Appendix of "Safety of Nöcker's Strictness Analysis"

Manfred Schmidt-Schauß, David Sabel

Institut für Informatik,
Johann Wolfgang Goethe-Universität,
Postfach 11 19 32,
D-60054 Frankfurt, Germany,
(*e-mail:* schauss@ki.informatik.uni-frankfurt.de)

Marko Schütz

Dept. of Mathematics and Computing Science,
University of the South Pacific,
Suva, Fiji Islands

Abstract

This document contains full proofs of (Schmidt-Schauß *et al.*, n.d.).

Contents

A Proof of the Context Lemma

In this section we prove the context lemma (Lemma 2.3). We will show the claim:

Lemma A.1
Let s, t be terms. If for all reduction contexts R: $(R[s]\Downarrow \Rightarrow R[t]\Downarrow)$, then $\forall C$: $(C[s]\Downarrow \Rightarrow C[t]\Downarrow)$; i.e. $s \leq_c t$.

Proof
Multicontexts are like contexts, but have several holes $\cdot_i$, and every hole occurs exactly once in the term. We write a multicontext as $C[\cdot_1, \ldots, \cdot_n]$, and if the terms s_i for $i = 1, \ldots, n$ are plugged into the holes $\cdot_i$, then we denote the resulting term as $C[s_1, \ldots, s_n]$. Note that variable capture is allowed for multicontexts.

We prove the more general claim:

For $i = 1, \ldots, n$, let s_i, t_i be expressions. Let the following hold:
$\forall i : \forall$ reduction contexts R: $(R[s_i]\Downarrow \Rightarrow R[t_i]\Downarrow)$.

Then $\forall C : C[s_1, \ldots, s_n]\Downarrow \Rightarrow C[t_1, \ldots, t_n]\Downarrow$.

Assume the claim is false. Then there is a counterexample. I.e., there is a multicontext C, a number $n \geq 1$ and terms s_i, t_i for $i = 1, \ldots, n$, such that $\forall i : \forall$ reduction contexts R: $(R[s_i]\Downarrow \Rightarrow R[t_i]\Downarrow)$, and $C[s_1, \ldots, s_n]\Downarrow$, but $C[t_1, \ldots, t_n]\Uparrow$.
We select the counterexample minimal w.r.t. the following lexicographic ordering:

1. the number of normal order reduction steps of a shortest evaluation of $C[s_1, \ldots, s_n]$.
2. the number of holes of C.

Either some hole of $C[\cdot_1, \ldots, \cdot_n]$ is in a reduction context or no hole is in a reduction context. The definition of reduction contexts and some easy reasoning shows that the unwind applied to $C[\cdot_1, \ldots, \cdot_n]$ either arrives at some hole, or does not arrive at a hole, and moreover, that this is not affected by the terms plugged into the holes.

If one hole of $C[\cdot_1, \ldots, \cdot_n]$ is in a reduction context, then we assume wlog that it is the first one.

Then $C[\cdot, t_2 \ldots, t_n]$ is a reduction context. Let $C' := C[s_1, \cdot_2, \ldots, \cdot_n]$. Since

$C'[s_2, \ldots, s_n] \equiv C[s_1, \ldots, s_n]$, these expressions have the same normal order reduction. Since the number of holes is smaller, we obtain $C'[t_2, \ldots, t_n]{\Downarrow}$, which means $C[s_1, t_2, \ldots, t_n]{\Downarrow}$. Since $C[\cdot, t_2, \ldots, t_n]$ is a reduction context, the preconditions of the lemma applied to s_1, t_1 imply $C[t_1, t_2, \ldots, t_n]{\Downarrow}$, a contradiction.

If no hole of $C[\cdot_1, \ldots, \cdot_n]$ is in a reduction context, then the first normal order reduction step $C[s_1, \ldots, s_n] \xrightarrow{n} C'[s'_1, \ldots, s'_m]$ can also be used for $C[t_1, \ldots, t_n]$ giving $C'[t'_1, \ldots, t'_m]$, where for every $i : \rho_{i,j}(s_j, t_j) = (s'_i, t'_i)$ for some variable renaming $\rho_{i,j}$ and some j. To verify this, we have to check that for a normal order redex, the parts that are modified are also in a reduction context.

- in a (cp) normal order reduction, every superterm of the to-variable position is in a reduction context.
- For normal order reductions (llet), (lapp), (lcase), (lseq), the inner `letrec` is in a reduction context.
- The constructor application used in a (case) is in a reduction context.

The following may happen to the terms s_i, t_i in the holes in one reduction step:

- If the hole is in an alternative of a (case)-expression that is discarded by the reduction, then the hole, and hence s_i as well as t_i, is eliminated after reduction.
- If the hole is not eliminated, and if the reduction is not a (cp), then the terms s_i, t_i in the holes are unchanged and also not copied, but both may appear at a different position in the resulting expression.
- If the reduction is a (cp), and the hole is not in the copied expression, then again the terms s_i, t_i in the holes are unchanged and also not copied.
- If the reduction is a (cp), and the hole is within the copied expression, then the terms s_i, t_i in the holes may be duplicated giving s'_i, t'_i. Since the reduction is a normal order reduction, and since we have assumed the "distinct bound variable convention", the renaming concerns the free variables in s_i, t_i which are bound in C. For a fixed i, we can use the same renaming ρ_i for the variables in s_i and t_i, so we have $\rho_i(s_i) = s'_i, \rho_i(t_i) = t'_i$. This means that the assumption holds also for the new pair of terms:

$$\forall i : \forall \text{ reduction contexts } R : (R[s'_i]{\Downarrow} \Rightarrow R[t'_i]{\Downarrow}).$$

Now we can use induction on the number of $\xrightarrow{n}$-reductions.

Since the length of an evaluation of $C[s'_1, \ldots, s'_m]$ is strictly smaller, we also have $C'[t'_1, \ldots, t'_m]{\Downarrow}$. But then we have also $C[t_1, \ldots, t_n]{\Downarrow}$, which contradicts the assumption that we have chosen a counterexample.

Now we look at the base case. If C has no holes, then a counterexample is impossible.

If the length of an evaluation is 0, then $C[s_1, \ldots, s_n]$ is already a WHNF. Since we can assume that no hole is in a reduction context, the context is a WHNF, and thus this holds for $C[t_1, \ldots, t_n]$ as well, which is impossible.

Concluding, we have proved that there is no counterexample to the general claim, hence the lemma holds, since it is a specialization of this claim. □

B Correctness of Reductions

In this section we prove that the reduction rules of the calculus LR (see Definition 1.3) and the extra transformation rules defined in Definition 2.8 are correct program transformations, i.e. maintain contextual equivalence.

Non-normal order reduction steps for the language LR are called *internal* and denoted by a label i. An internal reduction in a reduction context is marked by $i\mathcal{R}$, and an internal reduction in a surface context by $i\mathcal{S}$.

B.1 The Reductions (case-c), (seq-c), (lbeta), (lapp), (lcase), (lseq)

Lemma B.1
Every a-reduction in a reduction context where $a \in \{(\text{case-c}), (\text{seq-c}), (\text{lbeta}), (\text{lapp}), (\text{lcase}), (\text{lseq})\}$ is a normal order reduction.

Proof
This follows by checking the possible term structures in a reduction context. □

Proposition B.2
Contextual equivalence remains unchanged under the reductions (case-c), (seq-c), (lbeta), (lapp), (lcase), (lseq). I.e. $s \xrightarrow{a} t$ with $a \in \{(\text{case-c}), (\text{seq-c}), (\text{lbeta}), (\text{lapp}), (\text{lcase}), (\text{lseq})\}$ implies $s \sim_c t$.

Proof
This follows from the context lemma A.1. It is sufficient to consider $R[s]$ and $R[t]$. From $s \xrightarrow{a} t$ and Lemma B.1 it follows that $R[s] \xrightarrow{n} R[t]$.

Since normal order reduction is unique, it follows $R[s]{\Downarrow}$ iff $R[t]{\Downarrow}$. Now we apply the context lemma. □

The reductions (lll), (cp), (case-e), (case-in), (seq-e), (seq-in) may be non-normal order in a reduction context, so other arguments are required.

B.2 Complete Sets of Commuting and Forking Diagrams

For proving correctness of further program transformations, we require the notions of complete sets of commuting diagrams and of complete sets of forking diagrams.

A *reduction sequence* is of the form $t_1 \to \ldots \to t_n$, where t_i are terms and $t_i \to t_{i+1}$ is a reduction as defined in definition 1.3. In the following definition we describe transformations on reduction sequences. Therefore we use the notation

$$\xrightarrow{iX,red} \cdot \xrightarrow{n,a_1} \ldots \xrightarrow{n,a_k} \quad \rightsquigarrow \quad \xrightarrow{n,b_1} \ldots \xrightarrow{n,b_m} \cdot \xrightarrow{iX,red_1} \ldots \xrightarrow{iX,red_h}$$

for transformations on reduction sequences. Here the notation $\xrightarrow{iX,red}$ means a reduction with $iX \in \{iC, i\mathcal{R}, i\mathcal{S}\}$, and *red* is a reduction from LR.

In order for the above transformation rule to be applied to the prefix of the reduction sequence RED, the prefix has to be $s \xrightarrow{iX,red} t_1 \xrightarrow{n,a_1} \ldots t_k \xrightarrow{n,a_k} t$. Since we will use sets of transformation rules, it may be the case that there is a transformation rule in the set, where the pattern matches a prefix, but it is not applicable, since the right hand side cannot be constructed.

We will say the transformation rule

$$\xrightarrow{iX,red} . \xrightarrow{n,a_1} \ldots \xrightarrow{n,a_k} \quad \rightsquigarrow \quad \xrightarrow{n,b_1} \ldots \xrightarrow{n,b_m} . \xrightarrow{iX,red_1} \ldots \xrightarrow{iX,red_h}$$

is *applicable* to the prefix $s \xrightarrow{iX,red} x_1 \xrightarrow{n,a_1} \ldots x_k \xrightarrow{n,a_k,} t$ of the reduction sequence RED iff the following holds:

$$\exists y_1, \ldots, y_m, z_1, \ldots, z_{h-1} :$$
$$s \xrightarrow{n,b_1} y_1 \ldots \xrightarrow{n,b_m} y_m \xrightarrow{iX,red_1} z_1 \ldots z_{h-1} \xrightarrow{iX,red_h} t$$

The transformation consists in replacing this prefix with the result:

$$s \xrightarrow{n,b_1} t'_1 \ldots t'_{m-1} \xrightarrow{n,b_m} t'_m \xrightarrow{iX,red_1} t''_1 \ldots t''_{h-1} \xrightarrow{iX,red_h} t$$

where the terms in between are appropriately constructed.

Example B.3

An example of a single commuting diagram is $\xrightarrow{i\mathcal{S},cp} . \xrightarrow{n,seq} \rightsquigarrow \xrightarrow{n,seq} . \xrightarrow{n,cp}$. Consider the reduction before and after an application of the diagram rule.

$$\begin{array}{ll} & (\texttt{seq}\ (\lambda u.u)\ (\texttt{letrec}\ x = \lambda y.c\ \texttt{in}\ (x\ d))) \\ \xrightarrow{i\mathcal{S},cp} & (\texttt{seq}\ (\lambda u.u)\ (\texttt{letrec}\ x = \lambda y.c\ \texttt{in}\ ((\lambda y.c)\ d))) \\ \xrightarrow{n,seq} & (\texttt{letrec}\ x = \lambda y.c\ \texttt{in}\ ((\lambda y'.c)\ d)) \\ \xrightarrow{n,lbeta} & (\texttt{letrec}\ x = \lambda y.c\ \texttt{in}\ (\texttt{letrec}\ y' = d\ \texttt{in}\ c)) \\ \ldots & \ldots \end{array}$$

Can be transferred using the diagram to

$$\begin{array}{ll} & (\texttt{seq}\ (\lambda u.u)\ (\texttt{letrec}\ x = \lambda y.c\ \texttt{in}\ (x\ d))) \\ \xrightarrow{n,seq} & (\texttt{letrec}\ x = \lambda y.c\ \texttt{in}\ (x\ d)) \\ \xrightarrow{n,cp} & (\texttt{letrec}\ x = \lambda y.c\ \texttt{in}\ ((\lambda y'.c)\ d)) \\ \xrightarrow{n,lbeta} & (\texttt{letrec}\ x = \lambda y.c\ \texttt{in}\ (\texttt{letrec}\ y' = d\ \texttt{in}\ c)) \\ \ldots & \ldots \end{array}$$

One can view the operation as shifting the $\xrightarrow{i\mathcal{S},cp}$ to the right. In this special case the transformation has turned the internal into a normal order reduction.

Definition B.4

• A *complete set of commuting diagrams for the reduction* $\xrightarrow{iX,red}$ is a set of transformation rules on reduction sequences of the form

$$\xrightarrow{iX,red} . \xrightarrow{n,a_1} \ldots \xrightarrow{n,a_k} \quad \rightsquigarrow \quad \xrightarrow{n,b_1} \ldots \xrightarrow{n,b_m} . \xrightarrow{iX,red_1} \ldots \xrightarrow{iX,red_{k'}},$$

where $k, k' \geq 0, m \geq 1, h > 1$, such that for every reduction sequence $t_0 \xrightarrow{iX,red} t_1 \xrightarrow{n} \ldots \xrightarrow{n} t_h$, where t_h is a WHNF, at least one of the transformation rules is applicable to a prefix of the sequence.
In the proofs below using the complete sets of commuting diagrams, the case $h = 1$ must be treated separately in the induction base.

• A *complete set of forking diagrams for the reduction* $\xrightarrow{iX,red}$ is a set of transformation rules on reduction sequences of the form

$$\xleftarrow{n,a_1} \dots \xleftarrow{n,a_k} \cdot \xrightarrow{iX,red} \quad \rightsquigarrow \quad \xrightarrow{iX,red_1} \dots \xrightarrow{iX,red_{k'}} \cdot \xleftarrow{n,b_1} \dots \xleftarrow{n,b_m},$$

where $k, k' \geq 0, m \geq 1, h > 1$, such that for every reduction sequence $t_h \xleftarrow{n} \dots t_2 \xleftarrow{n} t_1 \xrightarrow{iX,red} t_0$, where t_h is a WHNF, at least one of the transformation rules from the set is applicable to a suffix of the sequence. In the proofs below using the complete sets of forking diagrams, the case $h = 1$ must be treated separately in the induction base.

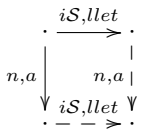
The two different kinds of diagrams are required for two different parts of the proof of the contextual equivalence of two terms.

Note that there may be different complete sets of, say, commuting diagrams for a single transformation. It depends on the needs of further proofs, which one is most appropriate.

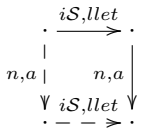
As a notation, we also use the * and +-notation of regular expressions for the diagrams. The interpretation is obvious and is intended to stand for an infinite set accordingly constructed.

In most of the cases, the same diagrams can be drawn for a complete set of commuting and a complete set of forking diagrams, though the interpretation is different for the two kinds of diagrams. We will exploit this to keep the presentation simple and give in general only the drawing in the form of diagrams. The starting term is in the northwestern corner, and the normal order reduction sequences are always downwards. where the deviating reduction is pointing to the east. There are rare exceptions for degenerate diagrams, which are self explaining.

For example, the forking diagram $\xleftarrow{n,a} \cdot \xrightarrow{i\mathcal{S},llet} \rightsquigarrow \xrightarrow{i\mathcal{S},llet} \cdot \xleftarrow{n,a}$ is represented as

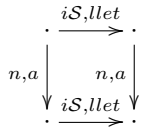


The commuting diagram $\xrightarrow{i\mathcal{S},llet} \cdot \xrightarrow{n,a} \rightsquigarrow \xrightarrow{n,a} \cdot \xrightarrow{i\mathcal{S},llet}$ is represented as



The solid arrows represent given reductions and dashed arrows represent existential reductions. A common representation is without the dashed arrows, where the interpretation depends on whether the diagrams is interpreted as a forking or a commuting diagram.

Note that the selection of the reduction label is considered to occur outside the transformation rule, i.e. if $\xrightarrow{n,a}$ occurs on both sides of the transformation rule the label a is considered to be the same on both sides.



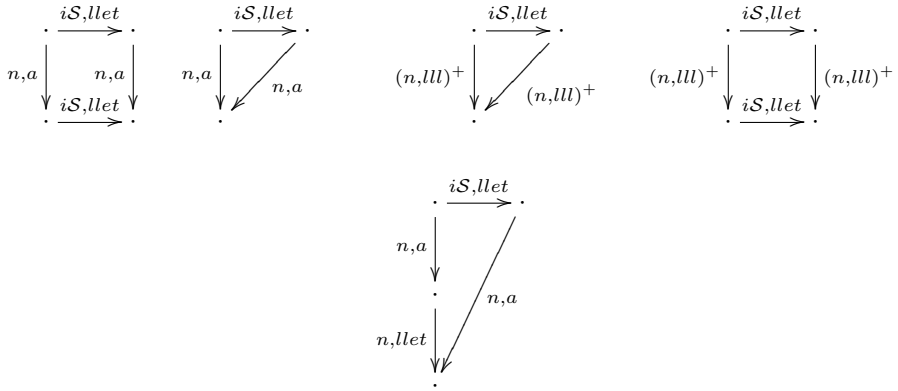
B.3 Correctness of (llet) and (cp)

We prove correctness of the reductions (llet) and (cp) by computing the required forking and commuting diagrams, and then give a sketch of an inductive proof for constructing normal order reductions.

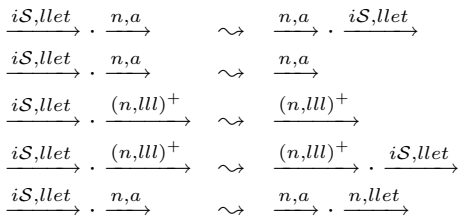
Correctness of (llet)

For the reduction (llet), we use the reductions in $\mathcal{S}$-contexts instead of reduction contexts, since they are more general and cover all reduction contexts.

Lemma B.5
A complete set of forking diagrams and a complete set of commuting diagrams for $(i\mathcal{S}, llet)$ can be read off of the following graphical diagrams:



The corresponding complete set of commuting diagrams is:



The corresponding complete set of forking diagrams is:

$$
\begin{array}{lcl}
\xleftarrow{n,a} \cdot \xrightarrow{i\mathcal{S},llet} & \rightsquigarrow & \xrightarrow{i\mathcal{S},llet} \cdot \xleftarrow{n,a} \\
\xleftarrow{n,a} \cdot \xrightarrow{i\mathcal{S},llet} & \rightsquigarrow & \xleftarrow{n,a} \\
\xleftarrow{(n,lll)^+} \cdot \xrightarrow{i\mathcal{S},llet} & \rightsquigarrow & \xleftarrow{(n,lll)^+} \\
\xleftarrow{(n,lll)^+} \cdot \xrightarrow{i\mathcal{S},llet} & \rightsquigarrow & \xrightarrow{i\mathcal{S},llet} \cdot \xleftarrow{(n,lll)^+} \\
\xleftarrow{n,llet} \cdot \xleftarrow{n,a} \cdot \xrightarrow{i\mathcal{S},llet} & \rightsquigarrow & \xleftarrow{n,a}
\end{array}
$$

Proof
Diagram 1 covers the cases where the (iS, llet) and (n,a)-reductions commute. Diagram 2 covers the case of removed expressions in a (case)-reduction or a (seq)-reduction. Lemma 2.5 describes the same cases as necessary for diagrams 3 and 4. Diagram 5 is the case where in diagram 1 the closing (llet) is turned into a normal order reduction. The typical case is ($\texttt{letrec}\ x = (\texttt{letrec}\ Env\ \texttt{in}\ s)\ \texttt{in}\ \texttt{seq}\ \texttt{True}\ x$). □

Lemma B.6
If $s \xrightarrow{i,lll} t$, then s is a WHNF iff t is a WHNF.

Proposition B.7
If $s \xrightarrow{llet} t$, then $s \sim_c t$.

Proof
By the context lemma A.1, it is sufficient to prove $R[s]{\Downarrow} \Leftrightarrow R[t]{\Downarrow}$ for all reduction contexts R. If $R[s] \xrightarrow{n,llet} R[t]$, then this is trivial. In the case $R[s] \xrightarrow{i\mathcal{S},llet} R[t]$, we use the complete sets of diagrams in Lemma B.5 to show that from an evaluation of $R[s]$, we can construct an evaluation of $R[t]$, and vice versa.

1. If $R[s]{\Downarrow}$, then by induction on the length of the normal order reduction sequence *Red* of $R[s]$, there is also an evaluation of $R[t]$: We use the fact that if $s \xrightarrow{i\mathcal{S},llet} t$, then also $R[s] \xrightarrow{i\mathcal{S},llet} R[t]$, since reduction contexts are also surface contexts and the combination of surface contexts again gives a surface context.
   In the base case we use Lemma B.6. If *Red* is not trivial, then the complete set of forking diagrams in Lemma B.5 provides all cases. Let $Red = R[s] \xrightarrow{n} s' \cdot Red'$. Diagrams 2,3,5 directly construct a terminating normal order reduction for $R[t]$. For diagrams 1 and 4, the induction hypothesis can be applied to $s' \xrightarrow{i\mathcal{S},llet} t'$ with $R[t] \xrightarrow{n,+} t'$, and we obtain a terminating normal order reduction for $R[t]$.
2. If $R[t]{\Downarrow}$, then we use similar methods. We apply induction on the number of normal order reduction steps of $R[t]$ to a WHNF using the complete set of commuting diagrams in Lemma B.5. In the base case we use Lemma B.6. □

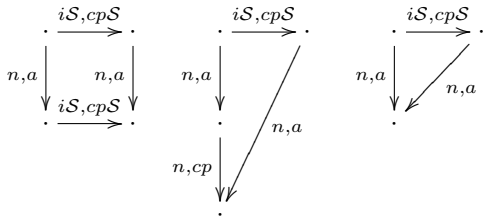
Correctness of (cp)

To show that the (cp)-reduction is correct as a program transformation, we have to split the reduction into two different reductions, depending on the position of the target variable.

(cpS) = (cp) where the position of the replaced variable is in a surface context.

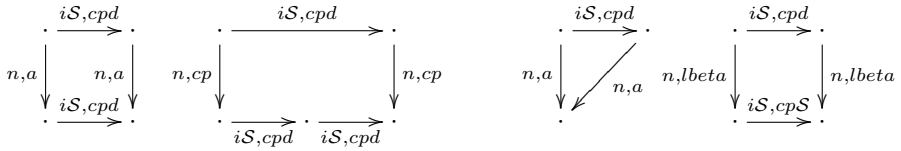
(cpd) = (cp) where the position of the replaced variable is not in a surface context.

Lemma B.8
A complete set of forking diagrams and a complete set of commuting diagrams for $\xrightarrow{i\mathcal{S},cp\mathcal{S}}$ can be read off of the following graphical diagrams:



Proof
By case analysis. □

Lemma B.9
A complete set of forking diagrams and a complete set of commuting diagrams for $\xrightarrow{i\mathcal{S},cpd}$ can be read off of the following graphical diagrams:



Proof
By case analysis. □

Lemma B.10
If $s \xrightarrow{i\mathcal{S},cp} t$, then s is a WHNF iff t is a WHNF.

Proposition B.11
If $s \xrightarrow{cp} t$, then $s \sim_c t$.

Proof
It is sufficient to prove $R[s]{\Downarrow} \Leftrightarrow R[t]{\Downarrow}$ for all reduction contexts. If $R[s] \xrightarrow{n,cp} R[t]$, then this is trivial. In the case $R[s] \xrightarrow{i\mathcal{S},cp} R[t]$, we use the diagrams for (cp), i.e., for (cpd) and (cpS).

1. Assume $R[t]{\Downarrow}$. The method is to transform the reduction $R[s] \xrightarrow{i\mathcal{S},cp} R[t] \cdot RED$, where RED is an evaluation, into an evaluation of $R[s]$ using transformations that correspond to the complete sets of commuting diagrams in Lemmas B.9 and B.8.

We have to show that the transformation terminates with an evaluation, where the local effect of the transformation is to shift $(i\mathcal{S}, cpd)$ and $(i\mathcal{S}, cp\mathcal{S})$ to the right. We define a well-founded measure for reduction sequences *Red* where $\xrightarrow{(i\mathcal{S},cpd)}$, $\xrightarrow{(i\mathcal{S},cp\mathcal{S})}$ and normal order reductions are mixed.
A single $(i\mathcal{S}, cpd)$ or $(i\mathcal{S}, cp\mathcal{S})$ in *Red* has as measure the pair consisting of

(a) the number of (n,lbeta)-reductions to the right of it;
(b) the number of all normal-order and $(i\mathcal{S}, cp\mathcal{S})$-reductions to the right of it before the next (n,lbeta)-reduction;

The pairs are ordered lexicographically. The measure μ of *Red* is the multiset of the pairs for all $i\mathcal{S}$-reductions, ordered by the derived multiset-ordering. Every transformation rule of the commuting diagrams for $(i\mathcal{S}, cpd)$ and $(i\mathcal{S}, cp\mathcal{S})$ strictly decreases the measure μ. That the measure is decreased must also be checked for $(i\mathcal{S}, cpd)$ and $(i\mathcal{S}, cp\mathcal{S})$- reductions that are to the left of the modification of the reduction sequence, i.e., that are not directly involved in the transformation.

(a) Diagram cpS-1: If $a =$ (lbeta), then it strictly decreases the first component for the shifted $(i\mathcal{S}, cp\mathcal{S})$-reduction, and it does not increase the pairs for other $(i\mathcal{S}, cpd)$ or $(i\mathcal{S}, cp\mathcal{S})$-reductions. If $a \neq$ (lbeta), then the the second component of the shifted $(i\mathcal{S}, cp\mathcal{S})$-reduction is strictly decreased.
(b) Diagram cpS-2: One pair is removed from the multiset, and the other pairs remain unchanged.
(c) Diagram cpS-3: One pair is removed, and the other pairs are not increased.
(d) Diagram cpd-1: Similar to diagram cpS-1.
(e) Diagram cpd-2: One pair is replaced by two pairs that have a strictly smaller second component, hence the measure is strictly decreased.
(f) Diagram cpd-3: See cpS-3.
(g) Diagram cpd-4: A pair is replaced by a pair with a strictly smaller first component.

Since a diagram is applicable whenever there is a (cpd) or (cpS) reduction for a non-WHNF term, the transformation terminates with a normal order reduction sequence followed by a last $(i\mathcal{S}, cp)$-reduction.
For the base case use Lemma B.10.

2. If $R[s]\Downarrow$, then we have to show that $R[t]\Downarrow$. We prove by induction on the length of an evaluation of $R[s]$ that $R[t]$ also has an evaluation of an equal or shorter length. This is easy, checking the complete set of forking diagrams in Lemmas B.9 and B.8. For the base case use Lemma B.10.

Now we can conclude by applying the context lemma for the two directions that $s \sim_c t$. □

B.4 Correctness of (gc), (cpx), (cpax), (abs), (xch) and (cpcx)

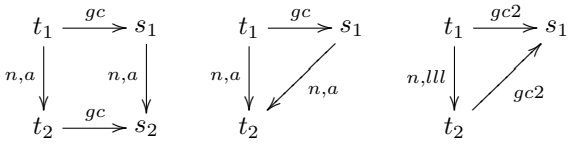
We show in this subsection that the extra transformations (gc), (cpx), (cpax), (abs), (xch) and (cpcx) are correct program transformations in the calculus LR. This

will lead to a proof that (case) is a correct program transformation in the next subsection.

In this subsection we extend the notion of complete sets of commuting and complete sets of forking diagrams slightly by allowing the extra transformations in the place of the internal transformations.

Correctness of (gc)

Lemma B.12
A complete set of commuting and a complete set of forking diagrams for $(\mathcal{S}, gc)$ can be read off the following set of graphical diagrams:



Proof
This follows by a case analysis. Diagram 2 occurs if the (gc)-redex is in an alternative removed by a case or in the term removed by a (seq). Diagram 3 occurs, e.g. in the case (`seq` (`letrec` *Env* `in` t_1) t_2) if (gc2) removes the environment *Env*, and in similar cases. A further example for the third case is

$$\begin{array}{ll} & R[((\texttt{letrec}\ Env_1\ \texttt{in}\ t_1)\ x)] \\ \xrightarrow{n,lapp} & R[(\texttt{letrec}\ Env_1\ \texttt{in}\ (t_1\ x))] \\ \hline \xrightarrow{gc} & R[(t_1\ x)] \end{array}$$

The following nontrivial overlapping results in a diagram of type 1.

$$\begin{array}{ll} & (\texttt{letrec}\ Env_1\ \texttt{in}\ ((\texttt{letrec}\ Env_2\ \texttt{in}\ s)\ t)) \\ \xrightarrow{gc} & ((\texttt{letrec}\ Env_2\ \texttt{in}\ s)\ t) \\ \xrightarrow{n,lapp} & (\texttt{letrec}\ Env_2\ \texttt{in}\ (s\ t)) \\ \hline \xrightarrow{n,lapp} & (\texttt{letrec}\ Env_1\ \texttt{in}\ (\texttt{letrec}\ Env_2\ \texttt{in}\ (s\ t))) \\ \xrightarrow{gc} & (\texttt{letrec}\ Env_2\ \texttt{in}\ s\ t) \qquad \square \end{array}$$

Lemma B.13
Let t, t' be expressions and $t \xrightarrow{gc} t'$. Then

- If t is a WHNF, then t' is a WHNF.
- If t' is a WHNF and t is not a WHNF, then $t \xrightarrow{n,llet} t'$; or $t \xrightarrow{[],n,llet} t'' \xrightarrow{gc2} t'$, and t'' is a WHNF.

Proposition B.14
Let t be an expression. If $t \xrightarrow{gc} t'$, then $t \sim_c t'$.

Proof

Using the context lemma and the same technique as in the proof of Proposition B.11, we have only to ensure that transforming an evaluation of $R[t]$ using the diagrams in Lemma B.12 to an evaluation of $R[t']$, and vice versa, always successfully terminates.

The measure for both directions is the length of an evaluation, where the base case requires Lemma B.13. In constructing an evaluation of $R[t']$ from a reduction $R[t] \xrightarrow{gc} R[t'] \xrightarrow{n,*} t_0$, Proposition 2.7 shows that there are only finitely many repeated applications of diagram 3. □

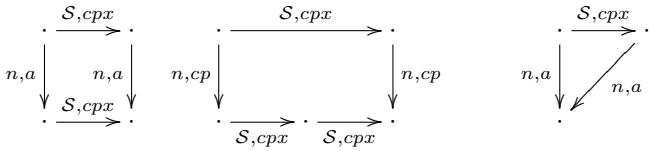
Diagrams and Properties of (cpx) and (cpax)

We first give diagrams of several transformations and then prove their correctness in one proof.

Note that the transformation $\xrightarrow{\mathcal{R},cpx}$ may not terminate:
$\texttt{letrec}\ x = y, y = x\ \texttt{in}\ C[x] \xrightarrow{\mathcal{R},cpx} \texttt{letrec}\ x = y, y = x\ \texttt{in}\ C[y] \xrightarrow{\mathcal{R},cpx} \texttt{letrec}\ x = y, y = x\ \texttt{in}\ C[x]$.

A further example for non-termination is: $\texttt{letrec}\ x = y, y = x, z = x\ \texttt{in}\ t \xrightarrow{\mathcal{R},cpx} \texttt{letrec}\ x = y, y = x, z = y\ \texttt{in}\ t \xrightarrow{\mathcal{R},cpx} \texttt{letrec}\ x = y, y = x, z = x\ \texttt{in}\ t$.

Lemma B.15
A complete set of forking and a complete set of commuting diagrams for $\xrightarrow{\mathcal{S},cpx}$ can be read off the following graphical diagrams:



Proof
The second case happens if the target of the (cpx)-transformation is in the copied abstraction of the (cp). The third case may happen if the transformation is a (case), (cp) or (seq). An example for the last case is

$$\begin{array}{ll} & \texttt{letrec}\ x = s, y = x\ \texttt{in}\ C[y] \\ \xrightarrow{\mathcal{S},cpx} & \texttt{letrec}\ x = s, y = x\ \texttt{in}\ C[x] \\ \xrightarrow{n,cp} & \texttt{letrec}\ x = s, y = x\ \texttt{in}\ C[s] \\ \hline \xrightarrow{n,cp} & \texttt{letrec}\ x = s, y = x\ \texttt{in}\ C[s] \end{array}$$ □

Lemma B.16
If $s \xrightarrow{\mathcal{S},cpx} t$, then s is a WHNF iff t is a WHNF.

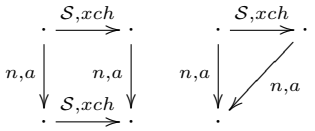
Proposition B.17
The transformation (cpax) is terminating.

Proof
Every (cpax) transformation strictly decreases the number of let-bound variables that have occurrences in the expression. □

Diagrams for (xch)

Lemma B.18

A complete set of forking and a complete set of commuting diagrams for $\xrightarrow{\mathcal{S},xch}$ can be read off the following graphical diagrams:



Proof

It is easy to verify that this holds for the different kinds of reductions. Only for (case) and a specific type of interference we show the concrete transformation:

$$
\begin{array}{ll}
 & (\texttt{letrec}\ x = c\ t, y = x\ \texttt{in}\ \texttt{case}_T\ x\ ((c\ u) \to r)) \\
\xrightarrow{xch} & (\texttt{letrec}\ y = c\ t, x = y\ \texttt{in}\ \texttt{case}_T\ x\ ((c\ u) \to r)) \\
\xrightarrow{n,case} & (\texttt{letrec}\ y = c\ z, z = t, x = y\ \texttt{in}\ (\texttt{letrec}\ u = z\ \texttt{in}\ r)) \\
\hline
\xrightarrow{n,case} & (\texttt{letrec}\ x = c\ z, z = t, y = x\ \texttt{in}\ (\texttt{letrec}\ u = z\ \texttt{in}\ r)) \\
\xrightarrow{xch} & (\texttt{letrec}\ y = c\ z, z = t, x = y\ \texttt{in}\ (\texttt{letrec}\ u = z\ \texttt{in}\ r))
\end{array}
\qquad \square
$$

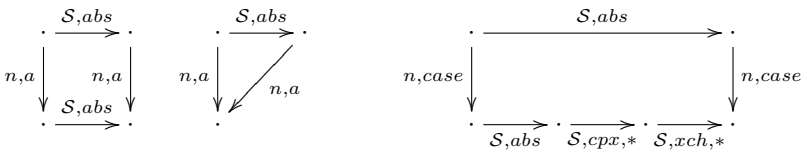
Lemma B.19

If $s \xrightarrow{\mathcal{S},xch} t$, then s is a WHNF iff t is a WHNF.

Diagrams for (abs)

Lemma B.20

A complete set of commuting and a complete set of forking diagrams for $\xrightarrow{\mathcal{S},abs}$ can be read off the following diagrams:



Proof

Instead of a complete proof, we only show the typical hard case:

$$
\begin{array}{ll}
 & (\texttt{letrec}\ x = c\ t\ \texttt{in}\ \texttt{case}_T\ x\ (c\ y \to s)) \\
\xrightarrow{abs} & (\texttt{letrec}\ x = c\ z, z = t\ \texttt{in}\ \texttt{case}_T\ x\ ((c\ y) \to s)) \\
\xrightarrow{n,case} & (\texttt{letrec}\ x = c\ u, u = z, z = t\ \texttt{in}\ (\texttt{letrec}\ y = u\ \texttt{in}\ s))
\end{array}
$$

$$
\begin{array}{ll}
\xrightarrow{n,case} & (\texttt{letrec}\ x = c\ u, u = t\ \texttt{in}\ (\texttt{letrec}\ y = u\ \texttt{in}\ s)) \\
\xrightarrow{abs} & (\texttt{letrec}\ x = c\ z, z = u, u = t\ \texttt{in}\ (\texttt{letrec}\ y = u\ \texttt{in}\ s)) \\
\xrightarrow{cpx,*} & (\texttt{letrec}\ x = c\ u, z = u, u = t\ \texttt{in}\ (\texttt{letrec}\ y = u\ \texttt{in}\ s)) \\
\xrightarrow{xch,*} & (\texttt{letrec}\ x = c\ u, u = z, z = t\ \texttt{in}\ (\texttt{letrec}\ y = u\ \texttt{in}\ s))
\end{array}
$$

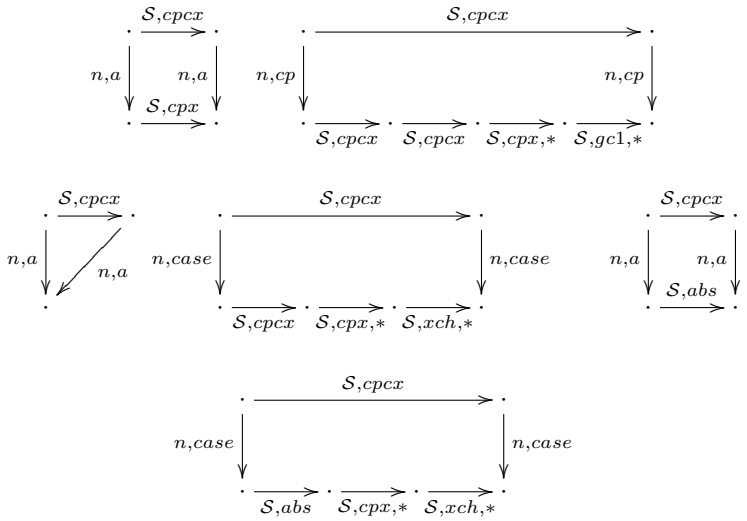
The second diagram covers the case where the (abs)-redex is removed by a (case), or (seq). □

Lemma B.21
If $s \xrightarrow{\mathcal{S},abs} t$, then s is a WHNF iff t is a WHNF.

Diagrams for (cpcx)

Note that there are infinite transformation sequences using only (cpcx):
$(\texttt{letrec}\ x = c\ x\ \texttt{in}\ x) \xrightarrow{cpcx} (\texttt{letrec}\ x = c\ x_1, x_1 = x\ \texttt{in}\ c\ x_1) \xrightarrow{cpcx} (\texttt{letrec}\ x = c\ x_2, x_2 = x_1, x_1 = x\ \texttt{in}\ c\ (c\ x_2)) \ldots$

Lemma B.22
A complete set of commuting and a complete set of forking diagrams for $\xrightarrow{\mathcal{S},cpcx}$ can be read off the following diagrams:



where a in the 5^{th} diagram may be (case) or (seq).

Proof

Instead of a complete proof, we only show the typical cases:

$$
\begin{array}{ll}
 & (\texttt{letrec}\ x = c\ t, y = \lambda u.C[x]\ \texttt{in}\ y) \\
\xrightarrow{\mathcal{S},cpcx} & (\texttt{letrec}\ x = c\ z, z = t, y = \lambda u.C[c\ z]\ \texttt{in}\ y) \\
\xrightarrow{n,cp} & (\texttt{letrec}\ x = c\ z, z = t, y = \lambda u.C[c\ z]\ \texttt{in}\ \lambda u'.C'[c\ z])
\end{array}
$$

$$
\begin{array}{ll}
\xrightarrow{n,cp} & (\texttt{letrec}\ x = c\ t, y = \lambda u.C[x]\ \texttt{in}\ \lambda u'.C'[x]) \\
\xrightarrow{cpcx} & (\texttt{letrec}\ x = c\ z, z = t, y = \lambda u.C[c\ z]\ \texttt{in}\ \lambda u'.C'[x]) \\
\xrightarrow{cpcx} & (\texttt{letrec}\ x = c\ z', z' = z, z = t, y = \lambda u.C[c\ z]\ \texttt{in}\ \lambda u'.C'[c\ z']) \\
\xrightarrow{cpx} & (\texttt{letrec}\ x = c\ z', z' = z, z = t, y = \lambda u.C[c\ z]\ \texttt{in}\ \lambda u.C[c\ z]) \\
\xrightarrow{cpx} & (\texttt{letrec}\ x = c\ z, z' = z, z = t, y = \lambda u.C[c\ z]\ \texttt{in}\ \lambda u.C[c\ z]) \\
\xrightarrow{gc} & (\texttt{letrec}\ x = c\ z, z = t, y = \lambda u.C[c\ z]\ \texttt{in}\ \lambda u.C[c\ z])
\end{array}
$$

$$
\begin{array}{ll}
 & (\texttt{letrec}\ x = c\ t\ \texttt{in}\ \texttt{case}_T\ x\ (c\ y \to s)) \\
\xrightarrow{cpcx} & (\texttt{letrec}\ x = c\ z, z = t\ \texttt{in}\ \texttt{case}_T\ (c\ z)\ ((c\ y) \to s)) \\
\xrightarrow{n,case} & (\texttt{letrec}\ x = c\ z, z = t\ \texttt{in}\ (\texttt{letrec}\ y = z\ \texttt{in}\ s))
\end{array}
$$

$$
\begin{array}{ll}
\xrightarrow{n,case} & (\texttt{letrec}\ x = c\ z, z = t\ \texttt{in}\ (\texttt{letrec}\ y = z\ \texttt{in}\ s))
\end{array}
$$

In the following example we use a multicontext $C[.,.]$ that may have different holes, every hole is mentioned as an argument.

$$
\begin{array}{ll}
 & (\texttt{letrec}\ x = c\ t\ \texttt{in}\ C[\texttt{case}_T\ x\ (c\ y \to s), x]) \\
\xrightarrow{cpcx} & (\texttt{letrec}\ x = c\ z, z = t\ \texttt{in}\ C[\texttt{case}_T\ x\ (c\ y \to s), c\ z]) \\
\xrightarrow{n,case} & (\texttt{letrec}\ x = c\ z', z' = z, z = t\ \texttt{in}\ C[(\texttt{letrec}\ y = z'\ \texttt{in}\ s), c\ z])
\end{array}
$$

$$
\begin{array}{ll}
\xrightarrow{n,case} & (\texttt{letrec}\ x = c\ z', z' = t\ \texttt{in}\ C[(\texttt{letrec}\ y = z'\ \texttt{in}\ s), x]) \\
\xrightarrow{cpcx} & (\texttt{letrec}\ x = c\ z, z = z', z' = t\ \texttt{in}\ C[(\texttt{letrec}\ y = z'\ \texttt{in}\ s), c\ z]) \\
\xrightarrow{cpx} & (\texttt{letrec}\ x = c\ z', z = z', z' = t\ \texttt{in}\ C[(\texttt{letrec}\ y = z'\ \texttt{in}\ s), c\ z]) \\
\xrightarrow{xch} & (\texttt{letrec}\ x = c\ z', z' = z, z = t\ \texttt{in}\ C[(\texttt{letrec}\ y = z'\ \texttt{in}\ s), c\ z])
\end{array}
$$

$$
\begin{array}{ll}
 & (\texttt{letrec}\ x = c\ t\ \texttt{in}\ \texttt{seq}\ x\ r) \\
\xrightarrow{cpcx} & (\texttt{letrec}\ x = c\ z, z = t\ \texttt{in}\ \texttt{seq}\ (c\ z)\ r) \\
\xrightarrow{n,seq} & (\texttt{letrec}\ x = c\ z, z = t\ \texttt{in}\ r)
\end{array}
$$

$$
\begin{array}{ll}
\xrightarrow{n,seq} & (\texttt{letrec}\ x = c\ t\ \texttt{in}\ r) \\
\xrightarrow{abs} & (\texttt{letrec}\ x = c\ z, z = t\ \texttt{in}\ r)
\end{array}
$$

$$
\begin{array}{ll}
 & (\texttt{letrec}\ x = c\ t\ \texttt{in}\ C[\texttt{case}_T\ x\ (c\ y \to s)\ (c' \ldots \to x)]) \\
\xrightarrow{cpcx} & (\texttt{letrec}\ x = c\ z, z = t\ \texttt{in}\ C[\texttt{case}_T\ x\ (c\ y \to s)\ (c' \ldots \to (cz))]) \\
\xrightarrow{n,case} & (\texttt{letrec}\ x = c\ z', z' = z, z = t\ \texttt{in}\ C[(\texttt{letrec}\ y = z'\ \texttt{in}\ s)])
\end{array}
$$

$$
\begin{array}{ll}
\xrightarrow{n,case} & (\texttt{letrec}\ x = c\ z', z' = t\ \texttt{in}\ C[(\texttt{letrec}\ y = z'\ \texttt{in}\ s)]) \\
\xrightarrow{abs} & (\texttt{letrec}\ x = c\ z, z = z', z' = t\ \texttt{in}\ C[(\texttt{letrec}\ y = z'\ \texttt{in}\ s)]) \\
\xrightarrow{cpx} & (\texttt{letrec}\ x = c\ z', z = z', z' = t\ \texttt{in}\ C[(\texttt{letrec}\ y = z'\ \texttt{in}\ s)]) \\
\xrightarrow{xch} & (\texttt{letrec}\ x = c\ z', z' = z, z = t\ \texttt{in}\ C[(\texttt{letrec}\ y = z'\ \texttt{in}\ s)])
\end{array}
\qquad \square
$$

Lemma B.23
If $s \xrightarrow{\mathcal{S},cpcx} t$, then s is a WHNF iff t is a WHNF.

B.5 Correctness of (cpx), (cpax), (cpcx), (xch), and (abs)

Theorem B.24
The transformations (cpcx), (cpx), (cpax), (abs), (xch) and (gc) maintain contextual equivalence. I.e. whenever $s \xrightarrow{a} t$, with $a \in \{\text{cpcx}, \text{cpx}, \text{cpax}, \text{abs}, \text{xch}, \text{gc}\}$, then $s \sim_c t$.

Proof
For (gc) this follows from B.14, and the correctness of (cpax) will follow from correctness of (cpx), proved below.

We will show the correctness of the reductions in $M := \{\text{cpcx}, \text{cpx}, \text{abs}, \text{xch}, \text{gc1}\}$ using an induction argument. Using the context lemma it is sufficient to show that for every reduction context R, and terms s, t, $a \in M$: with $s \xrightarrow{a} t$, the relation $R[s] \sim_c R[t]$ holds.
We show by induction on the length of an evaluation that if $R[s]$ has an evaluation of length m, then $R[t]$ has an evaluation of length at most m, and if $R[t]$ has an evaluation of length m, then $R[s]$ has an evaluation of length at most m. The base case is shown in Lemmas B.23, B.16, B.21, B.19, and B.13.

In the following induction argument, we have to take into account all diagrams in Lemmas B.22, B.15, B.20, B.18, and B.12, where in the latter lemma for (gc) only the first two are relevant here.

Let $R[s]$ have an evaluation of length m, let $R[s] \xrightarrow{a} R[t]$ with $a \in M$, and $R[s] \xrightarrow{n} s'$. Using the forking diagrams, we see that $R[t] \xrightarrow{n} t'$, and $s' \xrightarrow{*,M} t'$. The induction hypothesis can be applied to s', since the length of its evaluation is $\leq m - 1$, and hence to all terms in the sequence $s' \xrightarrow{*,M} t'$, which shows that an evaluation of t' is of length at most $m - 1$. We obtain that $R[t]$ has an evaluation of length at most m.

For the other direction, let $R[s]$ have an evaluation of length m, let $R[s] \xrightarrow{a} R[t]$ with $a \in M$, and $R[s] \xrightarrow{n} s'$. Using the commuting diagrams, we see that $R[t] \xrightarrow{n} t'$, and $s' \xrightarrow{*,M} t'$. Since the evaluation of t' is of length at most $m - 1$, hence we can apply the induction hypothesis, to show that s' has an evaluation of length at most

$m-1$. The reduction $R[s] \xrightarrow{n} s'$ shows that $R[s]$ has an evaluation of length at most m.

Now we can apply the context lemma and conclude that $s \sim_c t$. □

B.6 Correctness of (case) and (seq)

Proposition B.25
The reductions (case-in) and (case-e) are correct program transformations.

Proof
Proposition B.2 shows that (case-c) is a correct program transformation. From Theorem B.24 we obtain that (cpcx) and (cpx) are correct program transformations. We show by induction that a (case-e) and (case-in)-reduction is correct by using the correctness of the transformations (cpcx), (case-c) and (cpx). The induction is on the length of the variable chain in the reduction (case-in) (or (case-e), respectively). We give the proof only for (case-in), the other is a copy of this proof.
For the base case the (case-in) reduction can also be performed by the sequence of reductions: $\xrightarrow{cpcx} \cdot \xrightarrow{case-c}$

$$\begin{array}{ll} & (\texttt{letrec}\ x = c\ t, Env\ \texttt{in}\ C[\texttt{case}_t\ x\ (c\ z \to s)\ alts]) \\ \xrightarrow{cpcx} & (\texttt{letrec}\ x = c\ y, y = t, Env\ \texttt{in}\ C[\texttt{case}_T\ (c\ y)\ (c\ z \to s)\ alts]) \\ \xrightarrow{case-c} & (\texttt{letrec}\ x = c\ y, y = t, Env\ \texttt{in}\ C[(\texttt{letrec}\ z = y\ \texttt{in}\ s)]) \end{array}$$

For the induction we replace a (case-in) reduction operating on a chain $\{x_i = x_{i-1}\}_{i=2}^m$ with the sequence $\xrightarrow{cpcx} \cdot \xrightarrow{case-in} \cdot \xrightarrow{cpx^n} \cdot \xleftarrow{cpx^n} \cdot \xleftarrow{cpcx}$, where n is the arity of the constructor and the (case-in) reduction operates on the chain $\{x_i = x_{i-1}\}_{i=3}^m$:

$$\begin{array}{ll} & (\texttt{letrec}\ x_1 = c\ \overrightarrow{t}, \{x_i = x_{i-1}\}_{i=2}^m, Env\ \texttt{in}\ C[\texttt{case}_T\ x_m\ (c\ \overrightarrow{z} \to s)\ alts]) \\ \xrightarrow{cpcx} & \texttt{letrec}\ x_1 = c\ \overrightarrow{y}, \{y_i = t_i\}_{i=1}^n, x_2 = c\ \overrightarrow{y}, \{x_i = x_{i-1}\}_{i=3}^m, Env \\ & \texttt{in}\ C[\texttt{case}_T\ x_1\ (c\ \overrightarrow{z} \to s)\ alts] \\ \xrightarrow{case-in} & \texttt{letrec}\ x_1 = c\ \overrightarrow{y}, \{y_i = t_i\}_{i=1}^n, x_2 = c\ \overrightarrow{y'}, \{y'_i = y_i\}_{i=1}^n, \{x_i = x_{i-1}\}_{i=3}^m, Env \\ & \texttt{in}\ C[(\texttt{letrec}\ \{z_i = y'_i\}_{i=1}^n\ \texttt{in}\ s)] \\ \xrightarrow{cpx^n} & \texttt{letrec}\ x_1 = c\ \overrightarrow{y}, \{y_i = t_i\}_{i=1}^n, x_2 = c\ \overrightarrow{y'}, \{y'_i = y_i\}_{i=1}^n, \{x_i = x_{i-1}\}_{i=3}^m, Env \\ & \texttt{in}\ C[(\texttt{letrec}\ \{z_i = y_i\}_{i=1}^n\ \texttt{in}\ s)] \\ \xleftarrow{cpx^n} & \texttt{letrec}\ x_1 = c\ \overrightarrow{y'}, \{y_i = t_i\}_{i=1}^n, x_2 = c\ \overrightarrow{y'}, \{y'_i = y_i\}_{i=1}^n, \{x_i = x_{i-1}\}_{i=3}^m, Env \\ & \texttt{in}\ C[(\texttt{letrec}\ \{z_i = y_i\}_{i=1}^n\ \texttt{in}\ s)] \\ \xleftarrow{cpcx} & \texttt{letrec}\ x_1 = c\ \overrightarrow{y}, \{y_i = t_i\}_{i=1}^n, \{x_i = x_{i-1}\}_{i=2}^m, Env \\ & \texttt{in}\ C[(\texttt{letrec}\ \{z_i = y_i\}_{i=1}^n\ \texttt{in}\ s)] \end{array}$$ □

Proposition B.26
If $s \xrightarrow{seq} t$, then $s \sim_c t$.

Proof
Let s, t be expressions with $s \xrightarrow{C,seq} t$. If the (seq)-reduction is a (seq-c) reduction, then the claim follows from Proposition B.2. Otherwise, we can transform s into t as follows: if the value of (seq)-reduction is an abstraction then $s \xrightarrow{C,cp} \xrightarrow{C,seq-c} t$. If

the value is an constructor application then $s \xrightarrow{C,cpcx} \xrightarrow{C,seq} \xleftarrow{C,abs} t$. Now the claim follows from Proposition B.2, B.11 and Theorem B.24. □

Proposition B.27
The reduction (case) is a correct program transformation.

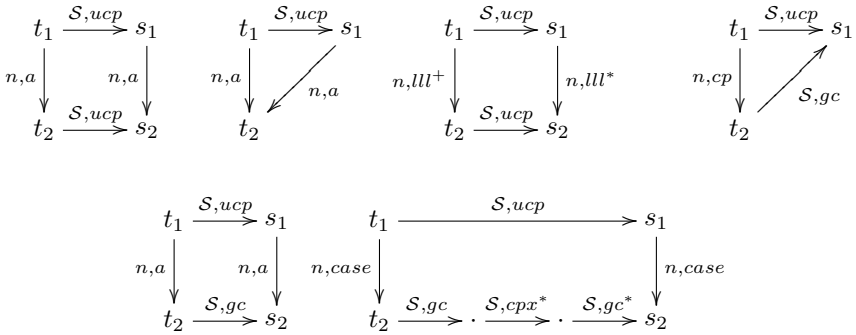
Proof
Follows from Proposition B.25 and B.2. □

B.7 Correctness of (ucp), (abse) and (lwas)

Correctness of (ucp)

A difference between (ucp) and (cp) is that (ucp) can be applied even if the expression bound to a variable is not an abstraction.

Lemma B.28
The complete sets of forking and of commuting diagrams for $\xrightarrow{\mathcal{S},ucp}$ can be read off of the following graphical diagrams:



Where $a \in \{\text{seq,case}\}$ in the fifth diagram.

Proof
We show the typical overlappings.

$$
\begin{array}{ll}
 & (\texttt{letrec}\ x = (\texttt{letrec}\ y = t\ \texttt{in}\ s), Env\ \texttt{in}\ (x\ z)) \\
\xrightarrow{ucp} & (\texttt{letrec}\ Env\ \texttt{in}\ ((\texttt{letrec}\ y = t\ \texttt{in}\ s)\ z)) \\
\xrightarrow{n,lapp} & (\texttt{letrec}\ Env\ \texttt{in}\ (\texttt{letrec}\ y = t\ \texttt{in}\ (s\ z))) \\
\xrightarrow{n,llet} & (\texttt{letrec}\ Env, y = t\ \texttt{in}\ (s\ z))
\end{array}
$$

$$
\begin{array}{ll}
\xrightarrow{n,llet} & (\texttt{letrec}\ x = s, y = t, Env\ \texttt{in}\ (x\ z)) \\
\xrightarrow{ucp} & (\texttt{letrec}\ y = t, Env\ \texttt{in}\ (s\ z))
\end{array}
$$

$$
\begin{array}{ll}
 & (\texttt{letrec}\ x = (\texttt{letrec}\ y = t_y\ \texttt{in}\ t_x), z = R'[x], Env\ \texttt{in}\ R[z]) \\
\xrightarrow{ucp} & (\texttt{letrec}\ z = R'[(\texttt{letrec}\ y = t_y\ \texttt{in}\ t_x)], Env\ \texttt{in}\ R[z]) \\
\xrightarrow{n,llet,+} & (\texttt{letrec}\ z = R'[t_x], y = t_y, Env\ \texttt{in}\ R[z])
\end{array}
$$

$$
\begin{array}{ll}
\xrightarrow{n,llet,+} & (\texttt{letrec}\ x = t_x, y = t_y, z = R'[x], Env\ \texttt{in}\ R[z]) \\
\xrightarrow{ucp} & (\texttt{letrec}\ y = t_y, z = R'[t_x], Env\ \texttt{in}\ R[z])
\end{array}
$$

$$
\begin{array}{ll}
 & (\texttt{letrec}\ x = (\texttt{letrec}\ Env\ \texttt{in}\ v)\ \texttt{in}\ x) \\
\xrightarrow{ucp} & (\texttt{letrec}\ Env\ \texttt{in}\ v)
\end{array}
$$

$$
\begin{array}{ll}
\xrightarrow{n,llet} & (\texttt{letrec}\ x = v, Env\ \texttt{in}\ x) \\
\xrightarrow{ucp} & (\texttt{letrec}\ Env\ \texttt{in}\ v)
\end{array}
$$

$$
\begin{array}{ll}
 & (\texttt{letrec}\ x = s, Env\ \texttt{in}\ (x\ y)) \\
\xrightarrow{ucp} & (\texttt{letrec}\ Env\ \texttt{in}\ (s\ y))
\end{array}
$$

$$
\begin{array}{ll}
\xrightarrow{n,cp} & (\texttt{letrec}\ x = s, Env\ \texttt{in}\ (s\ y)) \\
\xrightarrow{gc} & (\texttt{letrec}\ Env\ \texttt{in}\ (s\ y))
\end{array}
$$

$$
\begin{array}{ll}
 & (\texttt{letrec}\ x = cs\ \texttt{in}\ (\texttt{seq}\ x\ r)) \\
\xrightarrow{ucp} & (\texttt{seq}\ (c\ s)\ r) \\
\xrightarrow{n,seq} & r
\end{array}
$$

$$
\begin{array}{ll}
\xrightarrow{n,seq} & (\texttt{letrec}\ x = cs\ \texttt{in}\ r) \\
\xrightarrow{gc} & r
\end{array}
$$

$$
\begin{array}{ll}
 & (\texttt{letrec}\ x = c\ s_i\ \texttt{in}\ (\texttt{case}_T\ x\ (c\ z_i \to r))) \\
\xrightarrow{ucp} & (\texttt{case}_T\ (c\ s_i)\ (c\ z_i \to r)) \\
\xrightarrow{n,case} & (\texttt{letrec}\ z_i = s_i\ \texttt{in}\ r)
\end{array}
$$

$$
\begin{array}{ll}
\xrightarrow{n,case} & (\texttt{letrec}\ x = c\ y_i, y_i = s_i\ \texttt{in}\ (\texttt{letrec}\ z_i = y_i\ \texttt{in}\ r)) \\
\xrightarrow{gc} & (\texttt{letrec}\ y_i = s\ \texttt{in}\ (\texttt{letrec}\ z_i = y_i\ \texttt{in}\ r)) \\
\xrightarrow{cpx,*} & (\texttt{letrec}\ y_i = s\ \texttt{in}\ (\texttt{letrec}\ z_i = y_i\ \texttt{in}\ r[y_i/z_i])) \\
\xrightarrow{gc,*} & (\texttt{letrec}\ y_i = s\ \texttt{in}\ r[y_i/z_i]) \qquad \square
\end{array}
$$

Lemma B.29
Let t, t' be expressions and $t \xrightarrow{ucp} t'$. Then

- If t is a WHNF, then t' is a WHNF.
- If t' is a WHNF, then there are the following cases:
  — t is a WHNF

— $t \xrightarrow{n,(lll\cup\ cp),*} t''$, where t'' is a WHNF

Proof
The first case is obvious.
In the second case there are the following possibilities:

- $t = (\texttt{letrec}\ x = \lambda v.r, Env\ \texttt{in}\ x) \xrightarrow{ucp} (\texttt{letrec}\ Env\ \texttt{in}\ \lambda v.r)$. In this case an (n,cp)-reduction is sufficient to transform t into WHNF.
- $t = (\texttt{letrec}\ x = t_0\ \texttt{in}\ x) \xrightarrow{ucp} t_0$, where t_0 is a WHNF. Then either an (n,cp)-reduction, or an (n,llet)-reduction, or an (n, llet) followed by an (n,cp)-reduction transform t into WHNF.
- $t = (\texttt{letrec}\ x = v\ \texttt{in}\ (\texttt{letrec}\ Env\ \texttt{in}\ x)) \xrightarrow{ucp} (\texttt{letrec}\ Env\ \texttt{in}\ v)$, where v is a value. Then an (n,cp)-reduction or an (n,llet)-reduction or an (n,llet)-reduction followed by an (n,cp)-reduction transform t into WHNF.
- $t = (\texttt{letrec}\ Env\ \texttt{in}\ (\texttt{letrec}\ x = v\ \texttt{in}\ x)) \xrightarrow{ucp} (\texttt{letrec}\ Env\ \texttt{in}\ v)$. Again an (n, llet)-reduction or an (n, llet)-reduction followed by an (n,cp)-reduction transforms t into WHNF. □

Proposition B.30
Let t be an expression. If $t \xrightarrow{ucp} t'$, then $t \sim_c t'$

Proof
Using the context lemma and the same technique as in the proof of Proposition B.11, we have only to ensure that transforming an evaluation of $R[t]$ to an evaluation of $R[t']$, and vice versa, really terminates.

- Let $R[t]\Downarrow$. We show that $R[t']\Downarrow$ by using the forking diagrams in Lemma B.28 to transform $\overline{Red}\cdot \xrightarrow{\mathcal{S},ucp}$into an evaluation of $R[t']$, where Red is an evaluation of $R[t]$. Here $\overline{Red}$ means an evaluation denoted as a right-to-left reduction. The rearrangements have only two possibilities: either the $\xrightarrow{\mathcal{S},ucp}$-transformation is shifted to the left, perhaps over several normal-order reductions, or the $\xrightarrow{\mathcal{S},ucp}$-transformation is first shifted to the left, and then eliminated, and replaced by a sequence $\xrightarrow{\mathcal{S},gc,*} \cdot \xrightarrow{\mathcal{S},cpx,*} \cdot \xrightarrow{\mathcal{S},gc,*}$; a normal-order reduction may also be eliminated.
  If $\xrightarrow{\mathcal{S},ucp}$ is the left most reduction, then it transforms a WHNF into a WHNF by Lemma B.29, and an evaluation is already constructed. In the elimination case, the reduction sequence is of the form $t'' \xleftarrow{n,*} t''' \xrightarrow{\mathcal{S},gc} \cdot \xrightarrow{\mathcal{S},cpx,*} \cdot \xrightarrow{\mathcal{S},gc,*} t''' \xleftarrow{n,*} R[t']$. In this case Theorem B.24 shows that there is an evaluation of t''', and hence also of $R[t']$.
- Let $R[t']\Downarrow$. We show that $R[t]\Downarrow$ by transforming a terminating reduction $\xrightarrow{\mathcal{S},ucp} \cdot Red$ into an evaluation of $R[t]$, where Red is an evaluation of $R[t']$. We shift the single $\xrightarrow{\mathcal{S},ucp}$-transformation to the right using the diagrams in Lemma B.28. This terminates, since the number of normal order reductions to the right of the reduction $\xrightarrow{\mathcal{S},ucp}$ strictly decreases. The reduction $\xrightarrow{\mathcal{S},ucp}$ may be eliminated during this shifting, or if it reaches the WHNF, then we apply

Lemma B.29 and replace the final reduction by $\xrightarrow{n,(lll\cup\ cp),*} t''$, where t'' is a WHNF. This leaves a reduction sequence starting with $R[t']$, ending in a WHNF which is a mixture of normal order reductions and $\xrightarrow{\mathcal{S},gc}$and $\xrightarrow{\mathcal{S},cpx}$-transformations. Now we can apply Proposition B.14 and Theorem B.24 to transform this mixed sequence into an evaluation. □

Correctness of (abse)

Proposition B.31
The transformation (abse) is a correct program transformation.

Proof
This follows from Proposition B.30 and Theorem B.24, since (abse) can be undone by several (ucp)-and (gc)-transformations. □

Correctness of (lwas)

In this subsection we show the correctness of the transformation *(lwas)*, which lifts letrec bindings over an $\mathcal{W}_{(1)}^{-}$ context.

Proposition B.32
The (lwas)-transformation is correct. I.e., if $s \xrightarrow{lwas} t$, then $s \sim_c t$.

Proof
The reduction sequence

$$\begin{array}{ll} & W_{(1)}^{-}[(\texttt{letrec}\ Env\ \texttt{in}\ t)] \\ \xleftarrow{ucp} & (\texttt{letrec}\ x = (\texttt{letrec}\ Env\ \texttt{in}\ t)\ \texttt{in}\ W_{(1)}^{-}[x]) \\ \xrightarrow{llet} & (\texttt{letrec}\ x = t, Env\ \texttt{in}\ W_{(1)}^{-}[x]) \\ \xrightarrow{ucp} & (\texttt{letrec}\ Env\ \texttt{in}\ W_{(1)}^{-}[t]) \end{array}$$

and the correctness of (lll) and (ucp), which are proved in Proposition B.7 and Proposition B.30 show that the proposition holds. □

B.8 Correctness of the Variants of (case)-Reductions

Lemma B.33
The following holds:

1. The transformation rule (cpcxnoa) is a correct program transformation. It can be simulated by (cpcx) with a subsequent $\xrightarrow{cpx,*} \cdot \xrightarrow{gc,*}$ transformation.
2. A (case-cx) transformation can be simulated by (case) and subsequent $\xrightarrow{cpx,*} \cdot \xrightarrow{gc,*}$ transformation.
3. Every (case-e) and (case-in)-reduction can be simulated by an (abs)-transformation followed by a (case-cx)-transformation.
4. The transformation rule (case-cx) is a correct program transformation.

Proof
Follows from the correctness of (case) (Proposition B.27), (cpx), (gc) and (abs) by Theorem B.24. □

B.9 Proofs of Theorem 2.4 and 2.9

We prove Theorem 2.4, here repeated as a theorem in the appendix:

Theorem B.34
All the reductions in the base calculus are correct as transformations. I.e. whenever $t \xrightarrow{a} t'$, with $a \in \{\text{cp, lll, case, seq, lbeta}\}$, then $t \sim_c t'$.

It follows from Propositions B.7, B.26, B.11, B.27 and B.2.

We prove Theorem 2.9, here repeated as a theorem in the appendix:

Theorem B.35
The transformations (ucp), (cpx), (cpax), (gc), (lwas), (cpcx), (abs), (abse) (xch), (cpcxnoa) and (case-cx) are correct as transformations. I.e. whenever $t \xrightarrow{a} t'$, with $a \in \{\text{ucp, cpx, cpax, gc, lwas, cpcx, abs, abse, xch, cpcxnoa, case-cx}\}$, then $t \sim_c t'$.

It follows from Theorem B.24, Propositions B.30, B.32, and Lemma B.33.

C Properties of Bot

In this section we show that all bot-terms, i.e., all terms t with $t{\Uparrow}{\Uparrow}$ are equivalent to Ω, and that Ω is the least element w.r.t. $\leq_c$.

Proposition C.1
Let t be an expression such that $t{\Uparrow}{\Uparrow}$ and let s be an arbitrary expression.
Then $t \leq_c s$.

Proof
Let t be a bot-term with $t{\Uparrow}{\Uparrow}$. The context lemma shows that it is sufficient to prove for all reduction contexts R and all terms s: $R[t]{\Downarrow} \Rightarrow R[s]{\Downarrow}$. We simply prove that $R[t]{\Downarrow}$ does not hold. Assume that there is an evaluation of $R[t]$. We prove by induction that this implies that t has an evaluation.
Let $t \xrightarrow{n,*} t_1$, such that t_1 is the first `letrec`-expression in the sequence. If $t \neq t_1$, we can use induction, since the normal order reductions of $R[t] \xrightarrow{n,*} R[t_1]$ are precisely the same reductions. This holds, since inserting the maximal weak reduction context of t into the reduction context R also yields a reduction context.

In the rest of the proof we assume that t is a `letrec`-expression.
By Lemma 2.5, if $t = (\texttt{letrec}\ E_t\ \texttt{in}\ t')$, the normal order reduction reduces $R[t] = R[(\texttt{letrec}\ E_t\ \texttt{in}\ t')]$ to $(\texttt{letrec}\ E_t, E_R\ \texttt{in}\ R'[t'])$ in several steps, where R' is a weak reduction context, E_t the environment that belongs to t, and E_R the environment part that is at the top level of R. If R is not a `letrec`-expression, then E_R is empty.

The correspondence between normal order reduction sequences of t and of $(\texttt{letrec}\ E_t, E_R\ \texttt{in}\ R'[t'])$ is as follows:

- If there is a (n,llet-in)-reduction $t = (\texttt{letrec}\ E_t\ \texttt{in}\ (\texttt{letrec}\ E_1\ \texttt{in}\ t'')) \xrightarrow{n} (\texttt{letrec}\ E_t, E_1\ \texttt{in}\ t'')$, then the corresponding normal order reduction of $(\texttt{letrec}\ E_t, E_R\ \texttt{in}\ R'[(\texttt{letrec}\ E_1\ \texttt{in}\ t'')])$ is a normal order (lll,*)-reduction, shifting te environment to the top (see Lemma 2.5) resulting in $(\texttt{letrec}\ E_t, E_1, E_R\ \texttt{in}\ R'[t''])$. With $E'_t = E_t \cup E_1$, the correspondence holds.

- If there is another reduction of t, then this is of the form $(\texttt{letrec}\ E_t\ \texttt{in}\ t') \xrightarrow{n} (\texttt{letrec}\ E'_t\ \texttt{in}\ t'')$. It is easy to see that $(\texttt{letrec}\ E_t, E_R\ \texttt{in}\ R'[t']) \xrightarrow{n} (\texttt{letrec}\ E'_t, E_R\ \texttt{in}\ R'[t''])$. The environment E_R is never involved, since we have assumed that $t{\Uparrow}{\Uparrow}$.

Summarizing, the number of normal order reduction steps of $R[t]$ correspond to the number of normal order reductions of t. The number of (lll)-reductions may vary, but the non-(lll)-reductions are the same. Hence, if $R[t]$ has an evaluation, then we also obtain an evaluation of t by the translation above. This is a contradiction.
We conclude that the term $R[t]$ cannot have a terminating normal order reduction.
□

Corollary C.2

1. If t_1, t_2 are expressions with $t_1{\Uparrow}{\Uparrow}$ and $t_2{\Uparrow}{\Uparrow}$, then $\Omega \sim_c t_1 \sim_c t_2$.
2. For all expressions s: $\Omega \leq_c s$.
3. $R[\Omega] \sim_c \Omega$.
4. If $t = R[s]$ is an expression and R a reduction context, then s is a strict subexpression of t.

Proof
The first two claims follow from Proposition C.1. Claim 3 and 4 follow using the arguments in the proof of Proposition C.1. □

C.1 Reduction Rules for Bot-Terms

Definition C.3
The reduction rules that treat the bot-term Ω are defined in figure C 1. Note that these reductions are permitted in all contexts. Let (case-bot-all) be the union of (case-bot-c), (case-bot-in) and (case-bot-e). Further, let (app-bot-all) be the union of (app-bot-c),(app-bot-in) and (app-bot-e).

Proposition C.4
If $t \to t'$ by a bot-reduction as defined in Figure C 1, then

- $t \sim_c t'$.
- If t is a closed concrete term with $t{\Downarrow}$, then $t'{\Downarrow}$ and $\mathrm{rl}\sharp\sharp(t) = \mathrm{rl}\sharp\sharp(t')$.

Proof
Contextual equivalence follows from Corollary C.2 for (beta-bot), (case-bot-all), (app-bot-all), (case-bot), (seq-bot) and (strict-bot). For the rules (cp-in-bot), (cp-e-bot), and (hole) other arguments are required. The context lemma shows the claim: If $t \to t'$, and we check the evaluations of $R[t]$ and $R[t']$, then they are synchronous, as long as neither Ω nor x is required. The values of x or Ω are required in t iff they are required in t'. In this case this term is in a reduction context. We already know that then the expression is contextually equivalent to Ω by Corollary C.2. Thus the context lemma A.1 shows contextual equivalence.

The claim on the lengths of reductions can be seen as follows. In an evaluation, the subterm Ω, the subterm x, or the untyped expressions cannot be "required" by any reduction, local evaluation, hence the lengths of the evaluations are the same.
□

(beta-bot)	$(\Omega\ y) \to \Omega$
(cp-in-bot)	$(\texttt{letrec}\ x = \Omega, Env\ \texttt{in}\ C[x]) \to (\texttt{letrec}\ x = \Omega, Env\ \texttt{in}\ C[\Omega])$
(cp-e-bot)	$(\texttt{letrec}\ x = \Omega, y = C[x], Env\ \texttt{in}\ r) \to (\texttt{letrec}\ x = \Omega, y = C[\Omega], Env\ \texttt{in}\ r)$
(hole)	$(\texttt{letrec}\ x = x, Env\ \texttt{in}\ r) \to (\texttt{letrec}\ x = \Omega, Env\ \texttt{in}\ r)$
(case-bot)	$(\texttt{case}_T\ \Omega\ \ldots((c_i\ y_1 \ldots y_n)\ \texttt{->}\ t)\ldots) \to \Omega$
(app-bot-c)	$((c\ \overrightarrow{t})\ r) \to \Omega$
(app-bot-in)	$(\texttt{letrec}\ y = (c\ \overrightarrow{t}), Env\ \texttt{in}\ C[(y\ t)]) \to (\texttt{letrec}\ y = (c\ \overrightarrow{t}), Env\ \texttt{in}\ C[\Omega])$
(app-bot-e)	$(\texttt{letrec}\ y = (c\ \overrightarrow{t}), x = C[(y\ t)], Env\ \texttt{in}\ t) \to (\texttt{letrec}\ y = (c\ \overrightarrow{t}), x = C[\Omega], Env\ \texttt{in}\ t)$
(case-bot-c)	$(\texttt{case}_T\ v\ alts) \to \Omega$ if v is an abstraction or its top-constructor does not belong to the type T
(case-bot-in)	$(\texttt{letrec}\ x = v, Env\ \texttt{in}\ C[\texttt{case}_T\ x\ alts]) \to (\texttt{letrec}\ x = v, Env\ \texttt{in}\ C[\Omega])$ if v is an abstraction or its top-constructor does not belong to the type T
(case-bot-e)	$(\texttt{letrec}\ x = v, y = C[\texttt{case}_T\ x\ alts], Env\ \texttt{in}\ t) \to (\texttt{letrec}\ x = v, y = C[\Omega], Env\ \texttt{in}\ t)$ if v is an abstraction or its top-constructor does not belong to the type T
(seq-bot)	$(\texttt{seq}\ \Omega\ t) \to \Omega$
(strict-bot)	$(f\ t_1\ \ldots t_{i-1}\ \Omega\ t_{i+1}\ \ldots\ t_n) \to \Omega$ if f is strict in its i^{th} argument for arity n

Fig. C 1. Reduction rules for bot-terms

D Strict Subexpressions

In this section we show that the strictness optimization is correct. I.e., if a function f is strict in its i^{th} argument for arity n, then in every expression $f\ t_1 \ldots\ t_n$, that is itself strict in the top-term, it is permitted to first locally evaluate the argument t_i. More generally, we prove that a strict subexpression s of t in a surface context can be reduced eagerly to WHNF. There are also some other lemmas on properties of strict subexpressions needed to prove correctness of copying parts of environments later.

In the following, when we speak of a strict subterm s of t, we always mean the subterm together with its position in t. We assume also that we can use labeled reduction freely in order to identify a subterm before and after a reduction. Therefore, we assume that a subterm s is implicitly labeled, that the reduction respects these labels and that labels can be identified in the reduct, unless the reduction is an (llet)-reduction that destroys the top level `letrec` of s, or unless s is eliminated.

Lemma D.1
Let s be a strict subterm of the expression t, where $t = S[s]$ and S is a surface context. Then for every evaluation of $t \xrightarrow{n,*} t_0$, there is an intermediate term $R[s]$,

such that $t \xrightarrow{n,*} R[s] \xrightarrow{n,*} t_0$, R is a reduction context, and $R[s]$ is the first term in this sequence where the subexpression s is in a reduction context.

Proof
Since S is a surface context, if the reduction is independent of s, then the term s is either removed by a normal order reduction step or it remains in a surface context, and in particular, the successor subterm of s is unique. Suppose there is an evaluation $t \xrightarrow{n,*} t_0$, where s is never in a reduction context. Then we can replace s by Ω and get a corresponding evaluation. This contradicts the assumption that s is a strict subterm of t. Hence we will find an intermediate term $R[s]$, as required. □

Lemma D.2
Let s be a strict subterm of the expression t, where $t = S[s]$ and S is a surface context. Then the following holds:

1. If s is of one of the following forms:
   ($\texttt{letrec}\ E\ \texttt{in}\ s'$), ($s'\ s''$), ($\texttt{seq}\ s'\ s''$), or ($\texttt{case}_T\ s'\ alts$),
   then s' is a strict subterm of t.
2. Every superterm of s in t is a strict subterm of t.
3. If $t = C[(\texttt{letrec}\ x = s', E\ \texttt{in}\ C'[x])]$, and x is a strict subterm of t in a surface context, then s' is also a strict subterm of t.

Proof
The first claim follows from Corollary C.2, since ($\texttt{letrec}\ E\ \texttt{in}\ \Omega$) $\sim_c$ ($\Omega\ s''$) $\sim_c$ ($\texttt{seq}\ \Omega\ s''$) $\sim_c$ ($\texttt{case}_T\ \Omega\ alts$) $\sim_c \Omega$, since in each of the expressions Ω is in a reduction context of s.
The second claim follows from the properties of a precongruence: If $t = C[D[s]]$ we have $C[D[\Omega]] \sim_c \Omega$. Since $\Omega \leq_c D[\Omega]$, we obtain $C[\Omega] \leq_c C[D[\Omega]] \sim_c \Omega$, hence $C[\Omega] \sim_c \Omega$.
The third claim can be proved using Lemma D.1 which shows that every evaluation of t has an intermediate term $R[x]$. Hence s' will occur under a reduction context in every evaluation. Thus s' is a strict subterm of t. □

Lemma D.3
Let t be a term with $t{\Downarrow}$, let s be a strict subterm of the expression t with $s \neq t$, $t = S[s]$ where S is a surface context. Let $t \xrightarrow{a} t'$ by a reduction a from the base calculus.
If s is not a value and not a `letrec`-expression, then s or its contractum is also a strict subterm of t'. If s is a `letrec`-expression ($\texttt{letrec}\ Env_1\ \texttt{in}\ (\texttt{letrec}\ Env_2\ \texttt{in}\ \ldots(\texttt{letrec}\ Env_n\ \texttt{in}\ s_0)\ldots)$) and s_0 is not a `letrec`-expression and not a value, then s_0 or its contractum is also a strict subterm of t'.

Proof
(1) First assume that s is not a `letrec`-expression and not a value. If the reduction is within s, i.e. $s \to s'$ and $t' = t[s'/s]$, then the lemma holds, since $t[\Omega/s] \sim_c \Omega$, and so also $t[\Omega/s'] \sim_c \Omega$. This also holds, if a (cp) or (seq) has its inner redex in s, but

the redex is not in s. If a (case)-reduction is such that the `case`-expression is within s, and the constructor application is not in s, then we have $t[\Omega/s] \xrightarrow{abs} t'[\Omega/s']$, hence by Theorem B.35, we obtain $t'[\Omega/s'] \sim_c \Omega$,

If the reduction does not change s, then also $t[\Omega/s] \to t'[\Omega/s]$ by a reduction from the base calculus. In this case Theorem B.34 shows that $t'[\Omega/s] \sim_c \Omega$.

The other case is that s is not changed, but eliminated by (seq) or (case). In this case we have $t[\Omega/s] \to t'[\Omega/s] = t'$ and we reach the contradiction $t' \sim_c \Omega$ using Theorem B.34.

It is not possible by assumption that s is copied by a (cp), or that the top level of s is destroyed by a (lll)-reduction, or that s is the constructor application used in a (case)-reduction, or that s is the head of a (lbeta)-reduction.

(2) Now assume that $s = (\texttt{letrec}\ Env_1\ \texttt{in}\ (\texttt{letrec}\ Env_2\ \texttt{in}\ \ldots(\texttt{letrec}\ Env_n\ \texttt{in}\ s_0)\ldots))$. Lemma D.2 shows that s_0 is a strict subexpression of t. Since s_0 is is not a `letrec`-expression and not a value, we can apply the first part of the proof. □

E Reduction Lengths for Different Reductions

We prove the properties of the length of evaluations.

For the purposes of the proofs in this appendix, we generalize the definition of the length measures to reduction sequences, consistently with Definition 2.13.

Definition E.1
Let *Red* be a normal order reduction sequence. Then

1. If $\emptyset \neq M \subseteq \{\text{case,lbeta,seq,cp,lll}\}$, then $\mathrm{rl}_M(Red)$ is defined to be the number of normal-order reductions $\xrightarrow{a}$ in *Red*.
2. $\mathrm{rl}\sharp\sharp(Red) := \mathrm{rl}_{\{case,lbeta,seq\}}(Red)$.
3. $\mathrm{rl}\sharp(Red := \mathrm{rl}_{\{case,lbeta,seq,cp\}}(Red)$.
4. $\mathrm{rl}\flat(Red) := \mathrm{rl}_{\{lll\}}(Red)$.
5. $\mathrm{rl}(Red) := \mathrm{rl}_{\{case,lbeta,seq,cp,lll\}}(Red)$.

For claims about lengths of reductions, only complete sets of forking diagrams are required. On the other hand, we cannot use the context lemma, and thus also have to treat overlappings where the reduction is within the body of a lambda abstraction.

In the following section we prove Theorem 2.14, here repeated in this appendix:

Theorem E.2
Let t_1, s_1 be closed and terminating concrete expressions with $t_1\Downarrow$ and $t_1 \to s_1$ by a base reduction or an extra transformation. Then $s_1\Downarrow$ and the following holds:

1. If $t_1 \xrightarrow{a} s_1$ with $a \in \{\text{case}, \text{seq}, \text{lbeta}, \text{cp}\}$, then $\mathrm{rl}(t_1) \geq \mathrm{rl}(s_1)$, $\mathrm{rl}\sharp(t_1) \geq \mathrm{rl}\sharp(s_1)$ and $\mathrm{rl}\sharp\sharp(t_1) \geq \mathrm{rl}\sharp\sharp(s_1)$.
2. If $t_1 \xrightarrow{\mathcal{S},a} s_1$ with $a \in \{\text{caseS}, \text{seqS}, \text{lbeta}, \text{cpS}\}$, then $\mathrm{rl}\sharp(t_1) \geq \mathrm{rl}\sharp(s_1) \geq \mathrm{rl}\sharp(t_1) - 1$ and $\mathrm{rl}\sharp\sharp(t_1) \geq \mathrm{rl}\sharp\sharp(s_1) \geq \mathrm{rl}\sharp\sharp(t_1) - 1$. For $a = \text{cpS}$, the equation $\mathrm{rl}\sharp\sharp(t_1) = \mathrm{rl}\sharp\sharp(s_1)$ holds.
3. If $t_1 \xrightarrow{a} s_1$ with $a \in \{\text{lll}, \text{gc}\}$, then $\mathrm{rl}(t_1) \geq \mathrm{rl}(s_1)$, $\mathrm{rl}\sharp(t_1) = \mathrm{rl}\sharp(s_1)$ and $\mathrm{rl}\sharp\sharp(t_1) = \mathrm{rl}\sharp\sharp(s_1)$. For $a = gc1$ in addition $\mathrm{rl}(t_1) = \mathrm{rl}(s_1)$ holds.

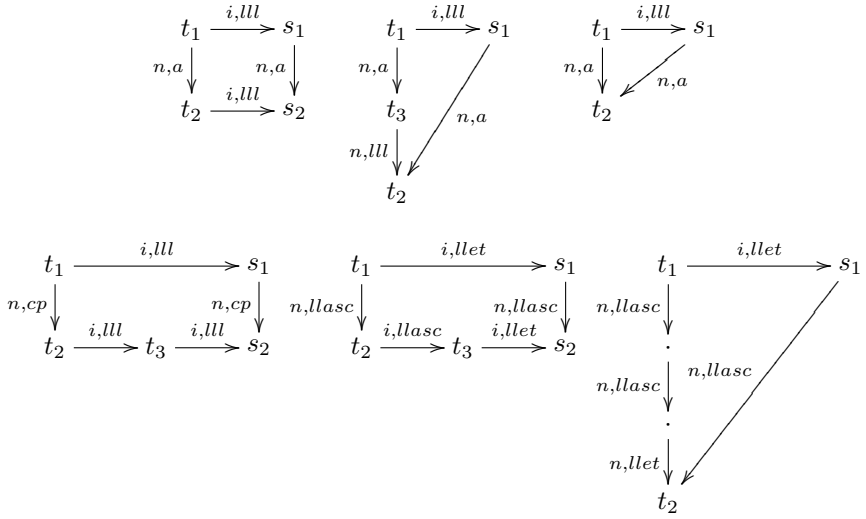
4. If $t_1 \xrightarrow{a} s_1$ with $a \in \{\text{cpx}, \text{cpax}, \text{xch}, \text{cpcx}, \text{abs}\}$, then $\text{rl}(t_1) = \text{rl}(s_1)$, $\text{rl}\sharp(t_1) = \text{rl}\sharp(s_1)$ and $\text{rl}\sharp\sharp(t_1) = \text{rl}\sharp\sharp(s_1)$.
5. If $t_1 \xrightarrow{ucp} s_1$, then $\text{rl}(t_1) \geq \text{rl}(s_1)$, $\text{rl}\sharp(t_1) \geq \text{rl}\sharp(s_1)$ and $\text{rl}\sharp\sharp(t_1) = \text{rl}\sharp\sharp(s_1)$.
6. If $t_1 \xrightarrow{lwas} s_1$, then $\text{rl}\sharp\sharp(t_1) = \text{rl}\sharp\sharp(s_1)$.

Proof
(1) follows from Proposition E.18. (2) follows from Proposition E.20. (3) follows from Proposition E.5. (4) follows from Propositions E.7, E.8, E.12 and E.10. (5) follows from Proposition E.14. (6) follows from Proposition E.15. □

E.1 Reduction Lengths for (lll) and (gc)

For the purposes of this subsection we denote the union of the reductions (lapp), (lseq), (lcase) as (llasc).

Lemma E.3
A complete set of forking and commuting diagrams for (i, lll), where a is an arbitrary reduction type, is as follows:



Proof
We make the case analyses for the forking diagrams. There are a number of standard cases:

- the reductions commute, or
- the reductions commute, and the (i,lll)-reduction is turned into a (n,lll)-reduction, or
- the (i,lll)-reduction is in a term that is removed by the reduction, i.e., a lost case-alternative.
- the (i,lll)-reduction is within a copied abstraction.

This leads to cases 1 to 4.
All overlappings of an (i,b)-reduction, where $b \in \{(\text{lseq}), (\text{lcase}), (\text{lapp}), (\text{llet-e})\}$ lead to one of the diagrams 1-4. The non-standard cases are overlappings of a reduction $\xrightarrow{i,\text{llet-in}}$ with a normal order redex: we demonstrate the reductions by representative examples.

- $(((\texttt{letrec}\ Env_1\ \texttt{in}\ (\texttt{letrec}\ Env_2\ \texttt{in}\ t_1))\ t_2)\ t_3)$
  $\xrightarrow{n,lapp}$ $((\texttt{letrec}\ Env_1\ \texttt{in}\ ((\texttt{letrec}\ Env_2\ \texttt{in}\ t_1)\ t_2))\ t_3)$
  $\xrightarrow{i,lapp}$ $((\texttt{letrec}\ Env_1\ \texttt{in}\ (\texttt{letrec}\ Env_2\ \texttt{in}\ (t_1\ t_2)))\ t_3)$
  $\xrightarrow{i,llet}$ $((\texttt{letrec}\ Env_1, Env_2\ \texttt{in}\ (t_1\ t_2))\ t_3)$

- $(((\texttt{letrec}\ Env_1\ \texttt{in}\ (\texttt{letrec}\ Env_2\ \texttt{in}\ t_1))\ t_2)\ t_3)$
  $\xrightarrow{i,llet}$ $(((\texttt{letrec}\ Env_1, Env_2\ \texttt{in}\ t_1)\ t_2)\ t_3)$
  $\xrightarrow{n,lapp}$ $((\texttt{letrec}\ Env_1, Env_2\ \texttt{in}\ (t_1\ t_2))\ t_3)$

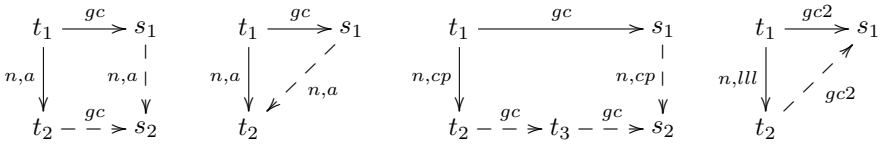
This is covered in the diagram number 5.
A slight variation is the case:

- $((\texttt{letrec}\ Env_1\ \texttt{in}\ (\texttt{letrec}\ Env_2\ \texttt{in}\ t_1))\ t_2)$
  $\xrightarrow{n,lapp}$ $(\texttt{letrec}\ Env_1\ \texttt{in}\ ((\texttt{letrec}\ Env_2\ \texttt{in}\ t_1)\ t_2))$
  $\xrightarrow{n,lapp}$ $(\texttt{letrec}\ Env_1\ \texttt{in}\ (\texttt{letrec}\ Env_2\ \texttt{in}\ (t_1\ t_2)))$
  $\xrightarrow{n,llet}$ $(\texttt{letrec}\ Env_1, Env_2\ \texttt{in}\ (t_1\ t_2))$

- $((\texttt{letrec}\ Env_1\ \texttt{in}\ (\texttt{letrec}\ Env_2\ \texttt{in}\ t_1))\ t_2)$
  $\xrightarrow{i,llet}$ $((\texttt{letrec}\ Env_1, Env_2\ \texttt{in}\ t_1)\ t_2)$
  $\xrightarrow{n,lapp}$ $(\texttt{letrec}\ Env_1, Env_2\ \texttt{in}\ (t_1\ t_2))$

This corresponds to the diagram 6.
The same holds if (lapp) is replaced by (lseq), or (lcase).
Checking all cases shows that no further diagrams are required. □

Lemma E.4
A complete set of forking diagrams for (gc), where a is arbitrary, is as follows:



Proof
We omit the arguments for the cases 1,2,3.

Checking all possibilities for an overlap, it is clear that a (gc)-transformation can only overlap with a normal order reduction that requires a `letrec`. A non-trivial overlap is only possible, if (gc) removes the complete environment, i.e. only with

(gc2). It is easy to check that all cases are covered by the diagrams (see also Lemma B.12). □

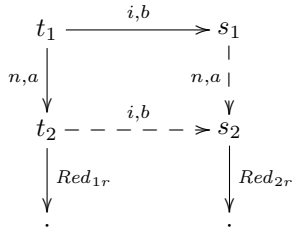
Now we can prove claim 3 of Theorem E.2.

Proposition E.5
Let t_1, s_1 be closed expressions with $t_1\Downarrow$, $Red_1 := nor(t_1)$ and $t_1 \xrightarrow{a} s_1$ where $a \in \{lll, gc\}$. Then $s_1\Downarrow$ and with $Red_2 := nor(s_1)$: $\mathrm{rl}(Red_1) \geq \mathrm{rl}(Red_2)$, $\mathrm{rl}\sharp(Red_1) = \mathrm{rl}\sharp(Red_2)$ and $\mathrm{rl}\sharp\sharp(Red_1) = \mathrm{rl}\sharp\sharp(Red_2)$. If $a = (gc1)$, then in addition $\mathrm{rl}(Red_1) = \mathrm{rl}(Red_2)$.

Proof
The proof constructs a reduction Red_2 using induction on $\mathrm{rl}(Red_1)$.
If t_1 is a WHNF, then s_1 is also a WHNF by Lemmas B.6 and B.13.

First we treat the case that the reduction is an (i,lll):
Let t_1 be the starting term. Let $Red_1 = t_1 \xrightarrow{n,a} t_2 \cdot Red_{1r}$. In the triangular diagrams, it is easy to see that the reduction Red_2 satisfies the length properties. For diagrams 1,4,5, the induction hypothesis has to be used.

The diagrams in Lemma E.3 fix the notation of the terms t_i, s_i. So we associate a reduction Red_{2r} to s_2.



In any case, we have $\mathrm{rl}(Red_1) > \mathrm{rl}(Red_{1r})$, and so we can apply the induction hypothesis to Red_{1r}, perhaps two times, and obtain a reduction Red_{2r} starting from s_2.
It is easy to see inspecting the diagrams, that $\mathrm{rl}\flat(Red_1) \geq \mathrm{rl}\flat(Red_2)$. The additional contribution of the (n,a)-reduction, or the (n,cp)-reduction to $\mathrm{rl}\sharp(Red_1)$ or $\mathrm{rl}\sharp(Red_2)$ is the same in all diagrams, hence $\mathrm{rl}\sharp(Red_1) = \mathrm{rl}\sharp(Red_2)$ holds using induction.

Now we consider the case that the internal reduction is a (gc). The diagrams in Lemma E.4 are used.

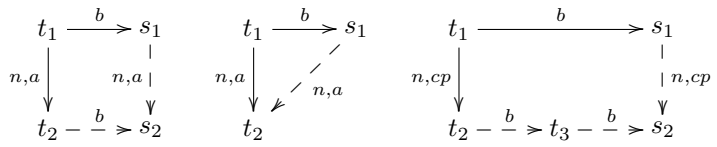
In any case, we have $\mathrm{rl}(Red_1) > \mathrm{rl}(Red_{1r})$, and so we can apply the induction hypothesis to Red_{1r}.
The equality $\mathrm{rl}\sharp(Red_1) = \mathrm{rl}\sharp(Red_2)$ holds in the diagram cases 1,2 since the (n,a)-reductions contribute the same number of reductions. The same for diagram 3, but we have to apply the induction hypothesis twice. In diagram 4, the (n,a)-reduction is a (n,lll), hence $\mathrm{rl}\sharp(Red_1) = \mathrm{rl}\sharp(Red_{2r})$, and $\mathrm{rl}\sharp(Red_{2r}) = \mathrm{rl}\sharp(Red_1)$ by induction.
Exactly the same arguments show the claim of the theorem for $\mathrm{rl}\sharp\sharp(\cdot)$ for (gc) and (lll)-reductions.

The easy induction proof for the property of (gc1) is done by the length $\mathrm{rl}(\cdot)$, omitting diagram 4. □

E.2 Reduction Length for (cpx)-, (cpax)- and (xch)-Transformations

We compute the effect of (cpx)- and (xch)-transformations on the length of evaluations. Note that the diagrams from Lemma B.15 have to be reconsidered, since now all positions in a term have to be covered.

Lemma E.6
A complete set of forking diagrams for $b \in \{cpx, xch\}$ in all contexts is as follows:



Proof
There are only the standard overlappings. □

Concerning the length of evaluations, the following holds:

Proposition E.7
Let t_1 be a closed expression with $t_1\Downarrow$, $Red_1 := nor(t_1)$, and $t_1 \xrightarrow{b} s_1$ where $b \in \{\text{cpx}, \text{xch}\}$. Then $s_1\Downarrow$ and with $Red_2 := nor(s_1)$ we have $\text{rl}\sharp\sharp(Red_1) = \text{rl}\sharp\sharp(Red_2)$, $\text{rl}\sharp(Red_1) = \text{rl}\sharp(Red_2)$ and $\text{rl}(Red_1) = \text{rl}(Red_2)$.

Proof
This follows by induction on $\text{rl}(Red_1)$ from Lemma E.6, Lemma B.16 and B.19. □

We have to treat the length-modifications by (cpax)-transformations:

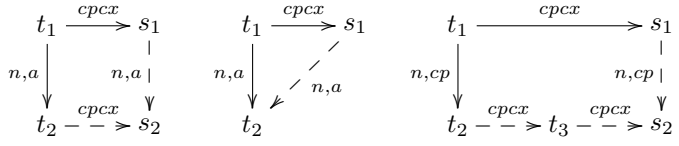
Proposition E.8
Let t_1 be a closed expression with $t_1\Downarrow$, $Red_1 := nor(t_1)$, and $t_1 \xrightarrow{cpax} s_1$. Then $s_1\Downarrow$ and with $Red_2 := nor(s_1)$ we have $\text{rl}\sharp\sharp(Red_1) = \text{rl}\sharp\sharp(Red_2)$, $\text{rl}\sharp(Red_1) = \text{rl}\sharp(Red_2)$ and $\text{rl}(Red_1) = \text{rl}(Red_2)$.

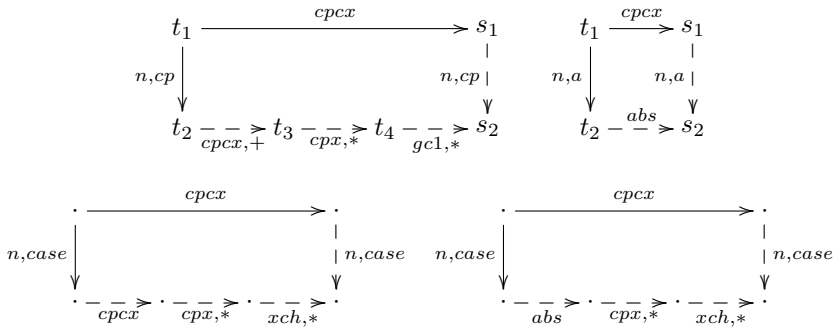
Proof
This follows by induction on the number of variables occurrences that are replaced by the (cpax)-transformation, and from Proposition E.7, since the (cpax)-transformation can be simulated by several (cpx) transformations. □

E.3 Reduction Length for (cpcx)

For the definition of the transformation (cpcx) see Definition 2.8.

Lemma E.9
A complete set of forking diagrams for (cpcx) in arbitrary contexts is as follows.





Proof
The first three cases cover the standard cases, prototypical examples for the other diagrams are already in the proof of Lemma B.22. We give a further prototypical example for diagram 6:

$$
\begin{array}{ll}
 & (\texttt{letrec}\ x = c\ t_1\ t_2, y = x\ \texttt{in}\ \texttt{case}_T\ y\ (c\ y_1\ y_2) \to s) \\
\xrightarrow{cpcx} & (\texttt{letrec}\ x = c\ x_1\ x_2, x_1 = t_1, x_2 = t_2, y = c\ x_1\ x_2\ \texttt{in}\ \texttt{case}_T\ y\ (c\ y_1\ y_2) \to s) \\
\xrightarrow{n,case} & \texttt{letrec}\ x = c\ x_1\ x_2, x_1 = t_1, x_2 = t_2, y = c\ z_1\ z_2, z_1 = x_1, z_2 = x_2 \\
 & \qquad\qquad \texttt{in}\ (\texttt{letrec}\ y_1 = z_1, y_2 = z_2\ \texttt{in}\ s)
\end{array}
$$

$$
\begin{array}{ll}
\xrightarrow{n,case} & (\texttt{letrec}\ x = c\ x_1\ x_2, x_1 = t_1, x_2 = t_2, y = x\ \texttt{in}\ (\texttt{letrec}\ y_1 = x_1, y_2 = x_2\ \texttt{in}\ s)) \\
\xrightarrow{cpcx} & \texttt{letrec}\ x = c\ z_1\ z_2, z_1 = x_1, z_2 = x_2, x_1 = t_1, x_2 = t_2, y = c\ z_1\ z_2 \\
 & \qquad\qquad \texttt{in}\ (\texttt{letrec}\ y_1 = x_1, y_2 = x_2\ \texttt{in}\ s) \\
\xrightarrow{cpx,*} & \texttt{letrec}\ x = c\ x_1\ x_2, z_1 = x_1, z_2 = x_2, x_1 = t_1, x_2 = t_2, y = c\ z_1\ z_2 \\
 & \qquad\qquad \texttt{in}\ (\texttt{letrec}\ y_1 = z_1, y_2 = z_2\ \texttt{in}\ s)
\end{array}
$$

The following case is covered by diagram 5:

$$
\begin{array}{ll}
 & (\texttt{letrec}\ x = c\ t_1\ t_2\ \texttt{in}\ \texttt{case}_T\ x\ (c\ y_1\ y_2) \to s) \\
\xrightarrow{cpcx} & (\texttt{letrec}\ x = c\ x_1\ x_2, x_1 = t_1, x_2 = t_2\ \texttt{in}\ \texttt{case}_T\ (c\ x_1\ x_2)\ (c\ y_1\ y_2) \to s) \\
\xrightarrow{n,case} & \texttt{letrec}\ x = c\ z_1\ z_2, z_1 = x_1, z_2 = x_2, x_1 = t_1, x_2 = t_2 \\
 & \qquad\qquad \texttt{in}\ (\texttt{letrec}\ y_1 = z_1, y_2 = z_2\ \texttt{in}\ s)
\end{array}
$$

$$
\begin{array}{ll}
\xrightarrow{n,case} & \texttt{letrec}\ x = c\ x_1\ x_2, x_1 = t_1, x_2 = t_2 \\
 & \qquad\qquad \texttt{in}\ (\texttt{letrec}\ y_1 = x_1, y_2 = x_2\ \texttt{in}\ s) \\
\xrightarrow{n,abs} & \texttt{letrec}\ x = c\ z_1\ z_2, z_1 = x_1, z_2 = x_2, x_1 = t_1, x_2 = t_2 \\
 & \qquad\qquad \texttt{in}\ (\texttt{letrec}\ y_1 = x_1, y_2 = x_2\ \texttt{in}\ s)
\end{array}
$$

□

Proposition E.10
Let s_1, t_1 be closed expressions with $t_1{\Downarrow}$, $Red_1 := nor(t_1)$ and $t_1 \xrightarrow{cpcx} s_1$. Then $s_1{\Downarrow}$ and with $Red_2 := nor(s_1)$ we have $\mathrm{rl}(Red_1) = \mathrm{rl}(Red_2)$, $\mathrm{rl}\sharp(Red_1) = \mathrm{rl}\sharp(Red_2)$ and $\mathrm{rl}\sharp\sharp(Red_1) = \mathrm{rl}\sharp\sharp(Red_2)$.

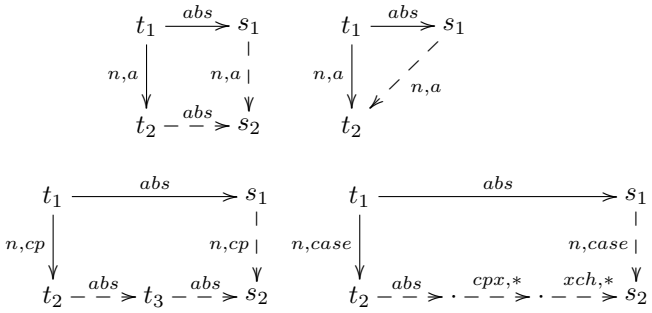
Proof

The proof is by induction on $\mathrm{rl}(Red_1)$, where Lemma B.23 is used for the base case, and the diagrams in the following Lemmas are used: E.9, E.4, E.11, and E.6.
□

E.4 Reduction Length for (abs)

For the definition of the (abs)-transformation see figure 3.

Lemma E.11
The forking diagrams for (abs) in arbitrary contexts are as follows.



Proof
The cases are standard, only the last diagram requires an explicit justification:

$$
\begin{array}{ll}
 & (\texttt{letrec}\ x = c\ t_1\ t_2\ \texttt{in}\ C[\texttt{case}_T\ x\ (c\ y_1\ y_2) \to s]) \\
\xrightarrow{abs} & (\texttt{letrec}\ x = c\ x_1\ x_2, x_1 = t_1, x_2 = t_2\ \texttt{in}\ C[\texttt{case}_T\ x\ (c\ y_1\ y_2) \to s]) \\
\xrightarrow{n,case} & (\texttt{letrec}\ x = c\ z_1\ z_2, z_1 = x_1, z_2 = x_2, x_1 = t_1, x_2 = t_2\ \texttt{in} \\
 & \quad C[(\texttt{letrec}\ y_1 = z_1, y_2 = z_2\ \texttt{in}\ s)])
\end{array}
$$

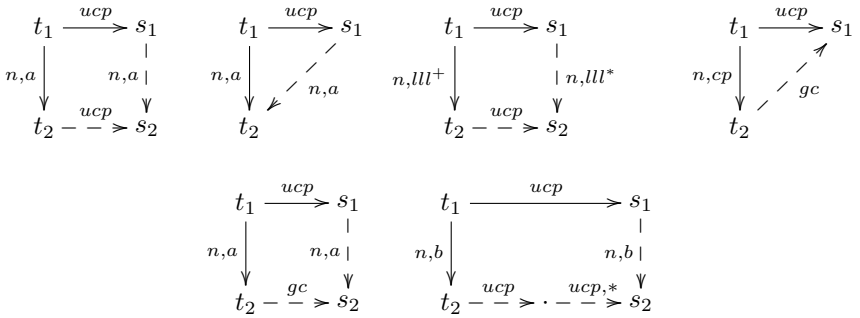
$$
\begin{array}{ll}
\xrightarrow{n,case} & (\texttt{letrec}\ x = c\ z_1\ z_2, z_1 = t_1, z_2 = t_2\ \texttt{in}\ C[(\texttt{letrec}\ y_1 = z_1, y_2 = z_2\ \texttt{in}\ s)]) \\
\xrightarrow{abs} & (\texttt{letrec}\ x = c\ x_1\ x_2, x_1 = z_1, x_2 = z_2, z_1 = t_1, z_2 = t_2\ \texttt{in} \\
 & \quad C[(\texttt{letrec}\ y_1 = z_1, y_2 = z_2\ \texttt{in}\ s)]) \\
\xrightarrow{cpx,*} & (\texttt{letrec}\ x = c\ z_1\ z_2, x_1 = z_1, x_2 = z_2, z_1 = t_1, z_2 = t_2\ \texttt{in} \\
 & \quad C[(\texttt{letrec}\ y_1 = z_1, y_2 = z_2\ \texttt{in}\ s)]) \\
\xrightarrow{xch,*} & (\texttt{letrec}\ x = c\ z_1\ z_2, z_1 = x_1, z_2 = x_2, x_1 = t_1, x_2 = t_2\ \texttt{in} \\
 & \quad C[(\texttt{letrec}\ y_1 = z_1, y_2 = z_2\ \texttt{in}\ s)])
\end{array}
$$
□

Proposition E.12
Let t_1, s_1 be closed expressions with $t_1{\Downarrow}$, $Red_1 := nor(t_1)$ and $t_1 \xrightarrow{abs} s_1$. Then $s_1{\Downarrow}$ and with $Red_2 := nor(s_1)$ we have $\mathrm{rl}(Red_1) = \mathrm{rl}(Red_2)$, $\mathrm{rl}_\sharp(Red_1) = \mathrm{rl}_\sharp(Red_2)$ and $\mathrm{rl}_{\sharp\sharp}(Red_1) = \mathrm{rl}_{\sharp\sharp}(Red_2)$.

Proof
The proof is by induction on $\mathrm{rl}(Red_1)$, where the diagrams in Lemma E.11 are used, and part 4 in Theorem E.2, and since (abs) transforms WHNFs into WHNFs and vice versa. □

E.5 Reduction Length for ucp-Transformations

Lemma E.13
A complete sets of forking diagrams for $\xrightarrow{ucp}$ in arbitrary contexts is as follows:



where $a \in \{(seq), (case)\}$ in the 5^{th} diagram, and $b \in \{(cp), (case)\}$ in the 6^{th} diagram.

Proof
The first five diagrams are as in Lemma B.28, the 6^{th} diagram covers in addition the case that the (ucp) takes place in the body of an abstraction. □

Proposition E.14
Let t_1 be a closed expression with $t_1{\Downarrow}$, $Red_1 := nor(t_1)$ and $t_1 \xrightarrow{ucp} s_1$. Then $s_1{\Downarrow}$ and with $Red_2 := nor(s_1)$ we have $\mathrm{rl}\sharp\sharp(Red_1) = \mathrm{rl}\sharp\sharp(Red_2)$ and $\mathrm{rl}\sharp(Red_1) \geq \mathrm{rl}\sharp(Red_2)$.

Proof
This follows by induction on $\mathrm{rl}\sharp(Red_1)$ and then on $\mathrm{rl}(Red_1)$ from Lemma B.29, Lemma E.13 and Proposition E.5. □

E.6 Reduction Length for (lwas)-Transformations

Proposition E.15
Let t_1 be a closed expression with $t_1{\Downarrow}$, $Red_1 := nor(t_1)$ and $t_1 \xrightarrow{lwas} s_1$. Then $s_1{\Downarrow}$ and with $Red_2 := nor(s_1)$ we have $\mathrm{rl}\sharp\sharp(Red_1) = \mathrm{rl}\sharp\sharp(Red_2)$.

Proof
Since (lwas) can be simulated using (ucp) and (llet)-reductions in both directions (see proof of Lemma B.32), Propositions E.14, and E.5 show the claim. □

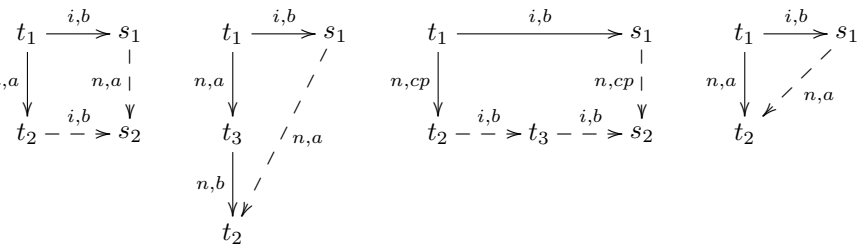
It would also be possible to sharpen this proposition, however, this is not necessary for the further development.

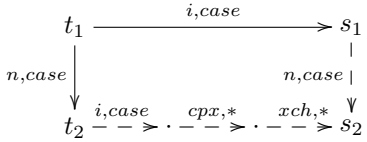
E.7 Using Diagrams for Internal Base Reductions

Now we analyze the length-change of evaluations due to internal base reductions.

Lemma E.16

A complete set of forking diagrams for internal reductions with $b \in \{\text{case}, \text{seq}, \text{lbeta}, \text{cp}\}$, where a is the kind of the normal order reduction, and all contexts are permitted, is as follows





Proof
The conflicts are only between (i,b) and the rule (cp), in which case the b-reduction may be within the copied expression, or in a removed alternative of a case, or in a subterm removed by (seq).
The exceptional diagram is a (case)-(case)-overlapping:

$$
\begin{array}{ll}
 & (\texttt{letrec}\ x = c\ t_1\ t_2\ \texttt{in}\ C[\texttt{case}_T\ x\ (c\ z_{1,1}\ z_{1,2}) \to s_1, \texttt{case}_T\ x\ (c\ z_{2,1}\ z_{2,2}) \to s_2]) \\
\xrightarrow{i,case} & (\texttt{letrec}\ x = c\ y_1\ y_2, y_1 = t_1, y_2 = t_2\ \texttt{in} \\
 & C[\texttt{case}_T\ x\ (c\ z_{1,1}\ z_{1,2}) \to s_1, (\texttt{letrec}\ z_{2,1} = y_1, z_{2,2} = y_2\ \texttt{in}\ s_2)]) \\
\xrightarrow{n,case} & (\texttt{letrec}\ x = c\ y_1'\ y_2', y_1 = t_1, y_2 = t_2, y_1' = y_1, y_2' = y_2\ \texttt{in} \\
 & C[(\texttt{letrec}\ y_1' = z_{1,1}, y_2' = z_{1,2}\ \texttt{in}\ s_1), (\texttt{letrec}\ z_{2,1} = y_1, z_{2,2} = y_2\ \texttt{in}\ s_2)])
\end{array}
$$

$$
\begin{array}{ll}
\xrightarrow{n,case} & (\texttt{letrec}\ x = c\ y_1'\ y_2', y_1' = t_1, y_2' = t_2\ \texttt{in} \\
 & C[(\texttt{letrec}\ z_{1,1} = y_1', z_{1,2} = y_2'\ \texttt{in}\ s_1), \texttt{case}_T\ x\ (c\ z_{2,1}\ z_{2,2}) \to s_2]) \\
\xrightarrow{i,case} & (\texttt{letrec}\ x = c\ y_1\ y_2, y_1' = t_1, y_2' = t_2, y_1 = y_1', y_2 = y_2'\ \texttt{in} \\
 & C[(\texttt{letrec}\ z_{1,1} = y_1', z_{1,2} = y_2'\ \texttt{in}\ s_1), (\texttt{letrec}\ z_{2,1} = y_1, z_{2,2} = y_2\ \texttt{in}\ s_2)]) \\
\xrightarrow{i,cpx,*} & (\texttt{letrec}\ x = c\ y_1'\ y_2', y_1' = t_1, y_2' = t_2, y_1 = y_1', y_2 = y_2'\ \texttt{in} \\
 & C[(\texttt{letrec}\ z_{1,1} = y_1', z_{1,2} = y_2'\ \texttt{in}\ s_1), (\texttt{letrec}\ z_{2,1} = y_1, z_{2,2} = y_2\ \texttt{in}\ s_2)]) \\
\xrightarrow{i,xch,*} & (\texttt{letrec}\ x = c\ y_1'\ y_2', y_1 = t_1, y_2 = t_2, y_1' = y_1, y_2' = y_2\ \texttt{in} \\
 & C[(\texttt{letrec}\ z_{1,1} = y_1', z_{1,2} = y_2'\ \texttt{in}\ s_1), (\texttt{letrec}\ z_{2,1} = y_1, z_{2,2} = y_2\ \texttt{in}\ s_2)])
\end{array}
$$

□

Lemma E.17
If t is a closed WHNF, and $t \xrightarrow{i,b} t'$ for $b \in \{\text{case}, \text{seq}, \text{cp}, \text{lbeta}\}$, then t' is a (closed) WHNF.

Proof
This follows by checking the possible positions of the reduction in a WHNF. □

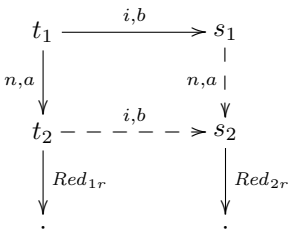
Now we can prove claim 1 of Theorem E.2

Proposition E.18
Let t_1, s_1 be closed expressions with $t_1\Downarrow$, $Red_1 := nor(t_1)$ and $t_1 \xrightarrow{a} s_1$ where $a \in \{\text{case}, \text{seq}, \text{lbeta}, \text{cp}\}$. Then $s_1\Downarrow$ and with $Red_2 := nor(s_1)$ we have $\text{rl}(Red_1) \geq \text{rl}(Red_2)$, $\text{rl}\sharp(Red_1) \geq \text{rl}\sharp(Red_2)$ and $\text{rl}\sharp\sharp(Red_1) \geq \text{rl}\sharp\sharp(Red_2)$.

Proof
The proof will be done by induction on the length $\text{rl}(Red_1)$.
The induction base is that t_1 is in WHNF, in which case we apply Lemma E.17 to show that $t_1 \xrightarrow{i,b} s_1$ and $\text{rl}(Red_1) = 0$ imply $\text{rl}(Red_2) = 0$, $\text{rl}\sharp(Red_1) = \text{rl}\sharp(Red_2) = 0$, and $\text{rl}\sharp\sharp(Red_1) = \text{rl}\sharp\sharp(Red_2) = 0$.

For the induction step assume that $Red_1 = t_1 \xrightarrow{n} t_2 \cdot Red_{1r}$ and $t_1 \xrightarrow{i,b} s_1$. Lemma E.16 shows that there are 5 possible cases.

In any case, we have $\text{rl}(Red_1) > \text{rl}(Red_{1r})$, and so we can apply the induction hypothesis to Red_{1r}.
In case 2 the relations $\text{rl}\sharp(Red_1) > \text{rl}\sharp(Red_2)$, and $\text{rl}(Red_1) > \text{rl}(Red_2)$, and $\text{rl}\sharp\sharp(Red_1) \geq \text{rl}\sharp\sharp(Red_2)$ can be directly derived from the diagrams, and in case 4, we obtain $\text{rl}\sharp(Red_1) \geq \text{rl}\sharp(Red_2)$, $\text{rl}(Red_1) \geq \text{rl}(Red_2)$, and $\text{rl}\sharp\sharp(Red_1) \geq \text{rl}\sharp\sharp(Red_2)$.
We use the following notational conventions in this proof for the rectangle-cases 1,3,5:



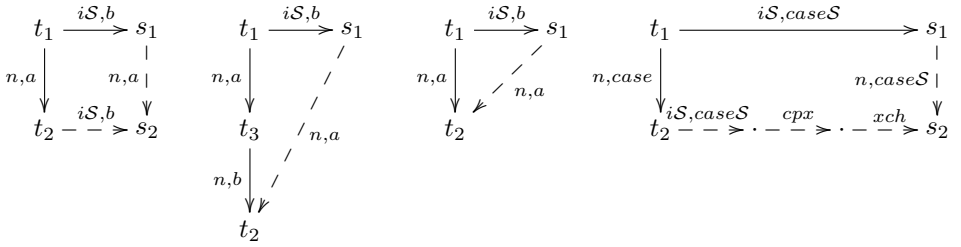
In case 1, we obtain by induction that there exists a reduction Red_{2r} of t_2 with $\text{rl}\sharp(Red_{1r}) \geq \text{rl}\sharp(Red_{2r})$, $\text{rl}(Red_{1r}) \geq \text{rl}(Red_{2r})$, and $\text{rl}\sharp\sharp(Red_{1r}) \geq \text{rl}\sharp\sharp(Red_{2r})$. In case 3, we have to apply the induction hypothesis twice and obtain that there is a reduction Red_3 with $\text{rl}(Red_{1r}) \geq \text{rl}(Red_3)$, hence also a reduction Red_{2r} of s_2 with $\text{rl}\sharp(Red_{1r}) \geq \text{rl}\sharp(Red_{2r})$, and $\text{rl}(Red_{1r}) \geq \text{rl}(Red_{2r})$, and $\text{rl}\sharp\sharp(Red_{1r}) \geq \text{rl}\sharp\sharp(Red_{2r})$.
In cases 1 and 3, we obtain $\text{rl}(Red_1) \geq \text{rl}(Red_2)$. Since the first normal order reduction steps starting from t_1 and from s_1 are of the same kind, we obtain also $\text{rl}\sharp(Red_1) \geq \text{rl}\sharp(Red_2)$ and $\text{rl}\sharp\sharp(Red_1) \geq \text{rl}\sharp\sharp(Red_2)$.

In the fifth case, we apply induction using the existence of evaluations and the preservation of their lengths by the (xch)- and (cpx)-transformations proved in Proposition E.7. □

E.8 Base Reductions in Surface Contexts

Now we treat the case of S-restricted internal base reductions in surface contexts, which is necessary to obtain sharper bounds in this case.

Lemma E.19
A complete set of forking diagrams for $b \in \{\text{caseS}, \text{seqS}, \text{lbeta}, \text{cpS}\}$, where a is the kind of the normal order reduction, and the b-reduction is in a surface context, is as follows:



Proof
The same arguments as in the proof of Lemma E.16 can be used. See also Lemma B.8. Note that the duplicating (n,cp)-diagram does not occur, since the reductions are in surface contexts and the context C in their definition is also restricted to a surface context. □

Now we can prove claim 2 of Theorem E.2

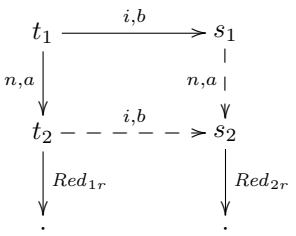
Proposition E.20
Let t_1, s_1 be a closed expression with $t_1\Downarrow$, $Red_1 := nor(t_1)$ and $t_1 \xrightarrow{\mathcal{S},a} s_1$ where $a \in \{\text{caseS}, \text{seqS}, \text{lbeta}, \text{cpS}\}$. Then $s_1\Downarrow$ and with $Red_2 := nor(s_1)$ we have $\text{rl}\sharp(Red_1) \geq \text{rl}\sharp(Red_2) \geq \text{rl}\sharp(Red_1) - 1$ and $\text{rl}\sharp\sharp(Red_1) \geq \text{rl}\sharp\sharp(Red_2) \geq \text{rl}\sharp\sharp(Red_1) - 1$. For $a = \text{cpS}$, in addition $\text{rl}\sharp\sharp(Red_1) = \text{rl}\sharp\sharp(Red_2)$ holds.

Proof
Proposition E.18 already shows that there exists $Red_2 = nor(s_1)$ with $\text{rl}(Red_1) \geq \text{rl}(Red_2)$, $\text{rl}\sharp(Red_1) \geq \text{rl}\sharp(Red_2)$ and $\text{rl}\sharp\sharp(Red_1) \geq \text{rl}\sharp\sharp(Red_2)$. So it remains to prove that $\text{rl}\sharp(Red_2) \geq \text{rl}\sharp(Red_1) - 1$ and $\text{rl}\sharp\sharp(Red_2) \geq \text{rl}\sharp\sharp(Red_1) - 1$ for the same constructed reduction Red_2.

The proof will be done by induction on the length $\text{rl}(Red_1)$. The induction base is that t_1 is in WHNF, in which case we apply Lemma E.17 to show that $t_1 \xrightarrow{i,a} s_1$ and $\text{rl}(Red_1) = 0$ imply $\text{rl}(Red_2) = 0$, $\text{rl}\sharp(Red_1) = \text{rl}\sharp(Red_2) = 0$, and $\text{rl}\sharp\sharp(Red_1) = \text{rl}\sharp\sharp(Red_2) = 0$.

For the induction step assume that $t_1 \xrightarrow{n} t_2$ and $t_1 \xrightarrow{i\mathcal{S},b} s_1$. Lemma E.19 shows that there are four possible cases.

We use the following notational conventions for the rectangle-case 1 :



In case 1 we have $\mathrm{rl}(Red_1) > \mathrm{rl}(Red_{2r})$, and so we can apply the induction hypothesis to Red_{1r}.
Furthermore, there is a reduction Red_{2r} of s_2 with $\mathrm{rl}\sharp(Red_{2r}) \geq \mathrm{rl}\sharp(Red_{1r}) - 1$ and $\mathrm{rl}\sharp\sharp(Red_{2r}) \geq \mathrm{rl}\sharp\sharp(Red_{1r}) - 1$ by induction hypothesis. This implies the claim, by adding a δ to either side of the two inequations, where δ may be 0 or 1 depending on the kind of reduction a.

In case 2, the measures depend on the kind of reductions a, b: The equation $\mathrm{rl}\sharp(Red_1) - 1 = \mathrm{rl}\sharp(Red_2)$ holds, and either the equation $\mathrm{rl}\sharp\sharp(Red_1) - 1 = \mathrm{rl}\sharp\sharp(Red_2)$ or $\mathrm{rl}\sharp\sharp(Red_1) = \mathrm{rl}\sharp\sharp(Red_2)$ holds.

In case 3, the equations $\mathrm{rl}\sharp(Red_1) = \mathrm{rl}\sharp(Red_2)$ and $\mathrm{rl}\sharp\sharp(Red_1) = \mathrm{rl}\sharp\sharp(Red_2)$ hold.

In case 4, the equations $\mathrm{rl}\sharp(Red_1) = \mathrm{rl}\sharp(Red_2)$ and $\mathrm{rl}\sharp\sharp(Red_1) = \mathrm{rl}\sharp\sharp(Red_2)$ hold by induction similar to diagram 1 using Proposition E.7.

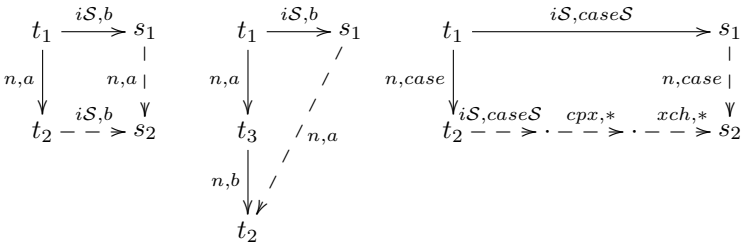
In the case that $a = \mathrm{cpS}$, the equation $\mathrm{rl}\sharp\sharp(Red_1) = \mathrm{rl}\sharp\sharp(Red_2)$ follows by induction using the diagrams 1,2,3. □

E.9 Length of Normal Order Reduction Using Strictness Optimization

In this subsection we give a proof of Proposition 2.15, here repeated in this appendix:

Proposition E.21
Let t_1, s_1 be closed concrete LR-expressions with $t_1{\Downarrow}$ and $t_1 \xrightarrow{\mathcal{S},b} s_1$, where $b \in \{(\mathrm{caseS}), (\mathrm{seqS}), (\mathrm{lbeta}), (\mathrm{cpS})\}$, such that the inner redex t_0 of the reduction is a strict subterm of t_1, and $t_1 = S[t_0]$ for a surface context S.
Then $\mathrm{rl}\sharp(t_1) = 1 + \mathrm{rl}\sharp(s_1)$. If $b \in \{(\mathrm{caseS}), (\mathrm{seqS}), (\mathrm{lbeta})\}$, then $\mathrm{rl}\sharp\sharp(t_1) = 1 + \mathrm{rl}\sharp\sharp(s_1)$ and if $b = (cp\mathcal{S})$, then $\mathrm{rl}\sharp\sharp(t_1) = \mathrm{rl}\sharp\sharp(s_1)$.

Proof
We apply induction on $\mathrm{rl}(t_1)$.
It is not possible that t_1 is a WHNF, since then the condition that there is a b-redex on a surface position for $b \in \{(case\mathcal{S}), (seq\mathcal{S}), (lbeta), (cp\mathcal{S})\}$ and that $t_1[\Omega/t_0] \sim_c \Omega$ cannot hold simultaneously.
Let $t_1 \xrightarrow{n} t_2$. If this is the same reduction as $t_1 \xrightarrow{\mathcal{S},b} s_1$, then the claim of the proposition holds. Hence we can assume that $t_1 \xrightarrow{\mathcal{S},b} s_1$ is not a normal order reduction. Since t_0 is neither a value nor a `letrec`-expression, Lemma D.3 shows that the successor of t_0 is also a strict subterm of t_2. The diagrams are as follows, where the $i\mathcal{S}$-reduction reduces the redex t_0 or its descendent (see also Lemma E.19).



The short triangle-diagram from Lemma E.19 does not occur, since t_0 remains a strict subterm.

We use induction on $\mathrm{rl}(t_1)$, where the diagrams above are the cases that have to be considered in the induction step, and use the already known results on the lengths of evaluations (see Theorem E.2) for (xch) and (cpx)-transformations. We obtain that the claim of the proposition holds. □

E.10 Local Evaluation and Deep Subterms

In this section we will prove the Propositions 2.17 and 2.19.

Definition E.22
In the closed concrete term $(\texttt{letrec}\ x = t, y = s, Env\ \texttt{in}\ r)$, we say x *requires* y, iff the local evaluation of x in $(\texttt{letrec}\ x = t, y = \Omega, Env\ \texttt{in}\ r)$ does not result in a WHNF for x.

Lemma E.23
Let $t = (\texttt{letrec}\ x = s_x, y = s_y, Env\ \texttt{in}\ r)$ be a closed term, where x requires y, and let $t \to t'$ by a base-reduction or an extra transformation, such that the bindings $x = s_x, y = s_y$ are not erased. Then in t' the variable x also requires y.

Proof
First assume that the reduction is not an (llet)-reduction.
If $t \to t'$ modifies only r, then the Lemma holds, since there is no difference in the local evaluations of x w.r.t. t and t'. If the reduction modifies a `case`-expression in r, where the constructor application is in the top environment, then $t' = (\texttt{letrec}\ x = s'_x, y = s'_y, Env'\ \texttt{in}\ r')$ and one of the two relations $(\texttt{letrec}\ x = s_x, y = \Omega, Env\ \texttt{in}\ x) = (\texttt{letrec}\ x = s'_x, y = \Omega, Env'\ \texttt{in}\ x)$ or $(\texttt{letrec}\ x = s_x, y = \Omega, Env\ \texttt{in}\ x) \xrightarrow{abs} (\texttt{letrec}\ x = s'_x, y = \Omega, Env'\ \texttt{in}\ x)$ is valid. Theorem B.35 implies that the Lemma holds.

If $t \to t'$ modifies the top environment, then Theorems B.34 and B.35 show the claim of Lemma.

Now assume that the reduction is an (llet)-reduction. If the (llet)-reduction does not change the top level structure of t, then again Theorem B.35 shows that the Lemma holds.

The only non-standard case is that $s_y = (\texttt{letrec}\ Env_y\ \texttt{in}\ s'_y)$ and that it is modified by a (n,llet-in)-reduction: $t' = (\texttt{letrec}\ x = s_x, Env_y, y = s'_y, Env\ \texttt{in}\ r)$. Now the Lemma follows from Lemma D.2. □

Lemma E.24
Let the closed concrete term $t = (\texttt{letrec}\ x_1 = t_1, \ldots, x_n = t_n, Env\ \texttt{in}\ r)$ have a cyclic dependency, i.e., x_i requires x_{i+1} for $i = 1, \ldots, n-1$ and x_n requires x_1.

Then for all i, the local evaluation of x_i does not produce a WHNF for x_i.

Proof
W.l.o.g. we can assume the first reduction step of the local evaluation of x_1 to be a non-(lll) reduction step. Moreover assume, that this is the $\mathrm{rl}\sharp(\cdot)$-shortest such local evaluation for all x_i.

If some x_i is bound to a term that is a value, then this contradicts the assumption that there is a cyclic dependency. Hence every x_i is bound to a term that is not a value. It is sufficient to treat the non- (lll)-reductions. Let the first reduction step of the local evaluation of x_1 be a non-(lll) reduction step. The cyclic dependency remains as before the reduction (see Lemma E.23). The term t' is a counterexample with a shorter $\mathrm{rl}\sharp(\cdot)$-number of a successful local evaluation of an x_i, hence we have a contradiction. This means there is no finite successful local evaluation for x_i for any $i = 1, \ldots, n$. □

We prove Proposition 2.17. The claim is:

Let $t_1 = (\texttt{letrec}\ \mathit{Env}\ \texttt{in}\ t_1')$ be a closed concrete LR-expression with $t_1{\Downarrow}$. Let $x \in LV(\mathit{Env})$ where the binding is $x = t_x$, and t_x is a strict subexpression in t_1.
Then $\mathrm{rl}\sharp(t_1) \geq \mathrm{rl}\sharp_{loc}(\texttt{letrec}\ \mathit{Env}\ \texttt{in}\ x)$ and $\mathrm{rl}\sharp\sharp(t_1) \geq \mathrm{rl}\sharp\sharp(\texttt{letrec}\ \mathit{Env}\ \texttt{in}\ x)$.

Proof
If $x = t_1'$ there is nothing to show. Hence in the following we assume $x \neq t_1'$.
The proof is by induction on $\mathrm{rl}\sharp_{loc}(\texttt{letrec}\ \mathit{Env}\ \texttt{in}\ x)$. If t_x is in WHNF, then $\mathrm{rl}\sharp_{loc}(\texttt{letrec}\ \mathit{Env}\ \texttt{in}\ x) = 0$, and the claim holds. Now let t_x be a non-WHNF. Let $t_1 \to t_2$ be the reduction corresponding to the first local evaluation step of x. If the reduction is an (lll)-reduction, then we can use induction and Theorem E.2. It is easy to see that the inner redex of the reduction is a strict subterm of t_1. The other local reduction types are (cpS), (lbeta), (caseS), (seqS), hence Proposition E.21 and induction on the number of local evaluations shows the claim. □

Finally, we prove Proposition 2.19:

Let $t_1 = (\texttt{letrec}\ \mathit{Env}, x = t_x, x_1 = t_1'\ \texttt{in}\ x_1)$ be a closed concrete LR-expression with $t_1{\Downarrow}$, such that t_x is a strict and deep subterm in t_1.
Then $\mathrm{rl}\sharp\sharp(t_1) > \mathrm{rl}\sharp\sharp(\texttt{letrec}\ \mathit{Env}, x = t_x\ \texttt{in}\ x)$.

Proof
We show by induction on the number of local evaluation steps of x that after a local evaluation of x, t_x remains a strict and deep subterm in t_1.
If t_x is already a WHNF, then it is a value. Due to the syntactic form of t_1, the normal order reduction of t_1 must include at least one (case), (seq), or (lbeta)-reduction to reach a normal form, hence the proposition holds.

If t_x is not a value, then consider a single local evaluation step of x in t_1, i.e. $t_1 \to t_2$. Then t_x or its descendant term (after an (llet-e)) remains a strict subterm in t_2 by Lemma D.3.
The reduction step $t_1 \to t_2$ does not modify t_1', since x_1 requires x due to strictness, and so x cannot require x_1 by Lemma E.24. Hence t_x remains a deep subterm due to the definition.

Concluding, the induction shows that $\mathrm{rl}\sharp\sharp(t_1) > \mathrm{rl}\sharp\sharp(\texttt{letrec}\ \mathit{Env}, x = t_x\ \texttt{in}\ x)$.
□

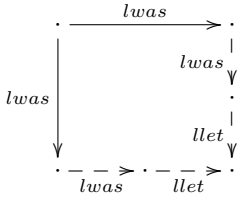
F Confluence and Termination of Simplification

Proof of Theorem 2.12 claiming confluence and termination of the simplification.

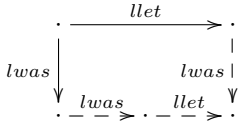
Proof
We have to compute the forking diagrams (critical pairs) between (lwas)-, (llet)-, (cpax)-, and (gc)-reductions and -transformations in order to show local confluence of the reductions. We omit the cases of commutation of the reductions.

The forking diagrams are:

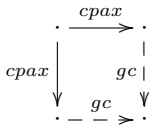
For (lwas) with (lwas):



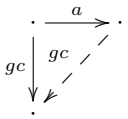
For (lwas) with (llet):



For (cpax) with (cpax):



For (gc) with other reductions:



For termination we only need a well-founded measure of terms that is strictly decreased in every reduction step. This measure μ is a tuple $(\mu_1(t), \mu_2(t), \mu_3(t))$, ordered lexicographically. The measure $\mu_1(t)$ is $\mu_{lll}(t)$ as defined in Definition 2.6 and used in the proof of Proposition 2.7, $\mu_2(t)$ is the number of all bindings in `letrec`-subexpressions in t, and $\mu_3(t)$ is the number of let-bound variables that have occurrences in the expression.

This measure of t is strictly decreased by every transformation (lwas), (cpax), (gc), (llet): The reductions (llet) and (lwas) strictly decrease μ_1, the transformation (gc) strictly decreases μ_1 or leaves μ_1 unchanged and strictly decreases μ_2, and the transformation (cpax) leaves μ_1, μ_2 unchanged, and strictly decreases μ_3.

Finally, we apply the well-known Newman's Lemma which states confluence for terminating and locally confluent reduction systems (see e.g. (Baader & Nipkow, 1998)). □

G Another Definition of Contextual Equivalence

Proposition G.1
$s \leq_c t$ is equivalent to:

$$\forall C[\cdot]: \ C[s], C[t] \text{ are closed} \Rightarrow \Big(C[s]{\Downarrow} \Rightarrow C[t]{\Downarrow}\Big)$$

and to

$$\forall R[\cdot]: \ R \text{ is a reduction context and } R[s], R[t] \text{ are closed} \Rightarrow \Big(R[s]{\Downarrow} \Rightarrow R[t]{\Downarrow}\Big)$$

Proof
One direction is trivial.

Assume that the following holds

$$\forall C[\cdot]: \; C[s], C[t] \text{ are closed} \Rightarrow \Big(C[s]{\Downarrow} \Rightarrow C[t]{\Downarrow}\Big)$$

Let D be an arbitrary context such that $D[s]{\Downarrow}$. Let $\{x_1, \ldots, x_n\}$ be the variables in $FV(D[s], D[t])$. Let $D' := (\texttt{letrec}\ \{x_i = \Omega\}_{i=1}^n\ \texttt{in}\ D)$. Then $D'[s]{\Downarrow}$ follows from $D[s]{\Downarrow}$, and $D'[t]{\Downarrow}$ follows from the assumption. The evaluation of $D'[t]$ never puts any x_i in a reduction context, since this would contradict Corollary C.2. Hence the same method as in the proof of Proposition C.1 shows that we can use the same evaluation to show that $D[t]{\Downarrow}$.

The second equivalence follows from the context lemma and the same reasoning using a closing `letrec`: $(\texttt{letrec}\ \{x_i = \Omega\}_{i=1}^n\ \texttt{in}\ [\cdot])$. □

H Correctness of Copying in Surface Contexts

Proposition H.1
Let $t = (\texttt{letrec}\ x_1 = t_0, Env\ \texttt{in}\ r)$ and $t' = (\texttt{letrec}\ x_1 = t_0', y_1 = t_0'', Env'\ \texttt{in}\ r')$, where the terms r, t_0 are not `letrec`-expressions, and $Env'[x_1/y_1] = Env, r'[x_1/y_1] = r, t_0'[x_1/y_1] = t_0''[x_1/y_1] = t_0$.
Then $t \sim_c t'$.

Proof
We use the context lemma A.1 to show contextual equivalence. For every reduction context R we have to show that $R[t]{\Downarrow} \Leftrightarrow R[t']{\Downarrow}$. It is obvious from the definition of normal order reduction that the first normal order reduction steps of $R[t]$ as well as of $R[t']$ are to shift the top environment of t (of t', respectively) to the top environment of $R[t]$ (of $R[t']$, respectively).

Then the part $\{x_1 = t_0, Env\}$, and $\{x_1 = t_0', y_1 = t_0'', Env'\}$, in the reduct of $R[t]$ or $R[t']$, respectively, are the respective parts of the top environments. The rest of the top environment is denoted in the following as Env_{rest}. We can write the intermediate term for $R[t]$ as $(\texttt{letrec}\ x_1 = t_0, Env, Env_{\text{rest}}\ \texttt{in}\ R_1[r])$, and the intermediate term for $R[t']$ as $(\texttt{letrec}\ x_1 = t_0', y_1 = t_0'', Env', Env_{\text{rest}}\ \texttt{in}\ R_1[r'])$. Now we prove the equivalence.

Let *Red* be the evaluation of $(\texttt{letrec}\ x_1 = t_0, Env, Env_{\text{rest}}\ \texttt{in}\ R_1[r])$. The reduction sequence *Red* is modified by replacing every (case)-reduction by $\xrightarrow{abs} \cdot \xrightarrow{case-cx}$, or by $\xrightarrow{case-cx}$, such that the replacements have the same effect as the original (case). The constructed reduction sequence is denoted as $Red_1 = (q_1 \to q_2 \ldots)$, where $q_1 = R[t]$, and it consists of base reductions, but not (case)-reductions, and (case-cx) and (abs)-reductions. From this sequence we construct a reduction sequence $(q_1' \xrightarrow{*} q_2' \xrightarrow{*} \ldots)$ for $R[t']$, where $q_1' = R[t']$. We will keep a correspondence between the reduction sequences, in particular between q_i and q_i' for all i. The term q_i is of the form $(\texttt{letrec}\ Env_x, Env_{rest}\ \texttt{in}\ q_{in})$. The environment Env_x in q_i deriving from $x_1 = t_0$ will be denoted as $Env_x := \{x_1 = s_1, \ldots, x_n = s_n\}$. The term q_i' is of the form $(\texttt{letrec}\ Env_x', Env_y', Env_{rest}'\ \texttt{in}\ q_{in}')$. The environments Env_x', Env_y' are derived from $x_1 = t_0'$, and $y_1 = t_0''$, respectively, and are $Env_x' := \{x_1 = s_1', \ldots, x_n = s_n'\}$, and $Env_y' := \{y_1 = t_1', \ldots, y_n = t_n'\}$.

Let $\rho := \{y_1 \mapsto x_1, \ldots, y_n \mapsto x_n\}$; then we have $s_i = \rho(s'_i) = \rho(t'_i)$, and $\rho(q'_{in}) = q_{in}, \rho(Env'_{rest}) = Env_{rest}$.

Now we define the reduction for q'_i reaching q'_{i+1} from the reduction of $q_i \to q_{i+1}$ and argue that the correspondence also holds for $i+1$. We distinguish the reduction steps in Red_1 as follows:

- Env_x-related reductions: reductions that make changes in Env_x.
- Env_x-independent reductions: reductions that do not make changes in Env_x.

If the reduction is Env_x-independent, then it is also performed on q'_i at the same position. Note that the involved terms are only equal up to ρ. If it is a (cp)-reduction and the abstraction comes from Env_x, then the to-place is at the corresponding position, but the abstraction may be from Env_x or Env_y, depending on the variable at the to-place. This is exactly one reduction $q'_i \to q'_{i+1}$. An Env_x-related reduction in Red_1 results in 2 reductions in Red': The reduction is duplicated and done on Env_x and Env_y. Again the abstraction in a (cp) may come from Env_x or Env_y, depending on the to-variable.

It is easy to see that the mentioned invariant holds, i.e. that after applying ρ_{i+1}, we obtain that the Env_x or Env_y environments become equal and that after applying ρ and after removal of one environment, we obtain q_i.

In summary, we have constructed a reduction sequence of $R[t']$ to a WHNF using reductions of the calculus and external reductions. From Lemma B.33 and Theorem 2.10 we obtain $R[t']{\Downarrow}$.

In order to prove the other direction, let $R[t']{\Downarrow}$. As mentioned above, the first normal order reduction steps shift the top environment of t' to the top environment. Thus there exists an evaluation of the term ($\texttt{letrec}\ x_1 = t'_0, y_1 = t''_0, Env', Env_{\text{rest}}\ \texttt{in}\ R_1[r']$). The final goal is to show that there is an evaluation of ($\texttt{letrec}\ x_1 = t_0, Env, Env_{\text{rest}}\ \texttt{in}\ R_1[r]$), where $Env'[x_1/y_1] = Env$, $t'_0[x_1/y_1] = t_0$ and $r'[x_1/y_1] = r$.

We consider *synchronized terms* of the form ($\texttt{letrec}\ Env_x, Env_y, Env_{rest}\ \texttt{in}\ q_{in}$), where $Env_x = \{x_1 = s_1, \ldots, x_n = s_n\}$, and $Env_y = \{y_1 = t_1, \ldots, y_n = t_n\}$, and for $\rho = \{x_1 \mapsto y_1, \ldots, x_n \mapsto y_n\}$, we have $\rho(s_i) = \rho(t_i)$ for all i. We will show in the following that given a synchronized term of the form ($\texttt{letrec}\ Env_x, Env_y, Env_{rest}\ \texttt{in}\ q_{in}$), if there is an evaluation, then there is also a *synchronized reduction* to a WHNF, consisting of (lll),(lbeta), (cp),(seq), (case-cx), and (abs)-reductions, where intermediate terms after 1 or 2 reductions are also synchronized. To ease argumentation, we will say *corresponding positions*, if we mean the same position in Env_x and Env_y, where we assume a fixed ordering of bindings.

We show by induction on the pair $(\text{rl}\sharp(q), \mu_{lll}(q))$, ordered lexicographically, that a normal order reduction sequence starting with the synchronized term q can be transformed into a synchronized reduction to WHNF satisfying the correspondence property.

We go through the different possibilities of the first reduction step $q \xrightarrow{n} q'$:

If the reduction step $q \xrightarrow{n} q'$ does not modify the environments Env_x, Env_y, then we use this reduction step and apply induction on q. The resulting term is a syn-

chronized term.
Consider the case that the reduction step $q \xrightarrow{n} q'$ modifies a part of Env_x or Env_y. W.l.o.g we only treat the case that the modification is in Env_x; the other case is treated in a symmetric way.

1. If the reduction is an (lll)-reduction, then apply the corresponding reduction in the other environment giving a synchronized term q''. Note that bindings may be added to Env_x, Env_y, however, this is only possible by an (llet)-reduction within Env_x, Env_y. Proposition 2.7 and Theorem E.2 show that we can use the induction hypothesis for q''.
   Now we assume that the reduction is not an (lll)-reduction.
2. If the reduction is a (seq) or (lbeta) within Env_x, make the same reduction for Env_y giving q''. Theorem E.2 shows that we can use the induction hypothesis for q''. Moreover, q'' is synchronized.
3. If there is a (cp) into Env_x, then the abstraction may be in Env_{rest}, in which case we add the corresponding (cp) to copy the same abstraction into Env_y. If the abstraction is in Env_x, or Env_y, then we also add a (cp) copying the uniquely determined abstraction to the corresponding position giving q''. The abstraction may come from Env_x, or Env_y, and the term q'' is synchronized. Theorem E.2 shows that we can use the induction hypothesis for q''.
4. If the reduction is a (case), then we replace the (case)-reduction as follows: if it is a (case-e) or (case-in)-reduction, then we replace it by an (abs) followed by a (case-cx)-reduction. If it is a (case-c), then it is also a (case-cx)-reduction. First we treat the (abs)-reduction: If the (abs)-reduction is within Env_x, then we make a corresponding (abs)-reduction in Env_y, otherwise, i.e. if it is neither in Env_x nor in Env_y, we only make the (abs)-reduction. The following (case-cx)-reduction is then used to construct the further reduction as follows: If it is neither in Env_x nor in Env_y, then the next reduction is the (case-cx)-reduction. If it is in Env_x, then we make the corresponding (case-cx)-reduction also in Env_y. The resulting q'' is a synchronized term. Theorem E.2 shows that we can use the induction hypothesis for q''.

Finally, we obtain a reduction sequence from q_0 to a WHNF, using reductions from the base calculus and extra transformations and (case-cx), applied only in surface contexts, where the reduction is a synchronized one.

Now it is easy to construct a terminating reduction sequence of $R[t]$: We only use the reductions for Env_x, after applying the renaming ρ (in every step).

We finally have a reduction sequence of $R[t]$ ending in a WHNF, where the steps may be from the base calculus, (abs)- reductions and (case-cx)-reductions. Now Lemma B.33 and Theorem 2.10 show that $R[t]\Downarrow$. □

Lemma H.2
The claim of Proposition H.1 that ($\texttt{letrec}\ x_1 = t_0, Env\ \texttt{in}\ r$) $\sim_c$ ($\texttt{letrec}\ x_1 = t_0', y_1 = t_0'', Env'\ \texttt{in}\ r'$) with $Env'[x_1/y_1] = Env, t_0'[x_1/y_1] = t_0''[x_1/y_1] = t_0, r'[x_1/y_1] = r$ also holds if t_0, r are `letrec`-expressions.

Proof

If r is a `letrec`-expression, then we apply Proposition H.1 after several (llet)-reductions, and then use (llet) backwards several times.
If t_0 is a `letrec`-expression, then we can assume that $t_0 = (\texttt{letrec}\ z_1 = s_1, \ldots, z_n = s_n\ \texttt{in}\ s_{n+1})$, where s_i are not `letrec`-expressions. Applying (llet)-reductions, we obtain
$t \sim_c (\texttt{letrec}\ x_1 = s_{n+1}, z_1 = s_1, \ldots, z_n = s_n, Env\ \texttt{in}\ r)$. An application of Proposition H.1 shows

$$t \quad \sim_c \quad (\texttt{letrec}\ x_1 = s'_{n+1}, y_1 = t_{n+1}, z_1 = s'_1, \ldots, z_n = s'_n, Env'\ \texttt{in}\ r')$$

where $s'_{n+1}[y_1/x_1] = t_{n+1}[y_1/x_1] = s_{n+1}$, $s'_i[y_1/x_1] = s_i$ for $i = 1, \ldots, n$, $Env'[y_1/x_1] = Env$ and $r'[y_1/x_1] = r$. Multiple applications of Proposition H.1 show that we can proceed as follows:

$$t \quad \sim_c \quad \texttt{letrec} \quad x_1 = s'_{n+1}, y_1 = t'_{n+1}, \quad z_1 = s'_1, \ldots, z_n = s'_n \quad z'_1 = t'_1, \ldots, z'_n = t'_n \quad Env'\ \texttt{in}\ r'$$

where $t'_i = s'_i[z'_1/z_1, \ldots, z'_n/z_n]$ for $i = 1, \ldots, n$ and $t'_{n+1} = t_{n+1}[z'_1/z_1, \ldots, z'_n/z_n]$. Now we can use (llet) backwards and obtain

$$t \quad \sim_c \quad \texttt{letrec} \quad x_1 = (\texttt{letrec}\ z_1 = s'_1, \ldots, z_n = s'_n\ \texttt{in}\ s'_{n+1}),\ y_1 = (\texttt{letrec}\ z'_1 = t'_1, \ldots, z'_n = t'_n\ \texttt{in}\ t'_{n+1}),\ Env'\ \texttt{in}\ r'$$

It is easy to verify that $(\texttt{letrec}\ z_1 = s'_1, \ldots, z_n = s'_n\ \texttt{in}\ s'_{n+1})[y_1/x_1] =_\alpha (\texttt{letrec}\ z'_1 = t'_1, \ldots, z'_n = t'_n\ \texttt{in}\ t_{n+1})[y_1/x_1]$, and that every possibility of replacing x_1 by y_1 is covered. □

Now we can show that copying into surface contexts can be done without restrictions:

Corollary H.3
Let $t = (\texttt{letrec}\ x = t_0, Env\ \texttt{in}\ S[x])$ and $t' = (\texttt{letrec}\ x = t_0, Env\ \texttt{in}\ S[t_0])$, or let $t = (\texttt{letrec}\ x = t_0, y = S[x], Env\ \texttt{in}\ r)$ and $t' = (\texttt{letrec}\ x = t_0, y = S[t_0], Env\ \texttt{in}\ r)$. Then $t \sim_c t'$.

Proof
The equivalences $t = (\texttt{letrec}\ x = t_0, Env\ \texttt{in}\ S[x]) \sim_c (\texttt{letrec}\ x = t_0, y = x, Env\ \texttt{in}\ S[y]) \sim_c (\texttt{letrec}\ x = t_0, y = t_0, Env\ \texttt{in}\ S[y]) \xrightarrow{ucp} (\texttt{letrec}\ x = t_0, Env\ \texttt{in}\ S[t_0])$ show the claim, using Proposition H.1. The other case is proved in the same way. □

The following corollary shows that terms in weak application surface contexts can be locally closed by surrounding them with a renamed copy of the global environment. Note that the bindings may be recursive, e.g. there may be an occurrence of x in W.

Corollary H.4

Let $t = (\texttt{letrec}\ Env, x = W[s]\ \texttt{in}\ r)$ be a closed term, where W is a weak application surface context, and let $t' = (\texttt{letrec}\ Env, x = W[(\texttt{letrec}\ Env', x' = W'[s']\ \texttt{in}\ s')]\ \texttt{in}\ r)$, where $\{Env', x' = W'[s']\}$ is a copy of the environment $\{Env, x = W[s]\}$ renamed by $\rho := \{x_1 \mapsto x'_1, \ldots, x_n \mapsto x'_n, x \mapsto x'\}$ for $LV(Env) = \{x_1, \ldots, x_n\}$, such that $(\texttt{letrec}\ Env', x' = W'[s']\ \texttt{in}\ s')$ is closed.
Then $t \sim_c t'$.

Proof
The following equivalences hold: $(\texttt{letrec}\ Env, x = W[s]\ \texttt{in}\ r) \sim_c (\texttt{letrec}\ Env, x = W[y], y = s\ \texttt{in}\ r) \sim_c (\texttt{letrec}\ Env, x = W[y], Env', x' = W'[y], y = s'\ \texttt{in}\ r)$, where $Env = \{x_1 = s_1, \ldots, x_n = s_n\}$, $Env' = \{x'_1 = s'_1, \ldots, x'_n = s'_n\}$, and for $\rho := \{x_1 \mapsto x'_1, \ldots, x_n \mapsto x'_n\}$ it is $s'_i = s_i\rho$, $s' = s\rho$ and $W'[y]\rho = W[y]$. The latter equivalences hold by multiple application of Proposition H.1 (see also the proof of Lemma H.2). We apply corollary H.3 twice and then use (gc) to obtain: $t \sim_c (\texttt{letrec}\ Env, x = W[s'], Env', x' = W'[s']\ \texttt{in}\ r)$. Now a reverse (llet) shows $t \sim_c (\texttt{letrec}\ Env, x = (\texttt{letrec}\ Env', x' = W'[s']\ \texttt{in}\ W[s'])\ \texttt{in}\ r)$. □

Corollary H.5
Let $t = (\texttt{letrec}\ Env\ \texttt{in}\ W[s])$ be a closed term, where W is a weak application surface context, and let $t' = (\texttt{letrec}\ Env\ \texttt{in}\ W[(\texttt{letrec}\ Env'\ \texttt{in}\ s')])$, where Env' is a renamed copy of the environment Env by $\rho = \{x_1 \mapsto x'_1, \ldots, x_n \mapsto x'_n\}$ and $s' = s\rho$, where $LV(Env) = \{x_1, \ldots, x_n\}$, such that $(\texttt{letrec}\ Env'\ \texttt{in}\ s')$ is closed.
Then $t \sim_c t'$.

Proof
Follows from Corollary H.4 using the equivalence $(\texttt{letrec}\ Env\ \texttt{in}\ W[s]) \sim_c (\texttt{letrec}\ Env, x = W[s]\ \texttt{in}\ x)$ and the fact that x does not occur in $W[s]$. □

Now the proof of Proposition 2.20 is easy. The claim is:

Let $t = (\texttt{letrec}\ Env, x = W[t']\ \texttt{in}\ r)$ be a closed expression, where W is a weak application surface context. Then there exists a closed expression t'', such that $t \sim_c (\texttt{letrec}\ Env, x = W[t'']\ \texttt{in}\ r)$.
The term t'' can be constructed as follows: Let $Env = \{y_i = s_i\}_{i=1}^n$, and let $t'' := (\texttt{letrec}\ Env', x' = W'[(t''')]\ \texttt{in}\ t''')$ where Env' and t''' is Env and t', respectively, renamed by $\rho := \{x \mapsto x', y_i \mapsto y'_i \mid i = 1, \ldots, n\}$ and y'_i are fresh variables.

Proof of Proposition 2.20
This follows from Corollary H.4. □

References

Baader, Franz, & Nipkow, Tobias. (1998). *Term rewriting and all that.* Cambridge University Press.

Schmidt-Schauß, Manfred, Schütz, Marko, & Sabel, David. Safety of Nöcker's strictness analysis. *Journal of functional programming.*