $M \xrightarrow[\lambda\circ_{\mathrm{name}}]{} \!\!\!\!\!\rightarrow N$ then there exists a term P such that $M \xrightarrow[\lambda\circ_{\mathrm{share}}]{} \!\!\!\!\!\rightarrow P$ and $\omega(N) \in \mathrm{print}(P)$.*

PROOF. Because standard reduction is complete up to information content (Lemma 12.9) we may assume that $M \xrightarrow[\lambda\circ]{}\!\!\!\!\twoheadrightarrow N$. We will now prove the statement by induction on the number of $\beta\circ$/es-steps in the standard reduction $M \xrightarrow[\lambda\circ]{}\!\!\!\!\twoheadrightarrow N$.

Given a standard reduction $M \xrightarrow[\lambda\circ]{}\!\!\!\!\twoheadrightarrow N$ with n $\beta\circ$/es-steps we can reduce M to some left lift and internal merge normal form M'. By Lemma 12.8 there exists an N' such that $M' \xrightarrow[\lambda\circ]{}\!\!\!\!\twoheadrightarrow N'$ in no more than n $\beta\circ$/es-steps and $N \preceq_{\text{name}} N'$. Now there are two possibilities:

- The sequence $M' \xrightarrow[\lambda\circ]{}\!\!\!\!\twoheadrightarrow N'$ contracts only es-redexes. We are done with $P \equiv M'$.
- Some non-es-redex is done in $M' \xrightarrow[\lambda\circ]{}\!\!\!\!\twoheadrightarrow N'$. Because M' is in left lift/internal merge normal form we must have that the sequence $M' \xrightarrow[\lambda\circ]{}\!\!\!\!\twoheadrightarrow N'$ is of the form $M' \xrightarrow[\lambda\circ]{}\!\!\!\!\twoheadrightarrow Q_1 \xrightarrow[\lambda\circ]{R} Q_2 \xrightarrow[\lambda\circ]{}\!\!\!\!\twoheadrightarrow N'$, where each step in $M' \xrightarrow[\lambda\circ]{}\!\!\!\!\twoheadrightarrow Q_1$ is a substitution of an application and where R is the first redex that is not a substitution of an application. This redex R can only be a substitution of a value or a $\beta\circ$-redex. In both cases the step R is the descendant of a sharing redex R' in M'. If we denote by M'' the result of contracting R' in M' then by Lemma 12.8 there exists an N'' such that $M'' \xrightarrow[\lambda\circ]{}\!\!\!\!\twoheadrightarrow N''$ in less than n $\beta\circ$/es-steps and $N' \preceq_{\text{name}} N''$. By induction hypothesis we can find a P with $M'' \xrightarrow[\lambda\circ_{\text{share}}]{}\!\!\!\!\twoheadrightarrow P$ and $N'' \preceq_{\text{name}} P$.

□

THEOREM 17.5. *Given* $M \in \Lambda\circ$. $\text{Inf}(M) = \text{Inf}_{\text{share}}(M)$.

PROOF.

"$\subseteq$" Follows from Lemma 17.4.

"$\supseteq$" Obvious.

□

REMARK 17.6. Note that even though $\langle(\Lambda\circ, \xrightarrow[\lambda\circ_{\text{share}}]{}, \preceq_{\text{share}}), \text{print}, (\text{print}(\Lambda\circ), \subseteq)\rangle$ is an ARS with ordered information content, $\xrightarrow[\lambda\circ_{\text{share}}]{}$ is not confluent up to $\preceq_{\text{share}}$. Consider the following two reductions:

$$\begin{aligned} M \equiv \langle x \mid x = \lambda z.z\ y, y = \lambda z'.z'\ (x\ z')\rangle \ &\xrightarrow[\lambda\circ_{\text{share}}]{}\!\!\!\!\twoheadrightarrow\ \langle \lambda z.z\ y \mid y = \lambda z'.z'\ ((\lambda z.z\ y)\ z')\rangle \\ &\xrightarrow[\lambda\circ_{\text{share}}]{}\ \langle \lambda z.z\ y \mid y = \lambda z'.z'\ (z'\ y)\rangle \equiv M_1 \end{aligned}$$

and

$$M \equiv \langle x \mid x = \lambda z.z\ y, y = \lambda z'.z'\ (x\ z')\rangle \twoheadrightarrow \langle x \mid x = \lambda z.z\ (\lambda z'.z'\ (x\ z'))\rangle \equiv M_2 \ .$$

We have that $\text{print}(M_1) = \text{Inf}_{\text{share}}(M_1)$, because the only redexes in M_1 and any of its reducts are value substitutions, which are done by the print function. However, there cannot exist M_3 such that $M_2 \xrightarrow[\lambda\circ_{\text{share}}]{}\!\!\!\!\twoheadrightarrow M_3$ and $\text{print}(M_1) \subseteq \text{print}(M_3)$ because $\text{print}(M_1)$ is infinite whereas the print of any reduct of M_2 is finite. The problem is that in the unwinding of M we have an infinite number of β-redexes. When we rewrite M into M_1 we do all of those redexes and when we rewrite M into M_2 we destroy the opportunity to do them one step.

This does not contradict Proposition 10.18, since the sharing infinite normal form is not defined as in Definition 10.15.

Not all the axioms of the sharing calculus are needed for evaluation. In Table 7 we present a set of axioms that is sufficient and necessary for evaluation. We then define the sharing infinite normal form based on this shared evaluation calculus by:

$$\text{Inf}_{\text{eval}}^{\text{share}}(M) = \bigcup\{\text{print}(N) \mid M \xrightarrow[\text{evalV}]{}\!\!\!\!\twoheadrightarrow N\} \ .$$

$\beta\circ$:		
$(\lambda x.M)N$	$\xrightarrow[\mathrm{eval_V}]{}$	$\langle M \mid x = N\rangle$
Lift:		
$\langle M \mid D\rangle N$	$\xrightarrow[\mathrm{eval_V}]{}$	$\langle MN \mid D\rangle$
External substitution:		
$\langle C[x] \mid x = V, D\rangle$	$\xrightarrow[\mathrm{eval_V}]{}$	$\langle C[V] \mid x = V, D\rangle$
Internal substitution:		
$\langle M \mid y = C[x], x = V, D\rangle$	$\xrightarrow[\mathrm{eval_V}]{}$	$\langle M \mid y = C[V], x = V, D\rangle$
Internal merge:		
$\langle M \mid x = \langle N \mid D\rangle, D'\rangle$	$\xrightarrow[\mathrm{eval_V}]{}$	$\langle M \mid x = N, D, D'\rangle$

Table 7. Sharing evaluation calculus: $\lambda\circ_{\mathrm{eval_V}}$

Values:	V	::=	$\lambda x.M$
Answers	A	::=	$\lambda x.M \mid \langle A \mid D\rangle$
Dependencies:	$D[x, x_n]$	::=	$x = E_{\mathrm{lazy}}[x_1], \cdots, x_{n-1} = E_{\mathrm{lazy}}[x_n], D$
Lazy Evaluation context:	E_{lazy}	::=	$[\,] \mid E_{\mathrm{lazy}}M \mid \langle E_{\mathrm{lazy}} \mid D\rangle \mid \langle E_{\mathrm{lazy}}[x] \mid x = E_{\mathrm{lazy}}, D\rangle$ $\langle E_{\mathrm{lazy}}[x] \mid D[x, x_n], x_n = E_{\mathrm{lazy}}\rangle$

$E_{\mathrm{lazy}}[(\lambda x.M)N]$	$\xmapsto[\mathrm{lazy}]{}$	$E_{\mathrm{lazy}}[\langle M \mid x = N\rangle]$
$E_{\mathrm{lazy}}[\langle E_{\mathrm{lazy}}[x] \mid x = V, D\rangle]$	$\xmapsto[\mathrm{lazy}]{}$	$E_{\mathrm{lazy}}[\langle E_{\mathrm{lazy}}[V] \mid x = V, D\rangle]$
$E_{\mathrm{lazy}}[\langle E_{\mathrm{lazy}}[x] \mid D[x, x_n], x_n = V\rangle]$	$\xmapsto[\mathrm{lazy}]{}$	$E_{\mathrm{lazy}}[\langle E_{\mathrm{lazy}}[x] \mid D[x, V], x_n = V\rangle]$
$E_{\mathrm{lazy}}[\langle A \mid D\rangle N]$	$\xmapsto[\mathrm{lazy}]{}$	$E_{\mathrm{lazy}}[\langle AN \mid D\rangle]$
$E_{\mathrm{lazy}}[\langle E_{\mathrm{lazy}}[x] \mid x = \langle A \mid D\rangle, D_1\rangle]$	$\xmapsto[\mathrm{lazy}]{}$	$E_{\mathrm{lazy}}[\langle E_{\mathrm{lazy}}[x] \mid x = A, D, D_1\rangle]$
$E_{\mathrm{lazy}}[\langle E_{\mathrm{lazy}}[x] \mid D[x, x_n], x_n = \langle A \mid D\rangle\rangle]$	$\xmapsto[\mathrm{lazy}]{}$	$E_{\mathrm{lazy}}[\langle E_{\mathrm{lazy}}[x] \mid D[x, x_n], x_n = A, D\rangle]$

Table 8. Lazy evaluation.

PROPOSITION 17.7. *Given $M \in \Lambda\circ$. Then*

$$\mathrm{Inf}_{\mathrm{share}}(M) = \mathrm{Inf}^{\mathrm{share}}_{\mathrm{eval}}(M).$$

PROOF. Same as the proof of Theorem 17.5 with the additional observation that the reduction sequence constructed in the proof of Lemma 17.4 only uses steps from the shared evaluation calculus. □

17.2. Lazy and lenient strategies of $\lambda\circ_{\mathrm{share}}$

$\lambda\circ_{\mathrm{share}}$ captures the essence of lazy languages, such as Haskell [HPJW+92], and of the functional core of lenient languages, such as Id [Nik91, AMNS97] and Parallel Haskell [AAH+93]. We substantiate

$$E_{\mathrm{lenient}} \quad ::= \quad [\,] \mid E_{\mathrm{lenient}}M \mid ME_{\mathrm{lenient}} \mid \langle E_{\mathrm{lenient}} \mid D\rangle \mid \langle M \mid x = E_{\mathrm{lenient}}, D\rangle$$

Table 9. Lenient evaluation context.

our claim by showing that both the lazy and lenient strategies are complete with respect to different observations. The lazy strategy (written as $\xrightarrow[\text{lazy}]{}\!\!\!\!\twoheadrightarrow$, see Table 8 for corresponding evaluation context and rules), as described by Launchbury [Lau93], only allows one to reach the top stable information. The lenient strategy (written as $\xrightarrow[\text{lenient}]{}\!\!\!\!\twoheadrightarrow$, see Table 9 for corresponding evaluation context) allows reduction of any redex as long as it does not occur under a lambda. Thus, it allows one to reach more information. For example, given $x(I(\lambda z.\Omega))$, the lazy strategy produces $x\Omega$ and the lenient strategy produces $x(\lambda z.\Omega)$. We will prove next two theorems stating the capabilities of the lazy and lenient strategies, but first we introduce some machinery.

DEFINITION 17.8. A term $M \in \Lambda\circ$ is called a black hole if it is of the form $\langle E_{\text{lazy}}[x] | D[x, x]\rangle$ or of the form $\langle E_{\text{lazy}}[x] | D[x, y], D[y, y]\rangle$.

LEMMA 17.9. *Given a term $M \in \Lambda\circ$, M is either an answer or there exists a unique evaluation context E_{lazy} such that $M \equiv E_{\text{lazy}}[N]$, where N is a redex or a black hole, or $M \equiv E_{\text{lazy}}[x]$, where x is a free variable.*

PROOF. We will argue by structural induction on the evaluation context. For simplicity we omit the subscript lazy.

$M \equiv x$: The consequence holds with $E \equiv [\,]$.

$M \equiv \lambda x.N$: Here M is an answer.

$M \equiv M_1M_2$: By the inductive hypothesis, either M_1 is equivalent to an evaluation context filled with a redex or a black hole or a free variable, or it is an answer:

- $M_1 \equiv E_1[N]$, where N is a redex: The claim is true with $E \equiv (E_1\ M_2)$.
- $M_1 \equiv E_1[N]$, where N is a black hole: The claim is true with $E \equiv (E_1\ M_2)$.
- $M_1 \equiv E_1[x]$: The claim holds with $E \equiv (E_1\ M_2)$.
- M_1 is an answer: This sub case demands another case analysis, depending on the nature of M_1:
  - **$M_1 \equiv (\lambda x.M_1')$:** M is a $\beta\circ$-redex; hence, the claim holds for $E \equiv [\,]$.
  - $M_1 \equiv \langle A \mid M_1'\rangle$: M is a *lift*-redex; hence, the claim holds for $E \equiv [\,]$.

$M \equiv \langle M_1 \mid D\rangle$: M_1 is an answer: M is an answer.

- $M_1 \equiv E_1[N]$, where N is a redex: The claim is true with $E \equiv \langle E_1 \mid D\rangle$.
- $M_1 \equiv E_1[N]$, where N is a black hole: The claim is true with $E \equiv \langle E_1 \mid D\rangle$.
- $M_1 \equiv E_1[x_2]$: If x_2 is not defined in D then the claims holds with $E \equiv \langle E_1 \mid D\rangle$. If $D \equiv x_2 = M_2, D_2$ then if
  - M_2 is an answer: Depending on M_2 being a value or not M is an external substitution redex or a box elimination redex.
  - $M_2 \equiv E_2[N]$, where N is a black hole or a redex: The claim is true with

$$E \equiv \langle E_1[x_2] | x_2 = E_2, D_2\rangle$$

  - $M_2 \equiv E_2[x_3]$: If $x_3 \equiv x_2$ then M is a black hole and $E = [\,]$. If x_3 not defined in D_2 then the claim is true with $E \equiv \langle E_1[x_2] | x_2 = E_2, D_2\rangle$ If x_3 is defined we have that $M \equiv \langle E_1[x_2] | x_2 = E_2[x_3], x_3 = M_3, D_3\rangle$ and we go into a loop: If $M \equiv \langle E_1[x_2] | x_2 = E_2[x_3], \ldots, x_{n-1} = E_{n-1}, x_n = M_n, D_n\rangle$ then if

M_n is an answer: Depending on M_2 being a value or not M is an internal substitution redex or a box elimination redex.

$M_n \equiv E_n[N]$, where N is a black hole or a redex: The claim is true with $E \equiv \langle E_1[x_2] | D[x_2, x_n], x_2 = E_n \rangle$

$M_n \equiv E_n[x_{n+1}]$: If x_{n+1} is not defined in D then the claim is true with

$$E \equiv \langle E_1[x_2] | D[x_2, x_n], x_2 = E_n \rangle$$

else if x_{n+1} is equivalent with one of the variables $x_2, \ldots, x_n$ then M is a black hole else repeat the loop.

This loop must terminate because D is finite.

□

PROPOSITION 17.10. *Given a lazy evaluation context E_{lazy} and a black hole N we have that*

$$\text{Inf}(E_{\text{lazy}}[N]) = \{\Omega\} \ .$$

PROOF. By a simple analysis we can prove the following two claims:

- $\omega(E_{\text{lazy}}[N]) = \Omega$
- If $E_{\text{lazy}}[N] \xmapsto[\lambda\circ_{\text{name}}]{} M$ then $M \equiv E'_{\text{lazy}}[N']$ for a black hole N'

Suppose that $\Omega \not\equiv a \in \text{Inf}(E_{\text{lazy}}[N])$. Then by Lemma 12.9 it follows that $E_{\text{lazy}}[N] \xmapsto[\lambda\circ_{\text{name}}]{} \!\!\!\!\rightarrow M$, where $a \leq_\Omega \omega(M)$. However from the earlier two claims it follows that $\omega(M) = \Omega$. Contradiction. □

THEOREM 17.11. *Given $M \in \Lambda\circ$. If $\lambda x.\Omega \in \text{Inf}(M)$ ($x\Omega \cdots \Omega \in \text{Inf}(M)$) then $\exists N \in \Lambda\circ$, $M \xmapsto[\text{lazy}]{} \!\!\!\!\rightarrow N$ with $\lambda x.\Omega \leq_\Omega \omega(N)$ ($x\Omega \cdots \Omega \in \text{print}(N)$).*

PROOF. We know that $M \xmapsto[\lambda\circ_{\text{name}}]{} \!\!\!\!\rightarrow P$ with $\lambda x.\Omega \leq_\Omega \omega(P)$ ($x\Omega \cdots \Omega \leq_\Omega \omega(P)$). By Lemma 17.9 we are in one of the following cases:

M is an answer: We have that $\lambda x.\Omega \leq_\Omega \omega(M)$.

$M \equiv E_{\text{lazy}}[N]$, for a black hole N: Impossible due to Proposition 17.10 and the fact that M standard reduces to a term with information content larger than Ω.

$M \equiv E_{\text{lazy}}[x]$, for a free variable x: A simple analysis shows that $x\Omega \cdots \Omega \in \text{print}(M)$.

$M \equiv E_{\text{lazy}}[R]$, for a redex R: If R is a substitution or a β-redex then analysis shows that until a descendant of R is reduced the standard sequence must maintain information content Ω. Since the information content of the last term in the standard sequence is more than Ω this means that a descendant of R has been contracted and therefore the bottom standard sequence is shorter. Hence, by iteration the result holds. If R is a lift-redex or an internal merge then we just iterate until we are in one of the other cases, which must happen because lift and internal merge together form a terminating system.

□

THEOREM 17.12. *Given $M \in \Lambda\circ$. For all $a \in \text{Inf}(M)$ that are in normal form with respect to the rule $\lambda x.M \to \lambda x.\Omega$ there exists $N \in \Lambda\circ$, $M \xmapsto[\text{lenient}]{} \!\!\!\!\rightarrow N$ with $a \in \text{print}(N)$.*

PROOF. We have a reduction $M \underset{\lambda\circ_{\text{name}}}{\longmapsto\!\!\!\!\twoheadrightarrow} P$ with $a \preceq_{\text{name}} P$. We will prove the statement by induction on the number of $\beta\circ$/es-steps in the sequence $M \underset{\lambda\circ_{\text{name}}}{\longmapsto\!\!\!\!\twoheadrightarrow} P$.

Given a standard reduction $M \underset{\lambda\circ_{\text{name}}}{\longmapsto\!\!\!\!\twoheadrightarrow} P$ with n $\beta\circ$/es-steps. First reduce M to normal form with respect to all left lift and internal merge steps allowed by the lenient strategy to obtain M'. By Lemma 12.8 we can find a P' such that $M' \underset{\lambda\circ_{\text{name}}}{\longmapsto\!\!\!\!\twoheadrightarrow} P'$ in no more than n $\beta\circ$/es-steps and $P \preceq_{\text{name}} P'$. We can reorder this sequence to obtain a sequence $M' \underset{\lambda\circ_{\text{name}}}{\longmapsto\!\!\!\!\twoheadrightarrow} Q \underset{\lambda\circ_{\text{name}}}{\longmapsto\!\!\!\!\twoheadrightarrow} P'$, where the sequence $M' \underset{\lambda\circ_{\text{name}}}{\longmapsto\!\!\!\!\twoheadrightarrow} Q$ only does redexes above a lambda and $Q \underset{\lambda\circ_{\text{name}}}{\longmapsto\!\!\!\!\twoheadrightarrow} P'$ only does redexes below a lambda. This follows from the fact that $P_1 \underset{\lambda\circ_{\text{name}}}{\longmapsto} P_2 \underset{\lambda\circ_{\text{name}}}{\longmapsto} P_3$ where the first step is below a lambda and the second one is not then we can reorder these two steps because the second step is the descendant of a redex present in P_1 and from the fact that this swap operation is terminating. A simple analysis shows that $Q \succeq_{\text{name}} a$. If the sequence $M' \underset{\lambda\circ_{\text{name}}}{\longmapsto\!\!\!\!\twoheadrightarrow} Q$ does only substitution steps we are done with $N \equiv Q$. Otherwise, the reduction $M' \underset{\lambda\circ_{\text{name}}}{\longmapsto\!\!\!\!\twoheadrightarrow} Q$ must be of the form $M' \underset{\lambda\circ_{\text{name}}}{\longmapsto\!\!\!\!\twoheadrightarrow} Q_1 \overset{R}{\underset{\lambda\circ_{\text{name}}}{\longmapsto}} Q_2 \underset{\lambda\circ_{\text{name}}}{\longmapsto\!\!\!\!\twoheadrightarrow} Q$, where R is the first redex that is not a substitution of an application. Let us analyze the possibilities for R:

- lift redex: Impossible. A lift redex cannot be created by a substitution of an application so R would be present in M', but M' has no lift redexes above lambda's and R must be above lambda.

- substitution redex, other than of an application: R must be a value substitution. The substituted term cannot be a letrec, because that would mean that M' had an internal merge redex above lambda. We also have that R is the descendant of a value substitution redex R' in M'. Let M'' denote the result of contracting R' in M'. By Lemma 12.8 we can find an N'' such that $M'' \underset{\lambda\circ_{\text{name}}}{\longmapsto\!\!\!\!\twoheadrightarrow} N''$ in less than n $\beta\circ$/es-steps and $N' \preceq_{\text{name}} N''$. By induction hypothesis we can find an N such that $M'' \underset{\text{lenient}}{\longmapsto\!\!\!\!\twoheadrightarrow} N$ and $a \in \text{print}(N)$.

- $\beta\circ$-redex: R is the descendant of a $\beta\circ$-redex R' in M'. We conclude as in the previous case using the induction hypothesis.

□

18. The cyclic call-by-value calculus $\lambda\circ_{\text{value}}$

Both the call-by-name cyclic calculus and the sharing calculus can be used to reason about non-strict functional languages. However, they are unsuitable to reason about strict functional languages such as SML [Har86]. Therefore, we develop another variant of the cyclic calculus, namely a cyclic call-by-value calculus. This calculus is derived from the sharing calculus by restricting the notion of value declaration to the set $VD_V \subset VD$, such that $VD_V \neq \{x_1 = x_2, \cdots, x_n = x_1, VD\}$. The garbage collection and the lambda lift axioms are then restricted to work with VD_V instead of D and VD, respectively.

DEFINITION 18.1. The cyclic call-by-value calculus ($\lambda\circ_{\text{value}}$) has the following axioms.

$(\lambda x.M)N$	$\xrightarrow[\beta\circ]{}$	$\langle M \mid x = N\rangle$	
$\langle C[x] \mid x = V, D\rangle$	$\xrightarrow[\mathrm{es_V}]{}$	$\langle C[V] \mid x = V, D\rangle$	
$\langle M \mid x = C[x_1], x_1 = V, D\rangle$	$\xrightarrow[\mathrm{is_V}]{}$	$\langle M \mid x = C[V], x_1 = V, D\rangle$	
$\langle M \mid D\rangle N$	$\xrightarrow[\mathrm{lift}]{}$	$\langle MN \mid D\rangle$	
$M\langle N \mid D\rangle$	$\xrightarrow[\mathrm{lift}]{}$	$\langle MN \mid D\rangle$	
$\lambda x.\langle M \mid D, VD_V\rangle$	$\xrightarrow[\mathrm{lift}]{}$	$\langle \lambda x.\langle M \mid D\rangle \mid VD_V\rangle$	$D \perp VD_V, VD_V \neq \{\}$ and x not free in VD_V
$\langle\langle M \mid D\rangle \mid D'\rangle$	$\xrightarrow[\mathrm{em}]{}$	$\langle M \mid D, D'\rangle$	
$\langle M \mid x = \langle N \mid D\rangle, D_1\rangle$	$\xrightarrow[\mathrm{im}]{}$	$\langle M \mid x = N, D, D_1\rangle$	
$\langle M \mid D, VD_V\rangle$	$\xrightarrow[\mathrm{gc_V}]{}$	$\langle M \mid D\rangle$	$\{\} \neq VD_V, VD_V \perp \langle M \mid D\rangle$
$\langle M \mid \rangle$	$\xrightarrow[\mathrm{gc}_\square]{}$	M	
M	$\xrightarrow[\mathrm{cp_V}]{}$	N	$\exists\sigma : \mathcal{V} \to \mathcal{V}, N^\sigma \equiv M$ and $\forall x \not\equiv x', \sigma(x) \equiv \sigma(x'), \sigma(x)$ bound to a value in M
$C_{\mathrm{safe}}[M\ N]$	$\xrightarrow[\mathrm{name}]{}$	$C_{\mathrm{safe}}[\langle x \mid x = M\ N\rangle]$	x a new variable

C_{safe} is given by:

$$\begin{array}{rcl} C_{\mathrm{safe}} & ::= & C' \mid C[\lambda x.C'] \mid C[C'\ M] \mid C[M\ C']\ . \\ C' & ::= & \square \mid \langle C' \mid D\rangle \end{array}$$

Table 10. The rewrite relation for the call-by-value calculus $\xrightarrow[\lambda\circ_{\mathrm{value}}]{}$

$\beta\circ$:

$(\lambda x.M)N$	$=$	$\langle M \mid x = N\rangle$	
Substitution:			
$\langle C[x] \mid x = V, D\rangle$	$=$	$\langle C[V] \mid x = V, D\rangle$	
$\langle M \mid x = C[x_1], x_1 = V, D\rangle$	$=$	$\langle M \mid x = C[V], x_1 = V, D\rangle$	
Lift:			
$\langle M \mid D\rangle N$	$=$	$\langle MN \mid D\rangle$	
$M\langle N \mid D\rangle$	$=$	$\langle MN \mid D\rangle$	
$\lambda x.\langle M \mid D, VD_V\rangle$	$=$	$\langle\langle \lambda x.\langle M \mid D\rangle \mid VD_V\rangle$	$D \perp VD_V$ and x not free in VD_V
Merge:			
$\langle M \mid x = \langle N \mid D\rangle, D_1\rangle$	$=$	$\langle M \mid x = N, D, D_1\rangle$	
$\langle\langle M \mid D\rangle \mid D'\rangle$	$=$	$\langle M \mid D, D'\rangle$	
Garbage collection:			
$\langle M \mid D, VD_V\rangle$	$=$	$\langle M \mid D\rangle$	$VD_V \perp \langle M \mid D\rangle$
$\langle M \mid \rangle$	$=$	M	
Copying:			
M	$=$	N	$\exists\sigma : \mathcal{V} \to \mathcal{V}, N^\sigma \equiv M$ and $\forall x \not\equiv x', \sigma(x) \equiv \sigma(x'), \sigma(x)$ bound to a value in M
Naming:			
M	$=$	$\langle x \mid x = M\rangle$	x a new variable

The rewrite relation $\xrightarrow[\lambda\circ_{\mathrm{value}}]{}$ associated to the calculus is given in Table 10.

The rewriting system and the calculus identify the same terms. This is formalized in the following proposition.

PROPOSITION 18.2. *Given $M, N \in \Lambda\circ$. Then $M \xleftrightarrow[\lambda\circ_{\mathrm{value}}]{} N$ iff $\lambda\circ_{\mathrm{value}} \vdash M = N$.*

$\langle M \mid D\rangle N$	$\xrightarrow[\text{lift}]{}$	$\langle MN \mid D\rangle$	
$M\langle N \mid D\rangle$	$\xrightarrow[\text{lift}]{}$	$\langle MN \mid D\rangle$	
$\langle\langle M \mid D\rangle \mid D'\rangle$	$\xrightarrow[\text{em}]{}$	$\langle M \mid D, D'\rangle$	
$\langle M \mid x = \langle N \mid D\rangle, D_1\rangle$	$\xrightarrow[\text{im}]{}$	$\langle M \mid x = N, D, D_1\rangle$	
$\langle M \mid \,\rangle$	$\xrightarrow[\text{gc}_\Box]{}$	M	
$C_{\text{safe}}[M\ N]$	$\xrightarrow[\text{name}]{}$	$C_{\text{safe}}[\langle x \mid x = M\ N\rangle]$	x a new variable

C_{safe} is given by:

$$\begin{array}{rcl} C_{\text{safe}} & ::= & C' \mid C[\lambda x.C'] \mid C[C'\ M] \mid C[M\ C']\ . \\ C' & ::= & \Box \mid \langle C' \mid D\rangle \end{array}$$

Table 11. The kernelizing system $\mathcal{K}$

$(\lambda x.M)N$	$\xrightarrow[\omega\mathcal{K}]{}$	Ω	
$\langle M \mid x = y, D\rangle$	$\xrightarrow[\omega\mathcal{K}]{}$	$\langle M[x := y] \mid D[x := y]\rangle$	$x \not\equiv y, D \neq \{\}$
$\langle M \mid x = y\rangle$	$\xrightarrow[\omega\mathcal{K}]{}$	$M[x := y]$	$x \not\equiv y$
$\langle M \mid x = x, D\rangle$	$\xrightarrow[\omega\mathcal{K}]{}$	Ω	
$\langle M \mid x_1 = x_2\ M_1, \cdots, x_n = x_1\ M_n, D\rangle$	$\xrightarrow[\omega\mathcal{K}]{}$	Ω	
$\langle M \mid x = \lambda y.N, D\rangle$	$\xrightarrow[\omega\mathcal{K}]{}$	$\langle M \mid D\rangle[x := \lambda y.\Omega]$	$D \neq \{\}$
$\langle M \mid x = \lambda y.N\rangle$	$\xrightarrow[\omega\mathcal{K}]{}$	$M[x := \lambda y.\Omega]$	
ΩM	$\xrightarrow[\omega\mathcal{K}]{}$	Ω	
$M\Omega$	$\xrightarrow[\omega\mathcal{K}]{}$	Ω	
$\langle\Omega \mid D\rangle$	$\xrightarrow[\omega\mathcal{K}]{}$	Ω	
$\langle M \mid x = \Omega, D\rangle$	$\xrightarrow[\omega\mathcal{K}]{}$	Ω	

Table 12. The call-by-value ω-rules

Proof. Same as the proof of Proposition 17.2. □

With respect to the sharing of computations, the sharing and call-by-value calculi are the same. This points out that call-by-value, call-by-need and lenient implementations support the same amount of sharing, *i.e.*, the argument of a function is not copied before it is reduced to a value. The difference between the two calculi is that in call-by-value the equations D of a term $\langle M \mid D\rangle$ do not only represent sharing, they also tell us that if one of the terms in D does not produce a value, the complete term should not either. Consider the term $\langle \lambda x.x \mid y = \Omega\rangle$. Its answer according to the sharing calculus is $\lambda x.x$. According to the call-by-value calculus, it must be Ω. This means that in call-by-value we have to be careful in eliminating inaccessible equations. In the acyclic case, a similar point was made by Maraist *et al.* [MOTW95]. In $\langle \lambda x.x \mid y = z\rangle$ and $\langle \lambda x.x \mid y = \lambda z.z\rangle$ it is safe to eliminate the binding for y, and instead in $\langle \lambda x.x \mid y = y\rangle$ and $\langle x \mid y = wz\rangle$ it is not. The proviso on the garbage collection axiom guarantees that these equations are not removed. A similar restriction is imposed on the lambda lift axiom. This is to guarantee that $\langle y \mid y = \lambda z.\langle z \mid x = x\rangle\rangle \neq \langle y \mid y = \lambda z.z, x = x\rangle$, since the first term evaluates to $\lambda z.\Omega$ while the second one evaluates to Ω.

18.1. Basic properties of the cyclic call-by-value lambda calculus

To compute the call-by-value information content we send $\beta\circ$-redexes to Ω. However, unlike the call-by-name calculus we do not send to Ω the substitution redexes and then remove all the inaccessible equations. This would introduce a non-confluence problem. That is, if we let N be $\langle \lambda x.y \mid y = z, z = \lambda w.w\rangle$, then we have the ω-reductions $N \to \langle \lambda x.y \mid y = \Omega\rangle \to \Omega$ and $N \to \langle \lambda x.\Omega \mid z = \lambda w.w\rangle \to$

$\lambda x.\Omega$. Instead, if a variable x is bound to a variable y or to a lambda-abstraction $\lambda y.P$, then each occurrence of x is replaced with y or $\lambda y.\Omega$, respectively, and then the binding gets removed. The information content of N thus becomes $\lambda x.\lambda w.\Omega$. Bindings of the form $x_1 = x_2\ M_1, \cdots, x_n = x_1\ M_n$ are treated in the same way as bindings of the form $x = x$, that is, they cause the entire term to be sent to Ω. In addition, we must be able to handle terms such as $\langle \lambda x.x \mid z = \lambda x.x\rangle\ y$ and $\langle x \mid x = \langle \lambda x.x \mid z = \lambda x.x\rangle\rangle$, which contain an obstructed $\beta\circ$ and value substitution redex, respectively. Thus, we need to add the left lift and internal merge rules. Moreover, we want to equate terms such as $x\ x\ x$ and $\langle y\ x \mid y = x\ x\rangle$ because they represent the same graph. The solution is to first normalize a term with respect to a suitable subset of $\xrightarrow[\lambda\circ_{\text{value}}]{}$. We call this subset the kernelizing system ($\mathcal{K}$). $\mathcal{K}$ is given in Table 11.

Definition 18.3. Given $M \in \Lambda\circ$. We define $\mathcal{K}(M)$ as the normal form of M with respect to the kernelizing system $\mathcal{K}$. We define the set $\Lambda\mathcal{K}$ as $\mathcal{K}(\Lambda\circ)$.

Definition 18.4. Given $M \in \Lambda\mathcal{K}$. The kernelized information content of M is given by the function $\omega_{\mathcal{K}}$, which given M returns the normal form of M with respect to the call-by-value ω-rules (given in Table 12).

From here on we assume that the constant Ω is also a call-by-value term. Ω is not a value; nor is legal to rewrite Ω to $\langle x \mid x = \Omega\rangle$ with the naming rule. We denote the reduction relation induced by the union of the ω-rules and system $\mathcal{K}$ by $\xrightarrow[\omega\mathcal{K},\mathcal{K}]{}$.

Remark 18.5. We can also define $\Lambda\mathcal{K}$ as the subset of $\Lambda\circ_{\text{value}}$ that satisfies the specification:

$$\begin{array}{rcl} M & ::= & V_\Omega \mid \langle V_\Omega \mid D\rangle \\ V_\Omega & ::= & x \mid \lambda x.M \mid \Omega \\ D & ::= & x = V_\Omega \mid x = V_\Omega\ V_\Omega \mid D, D \end{array}$$

Definition 18.6. Given $M, N \in \Lambda\circ$. The call-by-value information content of M is given by the function ω_{value}, which given M returns the normal form of M with respect to $\xrightarrow[\omega\mathcal{K},\mathcal{K}]{}$. We define $M \preceq_{\text{value}} N$ if $\omega_{\text{value}}(M) \leq_\Omega \omega_{\text{value}}(N)$.

Proposition 18.7. *Given $M \in \Lambda\circ$. Then $\omega_{\text{value}}(M) = \omega_{\mathcal{K}}(\mathcal{K}(M))$.*

Proof. Given a term M we are free to reduce to $\mathcal{K}(M)$ first in computing $\omega_{\text{value}}(M)$. When we reduce $\mathcal{K}(M)$ to $\omega_{\text{value}}(M)$ afterwards we only use ω-rules. Hence the equality. □

We want to consider the cyclic call-by-value calculus as an abstract rewriting system with ordered information content. To that end we first need to show monotonicity:

Lemma 18.8.

(i) *Given $(\Lambda\circ, \leq_\Omega, \preceq_{\text{value}})$. $\leq_\Omega$ is monotonic with respect to $\preceq_{\text{value}}$.*

(ii) *Given $(\Lambda\circ, \xrightarrow[\lambda\circ_{\text{value}}]{}, \preceq_{\text{value}})$. $\xrightarrow[\lambda\circ_{\text{value}}]{}$ is monotonic with respect to $\preceq_{\text{value}}$.*

Proof.

(i) A simple check yields that $\xrightarrow[\omega\mathcal{K},\mathcal{K}]{}$ locally commutes with $\xrightarrow[\Omega]{}$. From this and the fact that

$\xrightarrow[\omega\mathcal{K},\mathcal{K},\Omega]{}$ is terminating we get by Newman's lemma that:

$$\begin{tikzcd}
N \arrow[r, twoheadrightarrow, "\omega\mathcal{K},\mathcal{K}"] \arrow[d, twoheadrightarrow, "\Omega"'] & \omega_{\text{value}}(N) \arrow[d, dashed, twoheadrightarrow, "\Omega"] \\
M \arrow[r, dashed, twoheadrightarrow, "\omega\mathcal{K},\mathcal{K}"'] & M' \arrow[d, dashed, twoheadrightarrow, "\omega\mathcal{K},\mathcal{K}"'] \\
 & \omega_{\text{value}}(M)
\end{tikzcd}$$

The $\omega\mathcal{K},\mathcal{K}$-reduction from M' to $\omega_{\text{value}}(M)$ is also an Ω-reduction: Because $\omega_{\text{value}}(N)$ contains no more equations of the form $x = V$ and none of the other steps can create such an equation we never have to use the $\omega\mathcal{K}$-rules for a value substitution. Because $\omega_{\text{value}}(N)$ is a $\mathcal{K}$-normal form and because an Ω-step cannot create a $\mathcal{K}$-redex, we do not need any $\mathcal{K}$-steps.

(ii) If $C[(\lambda x.M)\ N] \to C[\langle M \mid x = N\rangle]$ then $\omega_{\text{value}}(C[(\lambda x.M)\ N]) = \omega_{\text{value}}(C[\Omega])$, which is less then $\omega_{\text{value}}(C[\langle M \mid x = N\rangle])$ because $C[\Omega] \leq_\Omega C[\langle M \mid x = N\rangle]$ and the first part. If $C[x] \to C[\lambda y.M]$ then $\omega_{\text{value}}(C[x]) = \omega_{\text{value}}(C[\lambda y.\Omega])$, which using the same argument as before is less than $\omega_{\text{value}}(C[\lambda y.M])$. If $M \equiv C[x] \to C[y] \equiv N$, or $M \xrightarrow[\mathcal{K}]{} N$ then obviously $M \simeq_{\text{value}} N$. If $M \to N$ by application of value lift or value garbage collection then by rewriting to normal form with respect to the value binding rules the bindings in the VD_V involved one can show that $M \simeq_{\text{value}} N$. Finally, if $M \xrightarrow[\text{cpv}]{} N$ then by proving commutativity of the unification rule and the value binding rules and the black hole rule for $x = x$ we get $M \simeq_{\text{value}} N$.

□

We now have that the function ω_{value} defines a proper notion of information content for $\lambda\circ_{\text{value}}$:

PROPOSITION 18.9. $\langle(\Lambda\circ, \xrightarrow[\lambda\circ_{\text{value}}]{}, \preceq_{\text{value}}), \omega_{\text{value}}, (\omega_{\text{value}}(\Lambda\circ), \leq_\Omega)\rangle$ *is an ARS with ordered information content.*

PROOF. Trivial. □

Next, we want to show that $\xrightarrow[\lambda\circ_{\text{value}}]{}$ is confluent up to $\preceq_{\text{value}}$. We do not prove this directly, instead we introduce a kernelized reduction ($\xrightarrow[\lambda\mathcal{K}]{}$) and prove confluence up to information content for that reduction relation.

DEFINITION 18.10. Given $M, N \in \Lambda\mathcal{K}$. We say that M rewrites to N in the kernelized system $\lambda\mathcal{K}$ ($M \xrightarrow[\lambda\mathcal{K}]{} N$) if $M \xrightarrow[\lambda\circ_{\text{value}}]{} P$ and $\mathcal{K}(P) \equiv N$.

If the $\lambda\circ_{\text{value}}$-step taken was a $\beta\circ$-step then the corresponding $\lambda\mathcal{K}$-step is a $\beta\mathcal{K}$-step. Similarly a lambda lift step is denoted lift$\mathcal{K}$. The labels for copy and substitution don't change because those steps preserve kernelized terms.

Because we want to derive a property of call-by-value reduction from kernelized reduction we need to relate reductions in the two systems.

PROPOSITION 18.11.

(i) *We have that* $\xrightarrow[\lambda\mathcal{K}]{} \subseteq \xrightarrow[\lambda\circ_{\mathrm{value}}]{}\!\!\!\!\twoheadrightarrow$.

(ii) *Given* $M, N \in \Lambda\circ$. *If* $M \xrightarrow[\lambda\circ_{\mathrm{value}}]{}\!\!\!\!\twoheadrightarrow N$ *then* $\mathcal{K}(M) \xrightarrow[\lambda\mathcal{K}]{}\!\!\!\!\twoheadrightarrow \mathcal{K}(N)$.

PROOF.

(i) Trivial.

(ii) In this proof we denote rewriting to $\mathcal{K}$-normal form by $\xrightarrow[\mathcal{K}^{\mathrm{nf}}]{}$. The result follows from the definition of $\xrightarrow[\lambda\mathcal{K}]{}$ and the diagrams below.

$$
\begin{tikzcd}
\cdot \arrow[r, "\mathrm{gc_V,cp_V,is_V,es_V}"] \arrow[d, "\mathcal{K}^{\mathrm{nf}}"'] & \cdot \arrow[d, dashed, "\mathcal{K}^{\mathrm{nf}}"] \\
\cdot \arrow[r, dashed, "\mathrm{gc_V,cp_V,is_V,es_V}"'] & \cdot
\end{tikzcd}
\qquad
\begin{tikzcd}
\cdot \arrow[rr, "\beta\circ\mathrm{,lift}\mathcal{K}"] \arrow[d, "\mathcal{K}^{\mathrm{nf}}"'] & & \cdot \arrow[d, dashed, "\mathcal{K}^{\mathrm{nf}}"] \\
\cdot \arrow[r, dashed, "\beta\circ\mathrm{,lift}\mathcal{K}"'] & \cdot \arrow[r, dashed, "\mathcal{K}^{\mathrm{nf}}"'] & \cdot
\end{tikzcd}
\qquad
\begin{tikzcd}
\cdot \arrow[r, "\mathrm{others}"] \arrow[d, "\mathcal{K}^{\mathrm{nf}}"'] & \cdot \arrow[d, dashed, "\mathcal{K}^{\mathrm{nf}}"] \\
\cdot \arrow[r, phantom, "\equiv"] & \cdot
\end{tikzcd}
$$

□

From this relation we immediately obtain that the kernelized calculus is also an ARS with ordered information content, when using the same notion of information content.

COROLLARY 18.12. $\langle(\Lambda\mathcal{K}, \xrightarrow[\lambda\mathcal{K}]{}, \preceq_{\mathrm{value}}), \omega_{\mathcal{K}}, (\omega_{\mathcal{K}}(\Lambda\mathcal{K}), \leq_{\Omega})\rangle$ *is an ARS with ordered information content.*

PROOF. From the previous proposition, Proposition 18.9 and Proposition 18.7. □

The proof of confluence up to information content for the kernelized system again uses the technique of Section 10.2. The notion of standard reduction for $\lambda\mathcal{K}$ differs from the standard reduction for call-by-name because we now have to develop the environment too. The idea of standard reduction is to do redexes that are needed and to avoid non-confluence (of the standard reduction) by not reducing needed redexes that can still be copied by a reduction step. This means that in

$$\langle x\ y \mid x = \lambda z.\langle y \mid y = (\lambda x.x)z\rangle\rangle$$

although both the substitution redex and the $\beta\circ$-redex are needed, the $\beta\circ$-redex cannot be standard because the substitution redex copies (the result of) the $\beta\circ$-redex. This principle rules out any redex inside the V of an equation $x = V$, because such a redex is not needed or it(s result) is copied later on. However, not every redex inside a value may be ruled out. In

$$\lambda z.\langle x \mid y = (\lambda x.x)z\rangle$$

the $\beta\circ$-redex is needed and will not be copied so it must be standard. Formally we have:

DEFINITION 18.13. Given $M \in \Lambda\mathcal{K}$. A reduction step $M \xrightarrow[\lambda\mathcal{K}]{} N$ is standard (written as $M \mapsto_{\lambda\mathcal{K}} N$) if

$$M \equiv E_{\mathcal{K}}[R] \xrightarrow[\lambda\mathcal{K}]{} E_{\mathcal{K}}[R'] \equiv N \ ,$$

where R and R' stand for a $\beta\circ$ or value substitution redex and its contractum, and $E_{\mathcal{K}}$ is defined as follows:

$$E_{\mathcal{K}} \quad ::= \quad \square \mid \langle E_{\mathcal{K}} \mid D\rangle \mid \lambda x.E_{\mathcal{K}} \mid \langle V \mid x = \square_{\beta}, D\rangle \mid \langle V_1 \mid x = \square\ V_2, D\rangle \mid \langle V \mid x = y\ E_{\mathcal{K}}, D\rangle$$

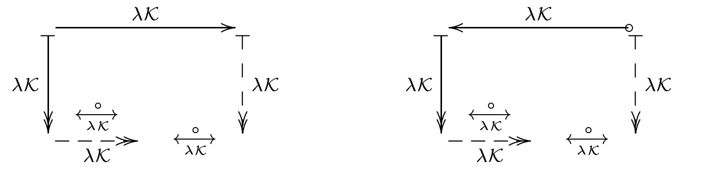
where:

- $\square_\beta$ means that a substitution redex is not allowed in that context
- The occurrence of y in the clause $\langle V \mid x = y\ E_{\mathcal{K}}, D\rangle$ in the main term M leads to a variable that is either free or bound by a lambda. A variable x leads to another variable y if $x \equiv y$ or if $x = z\ P$ and z leads to y or if $x = z$ and z leads to y.

LEMMA 18.14. *Given terms $M, N \in \Lambda\mathcal{K}$. If $M \xrightarrow[\lambda\mathcal{K}]{\circ} N$ then $M \simeq_{\text{value}} N$.*

PROOF. If the redex is not a $\beta\circ$ or substitution redex then the claim is obvious. If we have a $\beta\circ$ or substitution redex in a non-standard context then the redex and its contractum are deleted by the $\omega\mathcal{K}$-rules, no matter what the redex or the contractum is. $\square$

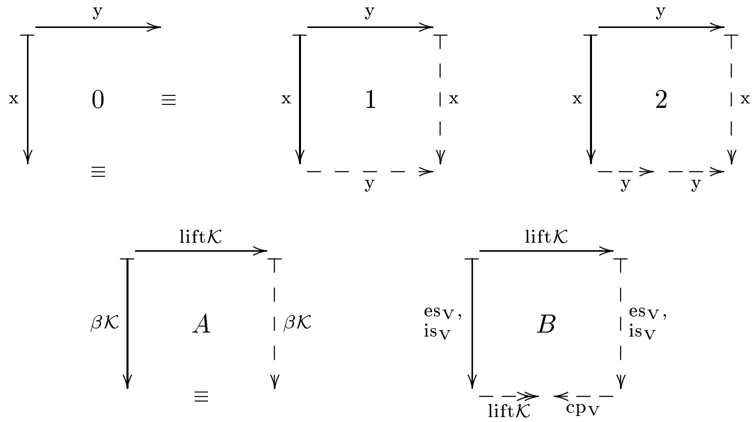
LEMMA 18.15. (REDUCTION LEMMA) *We have the following two diagrams:*



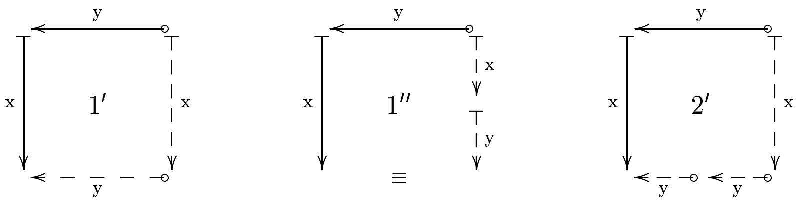
Moreover, in the left diagram the number of steps in the right standard reduction is not greater than the number of steps in the left standard reduction and smaller if one of the standard redexes is a descendant of the top step.

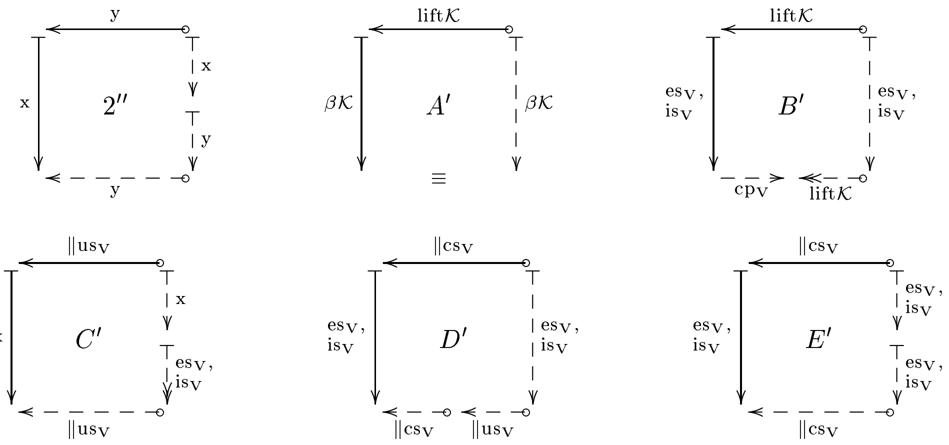
PROOF. We first investigate the local commutativity. All the possible diagrams are given by cases on the top step.

- If the top step is any step from left to right.



- If the top step is a non-standard step from right to left then, as before, we have to turn to complete developments of unnested and cyclic substitutions to obtain decreasing diagrams. That is, we introduce unnested substitution of values (us_{V}) and cyclic substitution of values (cs_{V}) as for the call-by-name case and by use disjoint complete developments of these redexes ($\|\mathrm{us}_{\mathrm{V}}$ and $\|\mathrm{cs}_{\mathrm{V}}$)





More precisely, we have:

$x \backslash y$	$\xrightarrow[\beta\mathcal{K}]{}$	$\xrightarrow[es_V, is_V]{}$	$\xrightarrow[\mathrm{gc_V}]{}$	$\xrightarrow[\mathrm{lift}\mathcal{K}]{}$	$\xrightarrow[\mathrm{cp_V}]{}$	$\circ\!\xrightarrow[\beta\mathcal{K}]{}$	$\circ\!\xrightarrow[\Vert\mathrm{cs_V}]{}$	$\circ\!\xrightarrow[\Vert\mathrm{us_V}]{}$	$\circ\!\xrightarrow[\mathrm{gc_V}]{}$	$\circ\!\xrightarrow[\mathrm{lift}\mathcal{K}]{}$	$\circ\!\xrightarrow[\mathrm{cp_V}]{}$
$\mapsto_{\beta\mathcal{K}}$	0,1	1	1	1,A	1	1',1"	1'	C'	1'	1',A'	1'
$\mapsto_{\mathrm{es_V,is_V}}$	1,2	0,1,2	1,2	1,2,B	1	1',2',2"	1',D',E'	C'	1',2'	1',2',B'	1'

When we order the reduction steps by $\Vert\mathrm{cs_V} > \Vert\mathrm{us_V} > \xmapsto[\mathrm{es_V,is_V}]{} > \xrightarrow[\mathrm{es_V,is_V}]{} > \beta\mathcal{K}, \mathrm{gc_V}, \mathrm{lift}\mathcal{K}, \mathrm{cp_V}$ we obtain that all diagrams are decreasing. This means that we can complete the tiling process. Because a non-standard step cannot increase information content (Lemma 18.14) and because reduction is monotonic (Lemma 18.8(ii)) we have that the two diagrams we need to prove hold. It is obvious from the diagrams that the length of the reduction does not increase for the left diagram. By tracing descendants in the diagrams we can conclude that if a descendant of the top step is contracted then the length decreases. □

LEMMA 18.16. *Given* $\langle(\Lambda\mathcal{K}, \xrightarrow[\lambda\mathcal{K}]{}, \preceq_{\mathrm{value}}), \omega_{\mathcal{K}}, (\omega_{\mathcal{K}}(\Lambda\mathcal{K}), \leq_\Omega)\rangle$. *Then*

(i) $\xmapsto[\lambda\mathcal{K}]{}$ *is complete for* $\xrightarrow[\lambda\mathcal{K}]{}$ *up to* $\preceq_{\mathrm{value}}$.

(ii) $\xmapsto[\lambda\mathcal{K}]{}$ *commutes with* $\xrightarrow[\lambda\mathcal{K}]{}$ *up to* $\preceq_{\mathrm{value}}$.

PROOF. From Lemma 18.15 and Proposition 10.10. □

Having proven these properties of the kernelized reduction system we can now go back to call-by-value reduction to prove that it is confluent up to information content.

THEOREM 18.17. *Given* $\langle(\Lambda\circ, \xrightarrow[\lambda\circ_{\mathrm{value}}]{}, \preceq_{\mathrm{value}}), \omega_{\mathrm{value}}, (\omega_{\mathrm{value}}(\Lambda\circ), \leq_\Omega)\rangle$. $\xrightarrow[\lambda\circ_{\mathrm{value}}]{}$ *is confluent up to* $\preceq_{\mathrm{value}}$.

PROOF. From Lemmas 18.16 and 10.6 and Proposition 18.11(ii). □

18.2. Semantics of the cyclic call-by-value lambda calculus

We have shown that $\xrightarrow[\lambda\circ_{\mathrm{value}}]{}$ is confluent up to information content. Hence, by Lemma 10.18 the call-by-value infinite normal form of any cyclic term M, written as $\mathrm{Inf}_{\mathrm{value}}(M)$, is well defined and unique. To prove that the infinite normal form is compositional we return to the kernelized reduction system. We denote the infinite normal form with respect to the kernelized reduction system by $\mathrm{Inf}_{\mathcal{K}}$. We first prove the following relation between $\mathrm{Inf}_{\mathrm{value}}$ and $\mathrm{Inf}_{\mathcal{K}}$:

Lemma 18.18. *Given $M \in \Lambda\circ$. Then* $\mathrm{Inf}_{\text{value}}(M) = \mathrm{Inf}_{\mathcal{K}}(\mathcal{K}(M))$.

Proof. Simple corollary of Proposition 18.11. □

For the call-by-name and call-by-value calculi, given a context C and term M, $C[M]$ is a legal term. For the kernelized calculus this does not always hold, *e.g.,* $\langle x \mid x = \square\rangle[\langle x \mid x = x\rangle] \equiv \langle x \mid x = \langle x \mid x = x\rangle\rangle$. Thus, instead of considering all possible contexts we only consider a subset of them. We can then follow the usual pattern to prove that $\mathrm{Inf}_{\mathcal{K}}$ is continuous in the limited set of contexts and use that fact to show that $\mathrm{Inf}_{\text{value}}$ is compositional.

Definition 18.19.
- A lambda context C_λ is any call-by-value context of the form $C[\lambda x.\square]$.
- For the $\mathcal{K}$-rules $\square$ is considered to be a value.
- A kernelized context $C_{\mathcal{K}}$ is a lambda context in $\mathcal{K}$-normal form.

These contexts have the following property:

Proposition 18.20. *Given a lambda context C_λ and $M \in \Lambda\circ_{\text{value}}$. Then*

$$\mathcal{K}(C_\lambda)[\mathcal{K}(M)] = \mathcal{K}(C_\lambda[M]) \ .$$

Proof. Trivial. □

Lemma 18.21. *Given terms $M, N \in \Lambda\mathcal{K}$. If $M \leq_\Omega N$ then* $\mathrm{Inf}_{\mathcal{K}}(M) \subseteq \mathrm{Inf}_{\mathcal{K}}(N)$.

Proof. We have that

$$\begin{tikzcd}
{} \arrow[r, "{\beta\mathcal{K},\mathrm{es}_{\mathrm{V}},\mathrm{is}_{\mathrm{V}}}"] & {} \\
{} \arrow[u, "\Omega"] \arrow[r, dashed, "{\beta\mathcal{K},\mathrm{es}_{\mathrm{V}},\mathrm{is}_{\mathrm{V}}}"'] & {} \arrow[u, dashed, two heads, "\Omega"']
\end{tikzcd}$$

From this fact, Lemma 10.19 Lemma 18.16(i) and Lemma 18.8(i) the result follows. □

Proposition 18.22. *Given a kernelized context $C_{\mathcal{K}}$ and a term $M \in \Lambda\mathcal{K}$.*

(i) *If $C_{\mathcal{K}}[M] \underset{\lambda\mathcal{K}}{\longmapsto\!\!\!\!\twoheadrightarrow} N$ then there exists M_1 such that $M \underset{\lambda\mathcal{K}}{\twoheadrightarrow} M_1$, $C_{\mathcal{K}}[M_1] \overset{\lambda\mathcal{K}}{\underset{M_1}{\not\longmapsto\!\!\!\!\twoheadrightarrow}} N_1$ without reducing any redex inside M_1 (written as $\underset{M_1}{\not\twoheadrightarrow}$) and $N \preceq_{\text{value}} N_1$.*

(ii) *If $C_{\mathcal{K}}[M] \overset{\lambda\mathcal{K}}{\underset{M}{\not\longmapsto\!\!\!\!\twoheadrightarrow}} N$ then $C_{\mathcal{K}}[\omega_{\mathcal{K}}(M)] \underset{\lambda\mathcal{K}}{\longmapsto\!\!\!\!\twoheadrightarrow} N'$ with $N \simeq_{\text{value}} N'$.*

(iii) $\mathrm{Inf}_{\mathcal{K}}(C_{\mathcal{K}}[\omega_{\mathcal{K}}(M)]) \subseteq \mathrm{Inf}_{\mathcal{K}}(C_{\mathcal{K}}[M])$.

Proof.

(i) We do this by induction on the number of steps in the standard reduction. Assume we can find M_1 and N_1 if the length of the given standard reduction is less than n and assume we are given a reduction of length n.

If $C_{\mathcal{K}}[M] \overset{\lambda\mathcal{K}}{\underset{M}{\not\longmapsto\!\!\!\!\twoheadrightarrow}} N$ we can take $M_1 \equiv M$ and $N_1 \equiv N$. Otherwise there is a first standard step that does a descendant of a redex in M. If we do this redex in M and obtain M_2 then we get the following diagram:

$$\begin{tikzcd}
C_{\mathcal{K}}[M] \arrow[r, maps to, two heads, "\lambda\mathcal{K}"] \arrow[d, "\lambda\mathcal{K}"'] \arrow[dr, phantom, "\text{Lemma 18.15}"] & N \arrow[d, phantom, "\preceq_{\text{value}}"] \\
C_{\mathcal{K}}[M_2] \arrow[r, dashed, maps to, two heads, "\lambda\mathcal{K}"'] \arrow[d, dashed, two heads, "\lambda\mathcal{K}"'] \arrow[dr, phantom, "\text{I.H.}"] & N_2 \arrow[d, phantom, "\preceq_{\text{value}}"] \\
C_{\mathcal{K}}[M_1] \arrow[r, dashed, maps to, two heads, "\lambda\mathcal{K}", "\not\ M_1"'] & N_1
\end{tikzcd}$$

(ii) It is easier for this proof to have the following rules for black holes:

$$\begin{array}{rcl} \langle M \mid x = x, D\rangle & \to & \langle M \mid x = \Omega, D\rangle \\ \langle M \mid x_1 = x_2\ M_1, \cdots, x_n = x_1\ M_n, D\rangle & \to & \langle M \mid x_1 = \Omega, \cdots, x_n = \Omega, D\rangle \end{array}$$

These rules are equivalent in the presence of the propagation rules, but make the proof work smoother by keeping changes local. One of the properties of the $\omega\mathcal{K}$ system is that we can delay the propagation rules until and first rewrite to normal form with respect to the other rules. Because of the fact that the given standard reduction does not do any redex that is rewritten by $\omega\mathcal{K}$ and cannot touch the black holes of M we have the following diagram:

$$\begin{tikzcd} \cdot \arrow[r, maps to, "\lambda\mathcal{K}"] \arrow[d, "\omega\mathcal{K}"'] & \cdot \arrow[d, dashed, two heads, "\omega\mathcal{K}"] \\ \cdot \arrow[r, dashed, maps to, "\lambda\mathcal{K}"'] & \cdot \end{tikzcd}$$

When we consider the $\omega\mathcal{K}$-rules dealing with the propagation of Ω we have the following diagrams:

$$\begin{tikzcd} \cdot \arrow[r, maps to, "\lambda\mathcal{K}"] \arrow[d, "\omega\mathcal{K}"'] & \cdot \arrow[d, dashed, two heads, "\omega\mathcal{K}"] \\ \cdot \arrow[r, dashed, maps to, "\lambda\mathcal{K}"'] & \cdot \end{tikzcd} \qquad \begin{tikzcd} \cdot \arrow[r, maps to, "\lambda\mathcal{K}"] \arrow[d, "\omega\mathcal{K}"'] & \cdot \arrow[d, dashed, "\omega\mathcal{K}"] \\ \cdot \arrow[r, phantom, "\equiv"] & \cdot \end{tikzcd} \qquad \begin{tikzcd} \cdot \arrow[r, maps to, "\lambda\mathcal{K}"] & \cdot \\ \cdot \arrow[u, "\omega\mathcal{K}"] \arrow[r, dashed, maps to, "\lambda\mathcal{K}"'] & \cdot \arrow[u, dashed, two heads, "\omega\mathcal{K}"'] \end{tikzcd} \qquad \begin{tikzcd} \cdot \arrow[r, maps to, "\lambda\mathcal{K}"] \arrow[dd, "\omega\mathcal{K}"'] & \cdot \arrow[d, dashed, "\omega\mathcal{K}"] \\ & \cdot \\ \cdot \arrow[r, phantom, "\equiv"] & \cdot \arrow[u, dashed, "\omega\mathcal{K}"'] \end{tikzcd}$$

(iii) If we view the lambda-substitution ω-rule as a sequence

$$\langle M \mid x = \lambda y.N, D\rangle \xrightarrow[\omega]{} \langle M \mid x = \lambda y.\Omega, D\rangle \xrightarrow[\mathrm{es_V.is_v,gc}]{} \!\!\!\!\rightarrow \langle M[x := \lambda y.\Omega] \mid D[x := \lambda y.\Omega]\rangle$$

then the proof of Proposition 13.4(iii) is applicable.

□

LEMMA 18.23. *Given $M \in \Lambda\mathcal{K}$ and a kernelized context $C_{\mathcal{K}}$. Then*

$$\mathrm{Inf}_{\mathcal{K}}(C_{\mathcal{K}}[M]) = \bigcup\{\mathrm{Inf}_{\mathcal{K}}(C_{\mathcal{K}}[a]) \mid a \in \mathrm{Inf}_{\mathcal{K}}(M)\}\ .$$

PROOF. Follows from Lemmas 18.22 and 18.21. □

We are now able to prove that $\mathrm{Inf}_{\mathrm{value}}$ defines a model:

THEOREM 18.24. *Given $M, N \in \Lambda\circ$ and a context C. Then*

$$\mathrm{Inf}_{\mathrm{value}}(M) = \mathrm{Inf}_{\mathrm{value}}(N) \Rightarrow \mathrm{Inf}_{\mathrm{value}}(C[M]) = \mathrm{Inf}_{\mathrm{value}}(C[N])\ .$$

PROOF. We distinguish cases for C being a lambda context or not.

- If C is a lambda context we have:

$$\begin{array}{rcll} \mathrm{Inf}_{\mathrm{value}}(C[M]) & = & \mathrm{Inf}_{\mathcal{K}}(\mathcal{K}(C[M])) & \text{Lemma 18.18} \\ & = & \mathrm{Inf}_{\mathcal{K}}(\mathcal{K}(C)[\mathcal{K}(M)]) & \text{Lemma 18.20} \\ & = & \bigcup\{\mathrm{Inf}_{\mathcal{K}}(\mathcal{K}(C)[a]) \mid a \in \mathrm{Inf}_{\mathcal{K}}(\mathcal{K}(M))\} & \text{Lemma 18.23} \\ & = & \bigcup\{\mathrm{Inf}_{\mathcal{K}}(\mathcal{K}(C)[a]) \mid a \in \mathrm{Inf}_{\mathrm{value}}(M)\} & \text{Lemma 18.18} \end{array}$$

Similarly we have:

$$\mathrm{Inf}_{\mathrm{value}}(C[N]) = \bigcup\{\mathrm{Inf}_{\mathcal{K}}(\mathcal{K}(C)[a]) \mid a \in \mathrm{Inf}_{\mathrm{value}}(N)\}\ .$$

The result then follows easily.

- If C is not a lambda context then let x and y be new variables. define $C_\lambda = C[(\lambda x.\Box)\ (\lambda y.y)]$. By the previous case we then have that $\mathrm{Inf}_{\mathrm{value}}(C_\lambda[M]) = \mathrm{Inf}_{\mathrm{value}}(C_\lambda[N])$. Because x and y are new we have for $P = M, N$ that $C_\lambda[P] \xrightarrow[\mathrm{value}]{} C[P]$. Hence, by uniqueness of the infinite normal form, we conclude that

$$\mathrm{Inf}_{\mathrm{value}}(C[M]) = \mathrm{Inf}_{\mathrm{value}}(C[N]) \ .$$

$\Box$

18.3. The cyclic call-by-value lambda calculus and the cyclic sharing calculus

We relate the call-by-value infinite normal form to the sharing infinite normal form. To do this we must print the call-by-value observations because they contain sharing while the observations in the sharing calculus are trees. On call-by-value observations the print function has the following property. Given $a \in \omega_{\mathrm{value}}(\Lambda\circ)$. Then

$$\bigsqcup \mathrm{print}(a) = a^\infty \ .$$

This property ensures us that when we print a call-by-value observation we do no loose any information. Thus the following comparison is meaningful:

THEOREM 18.25. *Given $M \in \Lambda\circ$. Then*

$$\bigcup\{\mathrm{print}(a) \mid a \in \mathrm{Inf}_{\mathrm{value}}(M)\} \subseteq \mathrm{Inf}_{\mathrm{share}}(M) \ .$$

PROOF. Follows from the fact that for every $P \in \Lambda\circ$ we have that $\mathrm{print}(\omega_{\mathrm{value}}(P)) \subseteq \mathrm{print}(P)$ and the monotonicity of print. $\Box$

EXAMPLE 18.26. The inclusion in the theorem above is sometimes strict. For $M \equiv \lambda x.x$ we have equality but for for $M \equiv \langle \lambda x.x \mid y = y\rangle$ we have strict inclusion.

Next, we show that the call-by-value and sharing calculus have the same evaluation calculus. First we define the call-by-value infinite normal form generated by the evaluation calculus of Table 7 as:

$$\mathrm{Inf}^{\mathrm{value}}_{\mathrm{eval}}(M) = \downarrow \{\omega_{\mathrm{value}}(N) \mid M \xrightarrow[\mathrm{eval}_V]{} N\} \ .$$

PROPOSITION 18.27. *Given $M \in \Lambda\circ$. Then*

$$\mathrm{Inf}_{\mathrm{value}}(M) = \mathrm{Inf}^{\mathrm{value}}_{\mathrm{eval}}(M)$$

PROOF.

"$\subseteq$" Given $a \in \mathrm{Inf}_{\mathrm{value}}(M)$. By Lemma 18.16(i) we have that $\mathcal{K}(M) \xrightarrow[\lambda\mathcal{K}]{} N$ with $a \leq_\Omega \omega_{\mathrm{value}}(N)$. This means that $M \xrightarrow[\beta\circ,\mathrm{es}_V,\mathrm{is}_V]{\overleftrightarrow{\mathcal{K}}} N$. (The superscript $\overleftrightarrow{\mathcal{K}}$ denotes reduction modulo $\overleftrightarrow{\mathcal{K}}$.) The result then follows from the fact that $P \overleftrightarrow{\mathcal{K}} Q \Rightarrow P \simeq_{\mathrm{value}} Q$ and the claim that

$$M \xrightarrow[\beta\circ,\mathrm{es}_V,\mathrm{is}_V]{\overleftrightarrow{\mathcal{K}}} N \Rightarrow M \xrightarrow[\beta\circ,\mathrm{es}_V,\mathrm{is}_V,\mathrm{ll},\mathrm{im}]{} \overleftrightarrow{\mathcal{K}} N \ .$$

The claim can be proven by proving

$$\begin{tikzcd} \cdot \arrow[r, "\mathcal{K}"] \arrow[d, "\beta*,\mathrm{es}*,\mathrm{is}*"'] & \cdot \arrow[d, "\beta*,\mathrm{es}*,\mathrm{is}*"] \\ \cdot \arrow[r, dashed, leftrightarrow, "\mathcal{K}"'] & \cdot \end{tikzcd} \qquad \begin{tikzcd} \cdot \arrow[d, "\beta*,\mathrm{es}*,\mathrm{is}*"'] & \cdot \arrow[l, "\mathcal{K}"'] \arrow[d, "\beta*,\mathrm{es}*,\mathrm{is}*"] \\ \cdot \arrow[r, dashed, leftrightarrow, "\mathcal{K}"'] & \cdot \end{tikzcd}$$

$$\begin{array}{rcl} M & ::= & V \mid NV \\ V & ::= & x \mid \lambda x.M \\ NV & ::= & \mathtt{let}\ x = M\ \mathtt{in}\ N \mid M\ N \end{array}$$

$$\begin{array}{rlrcl} \beta_V & : & (\lambda x.M)V & = & M[x := V] \\ \eta_V & : & \lambda x.V\ x & = & V \text{ if } x \not\in \mathrm{FV}(V) \\ id & : & \mathtt{let}\ x = M\ \mathtt{in}\ x & = & M \\ comp & : & \mathtt{let}\ x = \mathtt{let}\ y = M\ \mathtt{in}\ N\ \mathtt{in}\ P & = & \mathtt{let}\ y = M\ \mathtt{in}\ \mathtt{let}\ x = N\ \mathtt{in}\ P \\ let_V & : & \mathtt{let}\ x = V\ \mathtt{in}\ M & = & M[x := V] \\ let_1 & : & NV\ M & = & \mathtt{let}\ x = NV\ \mathtt{in}\ x\ M \\ let_2 & : & V\ NV & = & \mathtt{let}\ x = NV\ \mathtt{in}\ V\ x \\ let_C & : & \mathtt{let}\ x = M\ \mathtt{in}\ \mathtt{let}\ y = N\ \mathtt{in}\ P & = & \mathtt{let}\ y = N\ \mathtt{in}\ \mathtt{let}\ x = M\ \mathtt{in}\ P \\ & & & & \text{if } x \not\in \mathrm{FV}(M,N)\ y \not\in \mathrm{FV}(M,N) \end{array}$$

Table 13. Commutative Moggi's computational lambda calculus: λ_c

where

$$\begin{array}{rcl} \langle \cdots \langle (\lambda x.M) \mid D_1 \rangle \cdots D_n \rangle\ N & \xrightarrow[\beta *]{} & \langle \cdots \langle \langle M \mid x = N \rangle \mid D_1 \rangle \cdots D_n \rangle \\ \langle C[x] \mid x = \langle \cdots \langle V \mid D_1 \rangle \cdots D_n \rangle, D \rangle & \xrightarrow[\mathrm{es}*]{} & \langle C[x] \mid x = V, D_1, \cdots, D_n, D \rangle \\ \langle M \mid y = C[x], x = \langle \cdots \langle V \mid D_1 \rangle \cdots D_n \rangle, D \rangle & \xrightarrow[\mathrm{is}*]{} & \langle M \mid y = C[x], x = V, D_1, \cdots, D_n, D \rangle \end{array}$$

"$\supseteq$" Obvious.

□

18.4. The cyclic call-by-value calculus and Moggi's computational lambda calculus

We use the infinite call-by-value normal form to relate our calculus to the commutative version of the computational lambda calculus of Moggi (λ_c) [Mog88], given in Table 13. Since Moggi's calculus is acyclic, we first need to relate a cyclic term to the acyclic terms approximating it, as we did for the call-by-name calculus. However, this relation only works for a subset of the set of terms. For example, the answer of $\langle x \mid x = yx \rangle$ is the term itself. Instead, the answer of any of its approximations is Ω. If we consider only internal merge normal forms then the restriction is that if several declarations are mutually recursive then all those declarations must only involve values. For example, $\langle y \mid y = x, x = \lambda z.y \rangle$ and $\langle y \mid y = \lambda z.x, x = \lambda z.y \rangle$ are good terms, but $\langle x \mid x = \lambda y.z, z = x\ x \rangle$ is not. An arbitrary term is good if its internal merge normal form is good. We denote the set of good terms by $\Lambda\circ_{\mathrm{value}}$. The set of good terms is closed under rewriting, but a bad term may be rewritten into a good term. For example, $\langle x \mid x = \langle y \mid z = x\ x \rangle\ y \to \langle x \mid x = \langle y\ y \mid z = x\ x \rangle$ and $\langle x \mid x = (\lambda y.\langle y \mid z = x \rangle) w \rangle \to \langle x \mid x = \langle \langle y \mid z = x \rangle \mid y = w \rangle \rangle$.

To define call-by-value expansions, we introduce the notation $M \xrightarrow[\mathrm{GK(as)}]{}^{n} N$ which denotes n-steps of the Gross-Knuth strategy applied to acyclic value substitution redexes occurring in M. The notion of acyclic substitution redex is taken from [AK]. An acyclic value substitution redex is any value substitution redex that is not of the form $\langle M \mid x = C[y], y = V, D \rangle$, where x and y are mutually recursive. In $\langle \underline{x} \mid x = \lambda z.\underline{y}, y = \lambda z.\underline{w}, w = \lambda z.y \rangle$, the underlined x and y are acyclic value substitution redexes, and the underlined w is not. Since a value substitution redex can be obstructed by an environment, we first compute the internal merge normal form of a term M denoted by $\mathrm{nf}_{\mathrm{im}}(M)$.

Definition 18.28. Given $M \in \Lambda\circ_{\text{value}}$. The n^{th} call-by-value expansion of M, written as M_V^n, is the term $\text{strip}_V(M_n^V)$ such that $\text{nf}_{\text{im}}(M) \xrightarrow[\text{GK(as)}]{}^n M_n^V$ and $\text{strip}_V(N)$ is the normal form of N with respect to the rule:

$$\begin{array}{rcll} \langle M \mid x_1 = x_2, \cdots, x_n = x_1, D\rangle & \to & \langle M \mid x_1 = \Omega, \cdots, x_n = \Omega, D\rangle & \\ \langle M \mid x = \lambda y.N, D\rangle & \to & \langle M \mid x = \lambda y.\Omega, D\rangle & N \not\equiv \Omega \end{array}$$

For instance, given the term $M \equiv \langle x\ x \mid x = y\ y, y = \langle \lambda x.y \mid z = \lambda z.z\rangle\rangle$, M_V^0 is $\text{strip}_V(\langle x\ x \mid x = y\ y, y = \lambda x.y, z = \lambda z.z\rangle) \equiv \langle x\ x \mid x = y\ y, y = \lambda x.\Omega, z = \lambda z.\Omega\rangle$, $M_V^1 = \langle x\ x \mid x = (\lambda x.y)\ (\lambda x.y), y = \lambda x.\Omega, z = \lambda z.\Omega\rangle$. We have that every M_V^i is an acyclic term. Because only value declarations can be mutually recursive, every cycle is broken by the strip-rules. This is not completely obvious because there may be equations of the form $x = y$ left, but every cycle that only uses that type of equation gets broken by the first strip rule and every cycle that has at least one lambda on it gets broken by the second rule.

Theorem 18.29. *Given* $M \in \Lambda\circ_{\text{value}}$. $\text{Inf}_{\text{value}}(M) = \bigcup\{\text{Inf}_{\text{value}}(M_V^i) \mid i \geq 0\}$.

Proof.

$\supseteq$ The strip function and kernelization satisfy the following property:

$$\text{Inf}_{\mathcal{K}}(\mathcal{K}(M_V^i)) = \text{Inf}_{\mathcal{K}}((\mathcal{K}(M))_V^i) \ . \tag{18.1}$$

From Lemma 18.21 we get that for every $P \in \Lambda\mathcal{K}$:

$$\text{Inf}_{\mathcal{K}}(\text{strip}_V(P)) \subseteq \text{Inf}_{\mathcal{K}}(P) \ .$$

The result then follows from Lemma 18.18.

$\subseteq$ Given $M \xrightarrow[\lambda\circ_{\text{value}}]{} \!\!\!\!\to N$ we have by Lemma 18.11 that $\mathcal{K}(M) \xrightarrow[\lambda\mathcal{K}]{}\!\!\!\!\to \mathcal{K}(N)$. By Lemma 18.16(i) we have that $\mathcal{K}(M) \mapsto\!\!\!\!\!\!\to_{\lambda\mathcal{K}} Q_0$ with $\mathcal{K}(N) \preceq_{\text{value}} Q_0$. Let $P_0 \equiv \mathcal{K}(M)$. We apply the following recursive construction: If for some i we have that $P_i \mapsto\!\!\!\!\!\!\to_{\lambda\mathcal{K}} Q_i$ contracts a descendant of an acyclic substitution step in P_i then we define P_{i+1} by $P_i \xrightarrow[\text{GK(as)}]{}\!\!\!\!\to P_{i+1}$. By Lemma 18.15 there exists a Q_{i+1} such that $P_{i+1} \mapsto\!\!\!\!\!\!\to_{\lambda\mathcal{K}} Q_{i+1}$ in less steps than $P_i \mapsto\!\!\!\!\!\!\to_{\lambda\mathcal{K}} Q_i$ with $Q_i \preceq_{\text{value}} Q_{i+1}$.

Let P_n and Q_n be the last P_i and Q_i constructed like this. Because the standard sequence $P_n \mapsto\!\!\!\!\!\!\to_{\lambda\mathcal{K}} Q_n$ does not contract any descendant of a substitution redex in P_n we have that there exists a Q' with $\text{strip}_V(P_n) \xrightarrow[\lambda\mathcal{K}]{}\!\!\!\!\to Q'$ with $Q_n \preceq_{\text{value}} Q'$. Hence $\omega_{\text{value}}(N) \in \text{Inf}_{\mathcal{K}}((\mathcal{K}(M))_V^i)$. From 18.1 and Lemma 18.18 the result follows.

□

Let $[\![M]\!]$ denote the translation of the acyclic term M into a single-equation term by application of the let_M-rule, given by:

$$\langle M \mid x_1 = M_1, \cdots, x_n = M_n\rangle \xrightarrow[let_M]{} \langle \cdots \langle M \mid x_1 = M_1\rangle \cdots\rangle \mid x_n = M_n\rangle, \text{ if } n \neq 1 \ .$$

Let $\lambda_c\backslash\eta_V$ denote λ_c without the η_V-axiom and let $\lambda\circ_{\text{value}}^{\text{acyclic}}$ denote the $\lambda\circ_{\text{value}}$ where all left and right-hand sides of axioms have been restricted to acyclic terms. As before for good terms, the set of acyclic terms is closed under rewriting, but a cyclic term may rewrite to an acyclic term.

Theorem 18.30.

(i) *Given terms* $M, N \in \Lambda_c$. *If* $\lambda_c\backslash\eta_V \vdash M = N$ *then* $\lambda\circ_{\text{value}} \vdash M = N$.

(ii) *Given acyclic terms* $M, N \in \Lambda\circ_{\text{value}}$. *If* $\lambda\circ^{\text{acyclic}}_{\text{value}} \vdash M = N$ *then* $\lambda_c\backslash\eta_V \vdash [\![M]\!] = [\![N]\!]$.

PROOF.

(i) We can derive every axiom in the restricted commutative Moggi system:

- β_V: From left to right do a $\beta\circ$-step, followed by an es_V-step for every occurrence of x. Then apply garbage collection twice to first remove the equation and then remove the box.
- *id*: This is naming written right to left.
- *comp*: Apply internal merge to the left-hand side and external merge to the right-hand side to obtain $\langle P \mid y = M, x = N\rangle$.
- let_V: From left to right apply es_V for every occurrence of x and then apply garbage collection twice to first remove the equation and then remove the box.
- let_1 and let_2: From left to right apply naming and lift.
- let_C: Apply external merge to both sides.

(ii) Instead of provable equality we will consider conversion, which also for acyclic terms is the same as provable equality. We turn Moggi's calculus into a rewriting system by orienting the axioms from left to right. We will prove the following diagram:

$$\begin{tikzcd} M \arrow[r, leftrightarrow, "\lambda\circ_{\text{value}}"] \arrow[d, "let_M"'] & N \arrow[d, "let_M"] \\ {[\![M]\!]} \arrow[r, dashed, leftrightarrow, "\lambda_c\backslash\eta_V"'] & {[\![N]\!]} \end{tikzcd} \qquad (18.2)$$

where an arrow $\twoheadrightarrow\!\!\!\!|$ denotes reduction to normal form. The construction of diagram 18.2 consists of two steps. In the first step we transform the $\lambda\circ_{\text{value}}$ conversion into a $\lambda_c\backslash\eta_V/let_M$-conversion on $\Lambda\circ_{\text{value}}$, In the second step we transform that conversion into a $\lambda_c\backslash\eta_V$-conversion on Λ_c. This conversion is then the proof of $[\![M]\!] = [\![N]\!]$ in $\lambda_c\backslash\eta_V$. When we apply the axioms of $\lambda_c\backslash\eta_V$ to $\Lambda\circ_{\text{value}}$ then we count bindings. *E.g.,* we can only apply *id* to a letrec with exactly one declaration. In particular NV does *not* include letrecs with multiple bindings.

- For the first step we derive every axiom in $\lambda\circ_{\text{value}}$. We can derive the $\beta\circ$-axiom as follows:

$$\begin{array}{rcll} (\lambda x.M)N & = & (\lambda x.M)\texttt{let } x = N \texttt{ in } x & id \\ & = & \texttt{let } y = \texttt{let } x = N \texttt{ in } x \texttt{ in } (\lambda x.M)y & let_2 \\ & = & \texttt{let } x = N \texttt{ in let } y = x \texttt{ in } (\lambda x.M)y & comp \\ & = & \texttt{let } x = N \texttt{ in } (\lambda x.M)x & let_V \\ & = & \texttt{let } x = N \texttt{ in } M & \beta_V \end{array}$$

In diagrams that means:

$$\begin{tikzcd} {} \arrow[rrrrr, "\beta\circ"] \arrow[d, phantom, "\equiv"] & & & & & {} \arrow[d, phantom, "\equiv"] \\ {} & {} \arrow[l, dashed, "id"] \arrow[r, dashed, "let_2"'] & {} \arrow[r, dashed, "comp"'] & {} \arrow[r, dashed, "let_V"'] & {} \arrow[r, dashed, "\beta_V"'] & {} \end{tikzcd}$$

To deal with the application lift axioms we use a simple trick. We rewrite the left and right-hand sides of the axioms to let_M normal form and then find a $\lambda_c \backslash \eta_V$-conversion: Using the abbreviation $L \equiv \texttt{let}\ x = N\ \texttt{in}$ we get:

$$
\begin{array}{l}
(L_1 \cdots L_m M)\ (L'_1 \cdots L'_n N) \\
\quad = \ (L_1 \cdots L_m \texttt{let}\ x = M\ \texttt{in}\ x)\ (L'_1 \cdots L'_n N) \\
\quad = \ \texttt{let}\ y = L_1 \cdots L_m \texttt{let}\ x = M\ \texttt{in}\ x\ \texttt{in}\ y\ (L'_1 \cdots L'_n N) \\
\quad = \ L_1 \cdots L_m \texttt{let}\ y = \texttt{let}\ x = M\ \texttt{in}\ x\ \texttt{in}\ y\ (L'_1 \cdots L'_n N) \\
\quad = \ L_1 \cdots L_m \texttt{let}\ y = \texttt{let}\ x = M\ \texttt{in}\ x\ \texttt{in}\ y\ (L'_1 \cdots L'_n \texttt{let}\ u = N\ \texttt{in}\ u) \\
\quad = \ L_1 \cdots L_m \texttt{let}\ y = \texttt{let}\ x = M\ \texttt{in}\ x\ \texttt{in}\ \texttt{let}\ v = L'_1 \cdots L'_n \texttt{let}\ u = N\ \texttt{in}\ u\ \texttt{in}\ y\ v \\
\quad = \ L_1 \cdots L_m \texttt{let}\ y = \texttt{let}\ x = M\ \texttt{in}\ x\ \texttt{in}\ L'_1 \cdots L'_n \texttt{let}\ v = \texttt{let}\ u = N\ \texttt{in}\ u\ \texttt{in}\ y\ v \\
\quad = \ L_1 \cdots L_m L'_1 \cdots L'_n \texttt{let}\ y = \texttt{let}\ x = M\ \texttt{in}\ x\ \texttt{in}\ \texttt{let}\ v = \texttt{let}\ u = N\ \texttt{in}\ u\ \texttt{in}\ y\ v \\
\quad = \ L_1 \cdots L_m L'_1 \cdots L'_n (M\ N)
\end{array}
$$

This proves the following diagram:

$$
\begin{tikzcd}
\cdot \arrow[r, "\mathrm{lift}"] \arrow[d, "let_M"'] & \cdot \arrow[d, "let_M"] \\
\cdot \arrow[r, dashed, leftrightarrow, "\lambda_c\backslash\eta_V"'] & \cdot
\end{tikzcd}
$$

For the other axioms we may derive:

$$
\begin{tikzcd}
\cdot \arrow[rr, "\mathrm{es_V,is_V}"] \arrow[d, dashed, "let_M"'] & & \cdot \arrow[d, dashed, "let_M"] \\
\cdot \arrow[r, dashed, "let_V"'] & \cdot & \cdot \arrow[l, dashed, "let_V"]
\end{tikzcd}
\qquad
\begin{tikzcd}
\cdot \arrow[rr, "\lambda\mathrm{-lift}"] \arrow[d, dashed, "let_M"'] & & \cdot \arrow[d, dashed, "let_M"] \\
\cdot \arrow[r, dashed, two heads, "let_V"'] & \cdot & \cdot \arrow[l, dashed, two heads, "let_V"]
\end{tikzcd}
\qquad
\begin{tikzcd}
\cdot \arrow[r, "\mathrm{em}"] \arrow[d, dashed, "let_M"'] & \cdot \arrow[dd, dashed, "let_M"] \\
\cdot \arrow[d, dashed, "let_M"'] & \\
\cdot \arrow[r, phantom, "\equiv"] & \cdot
\end{tikzcd}
$$

$$
\begin{tikzcd}
\cdot \arrow[r, "\mathrm{im}"] \arrow[d, dashed, "let_M"'] & \cdot \arrow[dd, dashed, "let_M"] \\
\cdot \arrow[d, dashed, "let_M"'] & \\
\cdot \arrow[r, dashed, two heads, "comp"'] & \cdot
\end{tikzcd}
\qquad
\begin{tikzcd}
\cdot \arrow[r, "\mathrm{name}"] \arrow[d, phantom, "\equiv"] & \cdot \arrow[d, phantom, "\equiv"] \\
\cdot & \cdot \arrow[l, dashed, "id"]
\end{tikzcd}
\qquad
\begin{tikzcd}
\cdot \arrow[rr, "\mathrm{cp_V}"] \arrow[d, phantom, "\equiv"] & & \cdot \arrow[d, phantom, "\equiv"] \\
\cdot \arrow[r, dashed, two heads, "\mathrm{es_V,is_V}"'] & \cdot & \cdot \arrow[l, dashed, two heads, "\mathrm{es_V,is_V}"]
\end{tikzcd}
$$

- To finish diagram 18.2 we first reduce every term in the conversion to let_M normal form. Because let_M does not work on single binding let's, the let_M rule and all the Moggi rules are orthogonal. Since let_M cannot duplicate redexes we get:

$$
\begin{tikzcd}
\cdot \arrow[r, "\lambda_c\backslash\eta_V"] \arrow[d, "let_M"'] & \cdot \arrow[d, dashed, two heads, "let_M"] \\
\cdot \arrow[r, dashed, "\lambda_c\backslash\eta_V"'] & \cdot
\end{tikzcd}
$$

Because all axioms in $\lambda_c \backslash \eta_V$ preserve single binding terms we can deduce from this diagram that:

$$
\begin{tikzcd}
\cdot \arrow[r, "\lambda_c\backslash\eta_V"] \arrow[d, two heads, "let_M"'] & \cdot \arrow[d, dashed, two heads, "let_M"] \\
\cdot \arrow[r, dashed, "\lambda_c\backslash\eta_V"'] & \cdot
\end{tikzcd}
$$

Identity	$\langle x \mid x = M\rangle$	$=$	M	$x \notin FV(M)$
Associativity	$\langle N \mid y = \langle M \mid D_1\rangle, D_2\rangle$	$=$	$\langle N \mid y = M, D_1, D_2\rangle$	$D_1 \neq \{\}$
	$\langle\langle M \mid D_1\rangle \mid D_2\rangle$	$=$	$\langle M \mid D_1, D_2\rangle$	$D_1 \neq \{\}, D_2 \neq \{\}$
Commutativity	$\langle M \mid D\rangle\ N$	$=$	$\langle M\ N \mid D\rangle$	$D \neq \{\}$
	$M\ \langle N \mid D\rangle$	$=$	$\langle M\ N \mid D\rangle$	$D \neq \{\}$
β	$(\lambda x.M)\ N$	$=$	$\langle M \mid x = N\rangle$	
σ_v	$\langle M \mid x = V, y = C[x], D\rangle$	$=$	$\langle M \mid x = V, y = C[V], D\rangle$	$x \not\equiv y$
	$\langle C[x] \mid x = V, D\rangle$	$=$	$\langle C[V] \mid x = V, D\rangle$	
	$\langle M \mid x = V\rangle$	$=$	M	$x \notin FV(M) \cup FV(V)$
η_0	$\lambda x.y\ x$	$=$	y	

Table 14. Hasegawa's call-by-value calculus

We now have a series of $\lambda_c \backslash \eta_V$-conversions separated by let_M-conversions. We can fill in those gaps with the following diagram:

$$\begin{tikzcd}
\cdot \arrow[r, phantom, "\equiv"] \arrow[d, ->>, "let_M"'] & \cdot \arrow[d, ->>, "let_M"] \\
\cdot \arrow[r, dashed, ->>, "leq_C"'] & \cdot
\end{tikzcd}$$

□

Recently, Hasegawa [Has97] proposed a cyclic extension of Moggi's calculus, see Table 14. There are three major differences between Hasegawa's calculus and our own: (i) Hasegawa uses simply typed terms and treats values differently depending on them being cyclic or acyclic, (ii) he does not have lifting of values out of a lambda as an axiom †, and (iii) he restricts garbage collection to acyclic values only. There is one minor difference and that is that environments are always non-empty. Unfortunately, Hasegawa does not study the rewriting aspects of the calculus. But in a sense, his calculus is complete with respect to our calculus:

THEOREM 18.31. *Given $M \in \Lambda\circ$, such that there is no subterm $\langle N \mid\rangle$. We have that:*

$$\mathrm{Inf}_{\mathrm{value}}(M) =\downarrow \{\omega_{\mathrm{value}}(N) \mid \text{Hasegawa} \setminus \eta_0 \vdash M = N\}\ .$$

PROOF.

"$\supseteq$" Obvious from the fact that every axiom in Hasegawa $\setminus \eta_0$ is derivable in $\lambda\circ_{\mathrm{value}}$.

"$\subseteq$" From Lemma 18.16(i) and the following two facts:

$$\text{Hasegawa} \vdash M = \mathcal{K}(M) \text{ and } P \underset{\lambda\mathcal{K}}{\longmapsto} Q \Rightarrow \text{Hasegawa} \vdash P = Q\ .$$

□

REMARK 18.32. The η-axioms present in Moggi's and Hasegawa's calculi are not sound in our model, *e.g.,* $\lambda x.y\ x$ and y, $\langle M \mid x = x\rangle$ and $\langle M \mid x = \lambda z.x\ z\rangle$ do not have the same infinite normal forms. The latter example shows that simply adding η-expansion or reduction to the ω-rules is not a possible way of adding the η-axiom to our calculus.

† For acyclic values the axiom is derivable from substitution and garbage collection. The case for cyclic values is not derivable.

19. Extensions to data structures

We can extend the previous calculi with data constructors by encoding cons and the related destructors as functions and then deriving the associated rewriting rules. For example, in $\lambda\circ_{\mathrm{name}}$, we have:

$$\begin{array}{lcl} \mathsf{cons}(M,N) & \equiv & \lambda p.pMN \\ \mathsf{head}(M) & \equiv & M(\lambda x_1 x_2.x_1) \\ \mathsf{tail}(M) & \equiv & M(\lambda x_1 x_2.x_2) \ . \end{array}$$

This entails the following set of rules:

$$\begin{array}{lclclcl} \mathsf{head}(\mathsf{cons}(M,N)) & \to & M & \quad & \mathsf{head}(\langle M \mid D\rangle) & \to & \langle \mathsf{head}(M) \mid D\rangle \\ \mathsf{tail}(\mathsf{cons}(M,N)) & \to & N & & \mathsf{tail}(\langle M \mid D\rangle) & \to & \langle \mathsf{tail}(M) \mid D\rangle \ . \end{array}$$

In the development of the model for call-by-name we did not impose any restriction, *i.e.,* we allow reduction under lambda and our results also apply to open terms. Thus, the model construction for the extended calculus is a trivial extension of $\lambda\circ_{\mathrm{name}}$.

In $\lambda\circ_{\mathrm{share}}$, we encode $\mathsf{cons}(M,N)$ as $(\lambda x_1 x_2 p.px_1x_2)MN$. This encoding is interesting, since it points out that, before substituting a term of the form $\mathsf{cons}(\mathsf{cons}(1,\mathsf{nil}),\mathsf{nil})$, a name is associated to the head and tail, as the following example shows:

$$\begin{array}{l} \langle x \mid x = \mathsf{cons}(\mathsf{cons}(1,\mathsf{nil}),\mathsf{nil})\rangle \to \langle x \mid x = \langle \mathsf{cons}(x_1,x_2) \mid x_1 = \mathsf{cons}(1,\mathsf{nil}), x_2 = \mathsf{nil}\rangle\rangle \to \\ \langle x \mid x = \mathsf{cons}(x_1,x_2), x_1 = \mathsf{cons}(1,\mathsf{nil}), x_2 = \mathsf{nil}\rangle \to \langle \mathsf{cons}(x_1,x_2) \mid x = \mathsf{cons}(x_1,x_2),\cdots\rangle \to \cdots \end{array}$$

Apparently, we need the following axiom for cons:

$$\mathsf{cons}(M,N) \to \langle \mathsf{cons}(x_1,x_2) \mid x_1 = M, x_2 = N\rangle \ .$$

However, this axiom leads to an infinite computation, which is not preserved by the encoding. This suggests that we need a new constructor indicating that both head and tail are variables. This constructor is $\underline{\mathsf{cons}}(x_1,x_2)$, encoded as $\lambda p.px_1x_2$. We have the rules:

$$\begin{array}{lcl} \mathsf{cons}(M,N) & \to & \langle \underline{\mathsf{cons}}(x_1,x_2) \mid x_1 = M, x_2 = N\rangle \\ \mathsf{head}(\underline{\mathsf{cons}}(M,N)) & \to & M \\ \mathsf{tail}(\underline{\mathsf{cons}}(M,N)) & \to & N \end{array} \qquad \begin{array}{l} \mathsf{head}(\langle M \mid D\rangle) \to \langle \mathsf{head}(M) \mid D\rangle \\ \mathsf{tail}(\langle M \mid D\rangle) \ \ \to \langle \mathsf{tail}(M) \mid D\rangle \ . \end{array}$$

We extend the class of values V to include $\underline{\mathsf{cons}}\ x_1\ x_2$.

With respect to the lazy strategy, data constructors behave the same as functions. Instead, the lenient strategy distinguishes them. The lenient strategy of the extended calculus is defined by adding the following clauses to the definition of evaluation context of Table 9:

$$E_{lenient} \quad ::= \quad \cdots \mid \mathsf{head}(E) \mid \mathsf{tail}(E) \mid \mathsf{cons}(E,M) \mid \mathsf{cons}(M,E) \ .$$

Note that reduction can occur under a cons but not under a $\underline{\mathsf{cons}}$. This is necessary to guarantee confluence. Otherwise, $\langle x \mid x = \underline{\mathsf{cons}}(x,x)\rangle$ will lead to

$$\langle x \mid x = \underline{\mathsf{cons}}(\underline{\mathsf{cons}}(x,x),x)\rangle$$

and

$$\langle x \mid x = \underline{\mathsf{cons}}(x,\underline{\mathsf{cons}}(x,x))\rangle \ ,$$

which are clearly out of synch.

20. Conclusions

We have developed a precise connection between the terms of the lambda calculus extended with `letrec` (cyclic terms) and the class of well-formed cyclic lambda graphs. We have given this connection in the form of an axiom system, called the representational calculus, that is sound and complete. We have extended the axiom system to handle well-formed graphs up to garbage collection and alternative scoping. The extension to garbage collection is sound and complete, but the extension to alternative scoped graphs is only complete with respect to garbage free graphs. We also have extended the axiom system to be sound and complete with respect to unwinding of graphs. We have concluded the investigation of representations of graphs by discussing briefly several possible variations of the representational axioms. Finding an axiom system that is sound and complete for alternatively scoped graphs with garbage is part of further research.

The axiom system for tree unwinding combined with a notion of β-reduction constitutes our axiomatization of cyclic lambda structures. The presence of cycles and lambda-abstraction causes confluence to fail. Thus, instead of showing confluence, we have shown that our cyclic calculus satisfies an approximate notion of confluence. This notion guarantees uniqueness of the infinite normal form of a cyclic term. The infinite normal form, being compositional, provides a tool to reason about correctness of optimizations. The soundness of our cyclic calculus is shown with respect to the lambda calculus and the infinitary lambda calculus. Our cyclic lambda calculus can be used to show correctness of a wide range of evaluation calculi with respect to the infinitary lambda calculus. We however cannot capture optimal implementations since we do not have the ability to represent irregular infinite lambda terms.

To reason about the optimization and execution of non-strict functional languages we have developed a variant of our cyclic calculus that takes sharing into consideration. The new calculus is obtained by restricting the operations that cause a duplication, such as substitution, copying and lambda-lifting, to duplicate values only, where a value is either a variable or a lambda-abstraction. We show that these restrictions do not change the infinite normal form of a term. Next, it would be interesting to extend the sharing calculus with a notion of assignment. This is to provide a framework for reasoning about the correctness of the Haskell implementation of the monadic operations, which use updates in place.

To reason about strict languages we have developed yet another calculus, the cyclic call-by-value calculus. This calculus is obtained by restricting the sharing calculus to garbage collect values only, instead of arbitrary terms. This calculus is related to Moggi's computational lambda calculus.

In summary, we have developed three calculi: $\lambda\circ_{\text{name}}$, $\lambda\circ_{\text{share}}$ and $\lambda\circ_{\text{value}}$. These calculi correspond to the parameter-passing techniques of call-by-name, call-by-need and call-by-value. The ability to define mutually recursive objects makes these calculi more suitable then lambda calculus [Bar84], λ_{need} [AFM$^+$95, AF97] and λ_{V} [Plo75] to express the operational semantics, compilation and optimization of current functional languages.

References

AA95 Z. M. Ariola and Arvind. Properties of a first-order functional language with sharing. *Theoretical Computer Science*, 146:69–108, 1995.

AAH$^+$93 Arvind, L. Augusston, J. Hicks, R. S. Nikhil, S. Peyton-Jones, J. Stoy, and W. Williams. pH: A Parallel Haskell. Technical report, MIT Laboratory for Computer Science, September 1993.

AB97 Z. M. Ariola and S. Blom. Cyclic lambda calculi. In *Proc. TACS 94, Sendai, Japan*, 1997.

ACCL91 M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 4(1):375–416, 1991.

AF97 Z. M. Ariola and M. Felleisen. The call-by-need lambda calculus. *Journal of Functional Programming*, 7(3), 1997.

AFM+95 Z. M. Ariola, M. Felleisen, J. Maraist, M. Odersky, and P. Wadler. The call-by-need lambda calculus. In *Proc. ACM Conference on Principles of Programming Languages*, pages 233–246, 1995.

AK Z. M. Ariola and J. W. Klop. Lambda calculus with explicit recursion. Technical Report CIS-TR-96-04, Department of computer and information science, University of Oregon. To appear in Information and computation.

AK94 Z. M. Ariola and J. W. Klop. Cyclic lambda graph rewriting. In *Proc. Ninth Symposium on Logic in Computer Science (LICS'94), Paris, France*, pages 416–425, 1994.

AK96 Z. M. Ariola and J. W. Klop. Equational term graph rewriting. *Fundamentae Informaticae*, 26(3,4):207–240, 1996. Extended version: CWI Report CS-R9552.

AKK+94 Z. M. Ariola, J. W. Klop, J. R. Kennaway, F. J. de Vries, and M. R. Sleep. Syntactic definitions of undefined: On defining the undefined. In *Proc. TACS 94, Sendai, Japan*, 1994.

AL94 A. Asperti and C. Laneve. Interaction systems I: The theory of optimal reductions. *Mathematical structures for computer science*, 4:457–504, 1994.

AMNS97 Arvind, J-W. Maessen, R.S. Nikhil, and J. E. Stoy. λ_s: an implicitly parallel λ-calculus with letrec, synchronization and side-effects. Technical Report 393, MIT Laboratory for Computer Science, 1997.

Ari96 Z. M. Ariola. Relating graph and term rewriting via Böhm models. *Applicable Algebra in Engineering, Communication and Computing*, 7(5), 1996.

Bar84 H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* North-Holland, Amsterdam, 1984.

BL89 S. Billot and B. Lang. The structure of shared forests in ambiguous parsing. In *Proceedings of the Twenty-Seventh Annual Meeting of the Association for Computational Linguistics*, 1989. Association for Computational Linguistics.

BLR96 Z. Benaissa, P. Lescanne, and K.H. Rose. Modeling sharing and recursion for weak reduction strategies using explicit substitution. In *PLIP'96*, 1996.

BvEG+87 H. P. Barendregt, M. C. J. D. van Eekelen, J. R. W. Glauert, J. R. Kennaway, M. J. Plasmeijer, and M. R. Sleep. Term graph rewriting. In J. W. de Bakker, A. J. Nijman, and P. C. Treleaven, editors, *Proc. Conference on Parallel Architecture and Languages Europe (PARLE '87), Eindhoven, The Netherlands, Springer-Verlag LNCS 259*, pages 141–158, 1987.

DJ90 N. Dershowitz and J. P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–320. Elsevier - The MIT Press, 1990.

Har86 B. Harper. Introduction to Standard ML. Technical report, ECS-LFCS-86-14, Laboratory for the Foundation of Computer Science, Edinburgh University, 1986.

Has97 M. Hasegawa. Recursion from cyclic sharing: traced monoidal categories and models of cyclic lambda calculi. In *Proc. Conference on Typed Lambda Calculi and Applications*, April 1997.

HPJW+92 P. Hudak, S. Peyton Jones, P. Wadler, B. Boutel, J. Fairbairn, J. Fasel, K. Hammond, J. Hughes, T. Johnsson, D. Kieburtz, R. Nikhil, W. Partain, and J. Peterson. Report on the programming language Haskell. *ACM SIGPLAN Notices*, 27(5):1–64, 1992.

Hue80 G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *JACM*, 27(4), 1980.

KKSdV95 J. R. Kennaway, J. W. Klop, M. R. Sleep, and F. J. de Vries. Infinitary lambda calculus. In *Proc. Rewriting Techniques and Applications, Kaiserslautern*, 1995.

Kli91 P. Klint. A meta-environment for generating programming environments. In *Algebraic Methods II: Theory, Tools and Applications. Springer-Verlag LNCS 490*, pages 105–124, 1991.

KV94 P. Klint and E. Visser. Using filters for the disambiguation of context-free grammars. In G. Pighizzini and P. San Pietro, editors, *Proc. ASMICS Workshop on Parsing Theory*, pages 1–20, Milano, Italy, October 1994. Tech. Rep. 126–1994, Dipartimento di Scienze dell'Informazione, Università di Milano.

Laf90 Y. Lafont. Interaction nets. In *Proc. ACM Conference on Principles of Programming Languages, San Francisco*, 1990.

Lau93 J. Launchbury. A natural semantics for lazy evaluation. In *Proc. ACM Conference on Principles of Programming Languages*, pages 144–154, 1993.

Les94 P. Lescanne. From $\lambda\sigma$ to $\lambda\upsilon$ a journey through calculi of explicit substitutions. In *Proc. 21st Symposium on Principles of Programming Languages (POPL '94), Portland, Oregon*, pages 60–69, 1994.

Lév78 J.-J. Lévy. *Réductions Correctes et Optimales dans le Lambda-Calcul*. PhD thesis, Universite Paris VII, October 1978.

Mac94 I.C. Mackie. *The geometry of implementation*. PhD thesis, University of London, 1994.

Mog88 E. Moggi. Computational lambda calculus and monads. Technical Report ECS-LFCS-88-86, Edinburgh University, 1988.

MOTW95 J. Maraist, M. Odersky, D. Turner, and P. Wadler. Call-by-name, call-by-value, call-by-need, and the linear lambda calculus. In *Proc. of Mathematical Foundations of Programming Semantics (MFPS)*, 1995.

Nie96 J. Niehren. Functional computation as concurrent computation. In *Proc. ACM Conference on Principles of Programming Languages*, pages 333–343, 1996.

Nik91 R. S. Nikhil. Id (version 90.1) reference manual. Technical Report 284-2, MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139, 1991.

PJ87 S. L. Peyton Jones. *The implementation of Functional Programming Languages*. Prentice-Hall International, Englewood Cliffs, N.J., 1987.

Plo75 G. D. Plotkin. Call-by-name, call-by-value and the lambda calculus. *Theoretical Computer Science*, 1:125–159, 1975.

Ros92 K. H. Rose. Explicit cyclic substitutions. In M. Rusinowitch and J. L. Rémy, editors, *Proc. 3rd International Workshop on Conditional Term Rewriting Systems (CTRS-92), Pont-à-Mousson, France, Springer-Verlag LNCS 656*, pages 36–50, 1992.

Sel96 P. Selinger. Order-incompleteness and finite lambda models. In *Proc. Symposium on Logic in Computer Science (LICS'96)*, 1996.

SM93 D. Sangiorgi and R. Milner. Techniques of "weak bisimulation up to". Technical report, 1993.

Tom85 M. Tomita. *Efficient Parsing for Natural Languages. A Fast Algorithm for Practical Systems.* Kluwer Academic Publishers, 1985.

vO94 V. van Oostrom. *Confluence for Abstract and Higher-Order Rewriting.* PhD thesis, Vrije Universiteit, 1994.

Wad71 C. Wadsworth. *Semantics and Pragmatics of the Lambda-Calculus.* 1971. PhD thesis, University of Oxford.