A Call-by-Need Lambda-Calculus with Locally Bottom-Avoiding Choice: Context Lemma and Correctness of Transformations

David Sabel and Manfred Schmidt-Schauß

Institut für Informatik,
Fachbereich Informatik und Mathematik,
Johann Wolfgang Goethe-Universität,
Postfach 11 19 32, D-60054 Frankfurt, Germany,
{sabel,schauss}@ki.informatik.uni-frankfurt.de

Technical Report Frank-24
Research group for Artificial Intelligence and Software Technology,
Institut für Informatik,
Fachbereich Informatik und Mathematik
J.W.Goethe-Universität Frankfurt,

13. Jan. 2006

Abstract. We present a higher-order call-by-need lambda calculus enriched with constructors, `case`-expressions, recursive `letrec`-expressions, a `seq`-operator for sequential evaluation and a non-deterministic operator `amb`, which is locally bottom-avoiding. We use a small-step operational semantics in form of a normal order reduction. As equational theory we use contextual equivalence, i.e. terms are equal if plugged into an arbitrary program context their termination behaviour is the same. We use a combination of may- as well as must-convergence, which is appropriate for non-deterministic computations. We evolve different proof tools for proving correctness of program transformations. We provide a context lemma for may- as well as must- convergence which restricts the number of contexts that need to be examined for proving contextual equivalence. In combination with so-called complete sets of commuting and forking diagrams we show that all the deterministic reduction rules and also some additional transformations keep contextual equivalence. In contrast to other approaches our syntax as well as semantics does not make use of a heap for sharing expressions. Instead we represent these expressions explicitely via `letrec`-bindings.

Table of Contents

1 Introduction

1.1 Motivation

Higher-order lambda calculi with non-deterministic operators have been investigated in several works. Especially a non-deterministic choice operator, that chooses one of its arguments as result, but never diverges if one of the arguments is reducible to a value, is of relevance for modelling concurrent computation. It enables to express search algorithms in a naturally way [Hen80,BPLT02] or permits to implement a `merge` operator for event-driven systems like graphical user interfaces e.g [HC95] or functional operating systems [Hen82].

Such a nondeterministic operator is McCarthy's *amb* [McC63]. Its typical implementation is to start two concurrent (or parallel) processes, one for each of its arguments, and to choose the first one that terminates. Thus, *amb* is bottom-avoiding: let $\bot$ be an expression that cannot converge, then the expressions $(amb\ s\ \bot)$ and $(amb\ \bot\ s)$ are both equal to s. The bottom-avoidance of *amb* is only *local*, because its evaluation is independent of the surrounding context, e.g. the expression[1]

$$\texttt{if}\ (amb\ \texttt{True}\ \texttt{False})\ \texttt{then}\ \texttt{True}\ \texttt{else}\ \bot$$

may-diverge.

Together with constructs for explicit sharing and for sequential evaluation many other non-deterministic operators can be defined within the language. e.g. erratic-choice, locally demonic choice (see [SS92] for an overview of different non-deterministic operators) and also a parallel operator that evaluates both of its arguments in parallel and returns a pair of both values, e.g. [JH93] use such an operator. Hence, in this paper we introduce a higher-order lambda-calculus with a (weakly) typed `case`, constructors, `letrec`, `seq` and an operator `amb`. `letrec`-expressions are used for explicit sharing of terms as well as for describing recursive definitions. The binary operator `seq` evaluates to its second argument if and only if its first argument converges, otherwise the whole `seq`-expression diverges.

We will define a small-step operational semantics, which consists in rewriting terms. Moreover, our semantics is defined in form of a normal order reduction as a special strategy for finding a subterm for the next reduction. This strategy is deterministic for `amb`-free expressions and in the other case it nondeterministically chooses one of the concurrently possible reductions. An advantage of our approach is that we do not need to annotate the `amb`-expressions with resources (as e.g. in [Mor98]), which makes it possible to define a small-step reduction

[1] `if` b `then` s `else` t can be encoded in our calculus as $\texttt{case}_{Bool}\ b\ (\texttt{True} \to s)\ (\texttt{False} \to t)$

semantics directly on the expressions, without using a heap nor modifying the syntax by annotations before evaluation can be performed.

Based on normal order reduction we use as equational theory *contextual equivalence* (also known as observational equivalence) which equates two terms if their termination and additionally their non-termination behaviour in all program contexts is the same. It is well known that only regarding *may-convergence* is not sufficient for calculi with an `amb`-operator (e.g. see [Mor98]). Hence our equivalence will test for *must-convergence*, too. Our predicate for must-convergence is the logical inverse of may-divergence, where we only treat divergences that are called *strong* in [CHS05] (based on distinguishing between strong and weak divergence, introduced by [NC95]). I.e a term that has an infinite reduction but never loses the ability to converge is not seen as divergent. Our normal order reduction will not fulfil the fairness property for `amb`, i.e. if s or t converges, then normal order reduction not necessarily terminates while evaluating `amb` $s\ t$. But we will prove that fair evaluation induces the same notions of may- and must-convergence. [CHS05] have already shown this coincidence for a call-by-name calculus with `amb`.

Proving contextual equivalence directly seems to be very hard, since all program contexts need to be taken into account. On the other hand other methods, like using bisimulation for proving contextual equivalence have not been successful for call-by-need calculi with `amb` (see [Mor98] for a discussion). [Man05,MSS06] have shown that bisimulation can be used as a proof tool for a call-by-need calculus with non-recursive `let` and erratic choice. But there seems to be no obvious way to transfer this result to call-by-need calculi with recursive `let` and bottom-avoiding choice.

Thus, we will use the powerful technique of combining a context lemma for may- as well as must-convergence with complete sets of forking and commuting diagrams to prove the deterministic normal order reductions being correct program transformations, i.e. their application keeps contextual equivalence. This is of great value, since the reductions are formulated in a more general manner than needed for normal order reduction, and hence can be used as optimisations during program compilation, too. We will also show the correctness of some other program transformations which are used in compilers of functional programming languages for optimisation (e.g. [San95]).

We show that the equational theory defined by our normal order reduction is exactly the same as for fair normal order reductions. In passing we also show that resource annotations as in [Mor98] can be used to define a fair evaluation.

Another result is a standardisation theorem which states, that if there exists a sequence of transformations or reductions to a weak head normal form then there is also an evaluation in normal order to a weak head normal form. As second part the theorem states that if there exists a sequence of transformations inside surface contexts to a term that cannot converge, then normal order reduction can also reduce to such a term. Using the Standardisation Theorem we show that our `amb`-operator is indeed bottom-avoiding, i.e. `amb` $\Omega\ t \sim_c t$ and `amb` $t\ \Omega \sim_c t$, where Ω is a term that cannot converge. As final result we show that the contextual equivalence that only takes may-convergence into account is included in the contextual preorder for may- and must-convergence, i.e. $s \leq_c t \implies s \sim_c^{\downarrow} t$.

1.2 Related Work

To our knowledge the only two papers about call-by-need calculi with locally bottom-avoiding choice are [HM95,Mor98]. The work of [Mor98] is closely related to ours, since he considers also a call-by-need calculus with an *amb*-operator. His syntax is similar to ours, there are some small differences: He uses strict `let` expressions, where we use a `seq`-operator for implementing sequential evaluation. We use (weakly) typed `case`-expressions, whereas [Mor98] uses an untyped `case`. Moran uses contextual equivalence, but our equational theory differs from his, since the predicate for must-convergence is not the same (see Example 4.2). The advantage of our approach is that our small-step semantics does not use a heap and thus we are able to prove a context lemma for may- as well as must-convergence, whereas [Mor98] only provides a context lemma (based on improvement theory [MS99]) for may-convergence.

Our contextual preorder is similar to the one of [CHS05] for a call-by-name calculus with `amb`, since [CHS05] also test only for strong divergences. Call-by-name lambda calculi with `amb`-operators are also treated in [HM95,LM99,Mor98,Las05], but as [Mor98] did for their call-by-need calculus they also test for weak divergences in their contextual equivalence.

There is other work on call-by-need calculi with different choice operators, especially erratic choice. We compare some of them with our approach: The syntax of the used languages in [SSSS04,SS03] is very similar to ours, since both provide recursive `let`-expressions and also `case` and constructors and unrestricted applications. The last property does not hold for [MS99], since they allow only variables as arguments. Whereas [SSSS04] only use (may) convergence for the definition of contextual equivalence, [MS99,SS03] also use predicates for divergence. [SS03] uses a combination of contextual equivalence together with a trace semantics, where also only strong divergences are considered.

The proof technique of complete sets of commuting and forking diagrams has been introduced by [KSS98,Kut00] for a call-by-need lambda calculus with erratic choice and a non-recursive `let`. The same technique has also been used in [SS03,SSSS04,Man05] for their call-by-need calculi with erratic choice. The calculi in [Kut00,SS03,SSSS04,Man05] use a normal-order reduction as small-step semantics, where [SSSS04] is most similar to ours, whereas [MS99] use an abstract machine semantics.

Work on call-by-value calculi extended with bottom-avoiding choice has been done in [Las98].

1.3 Overview

In section 2 we introduce the calculus $\Lambda^{\mathtt{let}}_{\mathtt{amb}}$, and define the convergence predicates. In section 3 we introduce a fair evaluation strategy. In Section 4 we define the contextual preorder and contextual equivalence, we prove a context lemma, then we show some important properties of reduction rules that adjust `letrec`-environments and finally we introduce the notion of complete sets of commuting and forking diagrams. In sections 5, 6 and 7 we prove the correctness of all defined reduction rules and of some additional program transformations. In section 8 we prove the Standardisation Theorem and show that our `amb`-operator is indeed locally bottom-avoiding for a class of specific terms, that cannot converge. The section ends discussing some properties of the used contextual preorder that seem to be noteworthy. In the last section we conclude and give some directions for further research.

2 The Nondeterministic Call-by-Need Calculus $\Lambda^{\mathtt{let}}_{\mathtt{amb}}$

In this section we first introduce the syntax of the language of $\Lambda^{\mathtt{let}}_{\mathtt{amb}}$, then we define the reduction rules and the normal order reduction. After presenting encodings of other parallel and non-deterministic operators, we define different predicates for convergence and divergence.

2.1 The Syntax of the Language

The language of $\Lambda^{\mathtt{let}}_{\mathtt{amb}}$ is very similar to the abstract language used in [SSSS04] with the difference that $\Lambda^{\mathtt{let}}_{\mathtt{amb}}$ uses a bottom-avoiding choice-operator `amb` whereas [SSSS04] uses erratic choice. The language of the non-deterministic call-by-need lambda calculus of [Mor98] is also similar to ours, but we use an operator `seq` to provide sequential evaluation instead of strict `let` expressions and our `case`-expressions are weakly typed. In difference to the call-by-need calculus of [AFM+95] $\Lambda^{\mathtt{let}}_{\mathtt{amb}}$ provides constructors, weakly typed `case`-expressions and of course a nondeterministic `amb`-operator. The language we use also has only small differences (aside from the `amb`-operator) to the core language from [PM02] which is used in the Glasgow Haskell Compiler.

The language of $\Lambda^{\mathtt{let}}_{\mathtt{amb}}$ has the following syntax: There is a finite set of constructors which is partitioned into (nonempty) types. For every type T we denote the constructors as $c_{T,i}, i = 1, \dots, |T|$. Every constructor has an arity $\mathrm{ar}(c_{T,i}) \geq 0$.

The syntax for expressions E, case alternatives Alt and patterns Pat is defined by the following grammar:

$$
\begin{array}{llll}
E & ::= & V & (\textit{variable}) \\
 & \mid & (c_{T,i}\ E_1 \dots E_{\mathrm{ar}(c_{T,i})}) & (\textit{constructor application}) \\
 & \mid & (\mathtt{seq}\ E_1\ E_2) & (\mathtt{seq}\text{-}\textit{expression}) \\
 & \mid & (\mathtt{case}_T\ E\ Alt_1 \dots Alt_{|T|}) & (\mathtt{case}\text{-}\textit{expression}) \\
 & \mid & (E_1\ E_2) & (\textit{application}) \\
 & \mid & (\mathtt{amb}\ E_1\ E_2) & (\mathtt{amb}\text{-}\textit{expression}) \\
 & \mid & (\lambda\ V.E) & (\textit{abstraction}) \\
 & \mid & (\mathtt{letrec}\ V_1 = E_1, \dots V_n = E_n\ \mathtt{in}\ E) & (\mathtt{letrec}\text{-}\textit{expression}) \\
 & & \quad \text{where } n \geq 1 & \\
Alt & ::= & (Pat\ \rightarrow\ E) & (\mathtt{case}\text{-}\textit{alternative}) \\
Pat & ::= & (c_{T,i}\ V_1 \dots V_{\mathrm{ar}(c_{T,i})}) & (\textit{pattern})
\end{array}
$$

In addition to the presented grammar the following syntactic restrictions must hold for expressions:

- E, E_i are expressions and V, V_i are variables.
- Within a pattern the variables $V_1 \dots V_{\mathrm{ar}(c_{T,i})}$ are pairwise disjoint.
- In a $\texttt{case}_T$-expression, for every constructor $c_{T,i}, i = 1, \dots, |T|$, of type T, there is exactly one `case`-alternative.
- The constructs `case`, `seq`, `amb` and the constructors $c_{T,i}$ are only allowed when they occur fully saturated.
- The bindings of a `letrec`-expression form a mapping from variable names to expressions, in particular that means that the variables on the left hand side of the bindings are all distinct and that the bindings of `letrec`-expressions are commutative, i.e. `letrec`-expressions with permuted bindings are *syntactically equivalent*.
- `letrec` is recursive, i.e. in ($\texttt{letrec}\ x_1 = s_1, \dots, x_n = s_n\ \texttt{in}\ t$) the scope of x_i, $1 \leq i \leq n$, is $s_1, \dots, s_n$ and t.
- We use the distinct variable convention, i.e., all bound variables in expressions are assumed to be distinct, and free variables are distinct from bound variables. The reduction rules defined in later sections are assumed to implicitly rename bound variables in the result by α-renaming if necessary to obey this convention.

To abbreviate the notation, we will sometimes use:

- $(\texttt{case}_T\ E\ alts)$ instead of $(\texttt{case}_T\ E\ Alt_1 \dots Alt_{|T|})$,
- $(\texttt{letrec}\ Env\ \texttt{in}\ E)$ instead of $(\texttt{letrec}\ x_1 = E_1, \dots x_n = E_n\ \texttt{in}\ E)$. This will also be used freely for parts of the bindings.
- $(c_i\ \overrightarrow{s_i})$ instead of $(c_i\ s_1\ \dots\ s_{\mathrm{ar}(c_i)})$
- $\{x_{f(i)} = s_{g(i)}\}_{i=j}^{n}$ for the chain $x_{f(j)} = s_{g(j)}, x_{f(j+1)} = s_{g(j+1)}, \dots, x_{f(n)} = s_{g(n)}$, of `letrec`-bindings, where $f, g : \mathbb{N}_0 \to \mathbb{N}_0$
- We assume application to be left-associative, i.e. we write $(s_1\ s_2 \dots\ s_n)$ instead of $((s_1\ s_2)\ \dots\ s_n)$

Since $=$ is already used as a symbol in the syntax of the language, we use $\equiv$ to denote syntactical equivalence of expressions.

Definition 2.1. *A* value *is either an abstraction, or a constructor application.*

In the following we define different context classes and contexts, where we use different fonts for context classes and individual contexts. A context is a term with hole, we denote the hole with $[\cdot]$.

Definition 2.2 (Context). *The class $\mathcal{C}$ of all* contexts *is defined as follows.*

$$
\begin{aligned}
\mathcal{C} ::= \ & [\cdot] \mid (\mathcal{C}\ E) \mid (E\ \mathcal{C}) \mid (\texttt{seq}\ E\ \mathcal{C}) \mid (\texttt{seq}\ \mathcal{C}\ E) \mid \lambda x.\mathcal{C} \mid (\texttt{amb}\ \mathcal{C}\ E) \mid (\texttt{amb}\ E\ \mathcal{C}) \\
\mid\ & (\texttt{case}_T\ \mathcal{C}\ alts) \mid (\texttt{case}_T\ E\ Alt_1 \dots (Pat\ \to\ \mathcal{C}) \dots Alt_n) \\
\mid\ & (c_{T,i}\ E_1 \dots E_{i-1}\ \mathcal{C}\ E_{i+1} \dots E_{\mathrm{ar}(c)}) \\
\mid\ & (\texttt{letrec}\ x_1 = E_1, \dots, x_n = E_n\ \texttt{in}\ \mathcal{C}) \\
\mid\ & (\texttt{letrec}\ x_1 = E_1, \dots, x_{i-1} = E_{i-1}, x_i = \mathcal{C}, x_{i+1} = E_{i+1}, \dots, x_n = E_n\ \texttt{in}\ E)
\end{aligned}
$$

The main depth *of a context C is the depth of the hole in the context C. With $C_{\#i}$ we denote a context of main depth i. Let t be a term, C be a context, then $C[t]$ is the result of replacing the hole of C with term t.*

Let t, t_1, t_2 be terms and $C_1 \not\equiv C_2$ be contexts with $t \equiv C_1[t_1]$, $t \equiv C_2[t_2]$, then we say that C_1 and C_2 are disjoint for t *if there does not exist a context C_3 with $t \equiv C_1[C_3[t_2]]$ or $t \equiv C_2[C_3[t_1]]$.*

Definition 2.3 (Reduction Contexts). Reduction contexts $\mathcal{R}$ *and* weak reduction contexts $\mathcal{R}^-$ *are defined by the following grammar:*

$$\begin{aligned}\mathcal{R}^- ::= &\ [\cdot] \mid (\mathcal{R}^-\ E) \mid (\texttt{case}_T\ \mathcal{R}^-\ alts) \mid (\texttt{seq}\ \mathcal{R}^-\ E)\\ &\mid (\texttt{amb}\ \mathcal{R}^-\ E) \mid (\texttt{amb}\ E\ \mathcal{R}^-)\\ \mathcal{R}\ \ ::= &\ \mathcal{R}^- \mid (\texttt{letrec}\ Env\ \texttt{in}\ \mathcal{R}^-)\\ &\mid (\texttt{letrec}\ x_1 = \mathcal{R}_1^-, x_2 = \mathcal{R}_2^-[x_1], \ldots, x_j = \mathcal{R}_j^-[x_{j-1}], Env\ \texttt{in}\ \mathcal{R}^-[x_j])\\ &\quad \textit{where } j \geq 1 \textit{ and } \mathcal{R}^-, \mathcal{R}_i^-, i = 1, \ldots, j \textit{ are weak reduction contexts}\end{aligned}$$

For a term t with $t \equiv R^-[t_0]$ where R^- is a weak reduction context, we say R^- is maximal (for t) *if there is no larger non-disjoint weak reduction context for t, i.e. there is no weak reduction context R_1^- with $t \equiv R_1^-[t_1']$ where $t_1' \not\equiv t_0$ is a subterm of t_0.*

For a term t with $t \equiv R[t_0]$, we say R is a maximal reduction context (for t) *iff R is either*

- *a maximal weak reduction context, or*
- *of the form* (`letrec` $x_1 = E_1, \ldots, x_n = E_n$ `in` R^-) *where R^- is a maximal weak reduction context and $t_0 \not\equiv x_j$ for all $j = 1, \ldots, n$, or*
- *of the form* (`letrec` $x_1 = R_1^-, x_2 = R_2^-[x_1], \ldots, x_j = R_j^-[x_{j-1}], \ldots$ `in` $R^-[x_j]$)*, where $R_i^-, i = 1, \ldots, j$ are weak reduction contexts and R_1^- is a maximal weak reduction context for $R_1^-[t_0]$, and $t_0 \not\equiv y$ where y is a bound variable in t.*

Our definition of a maximal reduction context differs from the one in [SSSS04] in so far as such a context is only "maximal up to choice-points". As a consequence the maximal reduction context for a term t is not necessarily unique as the following example shows.

Example 2.4. For (`letrec` $x_2 = \lambda x.x, x_1 = x_2\ x_1, x_3 =$ (`amb` $(x_2\ x_1)\ y$) `in` (`amb` $x_1\ x_3$)) there exist the following maximal reduction contexts:

- (`letrec` $x_2 = [\cdot], x_1 = x_2\ x_1, x_3 =$ (`amb` $(x_2\ x_1)\ y$) `in` (`amb` $x_1\ x_3$))
- (`letrec` $x_2 = \lambda x.x, x_1 = x_2\ x_1, x_3 =$ (`amb` $(x_2\ x_1)\ [\cdot]$) `in` (`amb` $x_1\ x_3$))

The first maximal reduction context can be calculated by two different ways depending on which argument is chosen for the `amb`-expression in the `in`-expression of the `letrec`.

2.2 Reduction Rules

We define the reduction rules in a more general form than they will be used later for the normal order reduction. Thus the general rules can be used for partial evaluation and other compile time optimisations.

Definition 2.5 (Reduction Rules). *The reduction rules of $\Lambda^{\mathtt{let}}_{\mathtt{amb}}$ are defined in Fig. 1 and 2. We define the following unions of some reductions:*

$$
\begin{array}{ll}
(\mathsf{amb\text{-}c}) & := (\mathsf{amb\text{-}l\text{-}c}) \cup (\mathsf{amb\text{-}r\text{-}c}) \\
(\mathsf{lamb}) & := (\mathsf{lamb\text{-}l}) \cup (\mathsf{lamb\text{-}r}) \\
(\mathsf{amb\text{-}in}) & := (\mathsf{amb\text{-}l\text{-}in}) \cup (\mathsf{amb\text{-}r\text{-}in}) \\
(\mathsf{cp}) & := (\mathsf{cp\text{-}in}) \cup (\mathsf{cp\text{-}e}) \\
(\mathsf{amb\text{-}e}) & := (\mathsf{amb\text{-}l\text{-}e}) \cup (\mathsf{amb\text{-}r\text{-}e}) \\
(\mathsf{llet}) & := (\mathsf{llet\text{-}in}) \cup (\mathsf{llet\text{-}e}) \\
(\mathsf{amb}) & := (\mathsf{amb\text{-}l}) \cup (\mathsf{amb\text{-}r}) \\
(\mathsf{seq}) & := (\mathsf{seq\text{-}c}) \cup (\mathsf{seq\text{-}in}) \cup (\mathsf{seq\text{-}e}) \\
(\mathsf{amb\text{-}l}) & := (\mathsf{amb\text{-}l\text{-}c}) \cup (\mathsf{amb\text{-}l\text{-}in}) \cup (\mathsf{amb\text{-}l\text{-}e}) \\
(\mathsf{amb\text{-}r}) & := (\mathsf{amb\text{-}r\text{-}c}) \cup (\mathsf{amb\text{-}r\text{-}in}) \cup (\mathsf{amb\text{-}r\text{-}e}) \\
(\mathsf{case}) & := (\mathsf{case\text{-}c}) \cup (\mathsf{case\text{-}in}) \cup (\mathsf{case\text{-}e}) \\
(\mathsf{lll}) & := (\mathsf{llet}) \cup (\mathsf{lcase}) \cup (\mathsf{lapp}) \cup (\mathsf{lseq}) \cup (\mathsf{lamb})
\end{array}
$$

Reductions are denoted using an arrow with superscripts: e.g. $\xrightarrow{\mathsf{llet}}$. *To explicitly state the context in which a particular reduction is performed we annotate the reduction arrow with the context in which the reduction takes place. If no confusion arises, we omit the context at the arrow.*

The redex *of a reduction is the term as given on the left side of a reduction rule. We will also speak of the* inner redex*, which is the modified* `case`*-expression for (*case*)-reductions, the modified* `seq`*-expression for (*seq*)-reductions, the modified* `amb`*-expression for (*amb*)-reductions and the variable position which is replaced by rule (*cp*). Otherwise, it is the same as the redex.*

We denote the transitive closure of reductions by a $+$*, reflexive transitive closure by a* $*$*. We use uppercase words to denote (finite) sequences of reductions, e.g.* $\xrightarrow{RED}$.

We give a short comparison of our rules and the rules of the call-by-need calculus with recursion of [AFM$^+$95, Section 7.2]. The rule (lbeta) is the sharing-respecting variant of beta reduction, and is defined as rule (β_{need}) in [AFM$^+$95]. The rule (lll) adjusts `letrec`-environments, and is similar to the rules ($lift$), ($assoc$) and ($assoc_i$) of [AFM$^+$95] where we have more rules, since we have the constructs `case`, `seq` and `amb`. The rule (cp) is analogous to rule ($deref$) and ($deref_i$) of [AFM$^+$95] with the difference that we allow only abstractions to be copied, and do not copy variables. The consequence is that we need more variants for most of the reduction rules, since we explicitely follow the bindings during the reduction, instead of removing indirections. Another reason for having more rules than [AFM$^+$95] is that our syntax has `case`-, `seq`- and `amb`-expressions, which are not present for the call-by-need calculus of [AFM$^+$95]. The special

(lbeta)	$((\lambda x.s)\ r) \to (\texttt{letrec}\ x = r\ \texttt{in}\ s)$
(cp-in)	$(\texttt{letrec}\ x_1 = (\lambda x.s), \{x_i = x_{i-1}\}_{i=2}^m, Env\ \texttt{in}\ C[x_m])$ $\to (\texttt{letrec}\ x_1 = (\lambda x.s), \{x_i = x_{i-1}\}_{i=2}^m, Env\ \texttt{in}\ C[(\lambda x.s)])$
(cp-e)	$(\texttt{letrec}\ x_1 = (\lambda x.s), \{x_i = x_{i-1}\}_{i=2}^m, Env, y = C[x_m]\ \texttt{in}\ r)$ $\to (\texttt{letrec}\ x_1 = (\lambda x.s), \{x_i = x_{i-1}\}_{i=2}^m, Env, y = C[(\lambda x.s)]\ \texttt{in}\ r)$
(llet-in)	$(\texttt{letrec}\ Env_1\ \texttt{in}\ (\texttt{letrec}\ Env_2\ \texttt{in}\ r)) \to (\texttt{letrec}\ Env_1, Env_2\ \texttt{in}\ r)$
(llet-e)	$(\texttt{letrec}\ x_1 = s_1, \ldots, x_i = (\texttt{letrec}\ Env_2\ \texttt{in}\ s_i), \ldots, x_n = s_n\ \texttt{in}\ r)$ $\to (\texttt{letrec}\ x_1 = s_1, \ldots, x_i = s_i, \ldots, x_n = s_n, Env_2\ \texttt{in}\ r)$
(lapp)	$((\texttt{letrec}\ Env\ \texttt{in}\ t)\ x) \to (\texttt{letrec}\ Env\ \texttt{in}\ (t\ x))$
(lcase)	$(\texttt{case}_T\ (\texttt{letrec}\ Env\ \texttt{in}\ t)\ alts) \to (\texttt{letrec}\ Env\ \texttt{in}\ (\texttt{case}_T\ t\ alts))$
(lseq)	$(\texttt{seq}\ (\texttt{letrec}\ Env\ \texttt{in}\ s)\ t) \to (\texttt{letrec}\ Env\ \texttt{in}\ (\texttt{seq}\ s\ t))$
(lamb-l)	$(\texttt{amb}\ (\texttt{letrec}\ Env\ \texttt{in}\ s)\ t) \to (\texttt{letrec}\ Env\ \texttt{in}\ (\texttt{amb}\ s\ t))$
(lamb-r)	$(\texttt{amb}\ s\ (\texttt{letrec}\ Env\ \texttt{in}\ t)) \to (\texttt{letrec}\ Env\ \texttt{in}\ (\texttt{amb}\ s\ t))$
(seq-c)	$(\texttt{seq}\ v\ t) \to t$, if v is a value
(seq-in)	$(\texttt{letrec}\ x_1 = v, \{x_i = x_{i-1}\}_{i=2}^m, Env\ \texttt{in}\ C[(\texttt{seq}\ x_m\ t)])$ $\to (\texttt{letrec}\ x_1 = v, \{x_i = x_{i-1}\}_{i=2}^m, Env\ \texttt{in}\ C[t])$, if v is a value
(seq-e)	$(\texttt{letrec}\ x_1 = v, \{x_i = x_{i-1}\}_{i=2}^m, Env, y = C[(\texttt{seq}\ x_m\ t)]\ \texttt{in}\ r)$ $\to (\texttt{letrec}\ x_1 = v, \{x_i = x_{i-1}\}_{i=2}^m, Env, y = C[t]\ \texttt{in}\ r)$, if v is a value
(amb-l-c)	$(\texttt{amb}\ s\ v) \to v$, if v is a value
(amb-r-c)	$(\texttt{amb}\ v\ s) \to v$, if v is a value
(amb-l-in)	$(\texttt{letrec}\ x_1 = v, \{x_i = x_{i-1}\}_{i=2}^m, Env\ \texttt{in}\ C[(\texttt{amb}\ x_m\ s)])$ $\to (\texttt{letrec}\ x_1 = v, \{x_i = x_{i-1}\}_{i=2}^m, Env\ \texttt{in}\ C[x_m])$, if v is a value
(amb-r-in)	$(\texttt{letrec}\ x_1 = v, \{x_i = x_{i-1}\}_{i=2}^m, Env\ \texttt{in}\ C[(\texttt{amb}\ s\ x_m)])$ $\to (\texttt{letrec}\ x_1 = v, \{x_i = x_{i-1}\}_{i=2}^m, Env\ \texttt{in}\ C[x_m])$, if v is a value
(amb-l-e)	$(\texttt{letrec}\ x_1 = v, \{x_i = x_{i-1}\}_{i=2}^m, Env, y = C[(\texttt{amb}\ x_m\ t)]\ \texttt{in}\ r)$ $\to (\texttt{letrec}\ x_1 = v, \{x_i = x_{i-1}\}_{i=2}^m, Env, y = C[x_m]\ \texttt{in}\ r)$, if v is a value
(amb-r-e)	$(\texttt{letrec}\ x_1 = v, \{x_i = x_{i-1}\}_{i=2}^m, Env, y = C[(\texttt{amb}\ t\ x_m)]\ \texttt{in}\ r)$ $\to (\texttt{letrec}\ x_1 = v, \{x_i = x_{i-1}\}_{i=2}^m, Env, y = C[x_m]\ \texttt{in}\ r)$, if v is a value

Fig. 1. Reduction rules of the calculus

(case-c) for the case $\mathrm{ar}(c_{T,i}) = n \geq 1$: Let y_i be fresh variables, then
$(\texttt{case}_T\ (c_{T,i}\ \overrightarrow{t_i})\ \ldots((c_{T,i}\ \overrightarrow{y_i}) \to t)\ldots) \to (\texttt{letrec}\ \{y_i = t_i\}_{i=1}^n\ \texttt{in}\ t)$

(case-c) for the case $\mathrm{ar}(c_{T,i}) = 0 : (\texttt{case}_T\ c_{T,i}\ \ldots\ (c_{T,i} \to t)\ldots) \to t$

(case-in) for the case $\mathrm{ar}(c_{T,i}) = n \geq 1$: Let y_i be fresh variables, then
$\texttt{letrec}\ x_1 = (c_{T,i}\ \overrightarrow{t_i}), \{x_i = x_{i-1}\}_{i=2}^m, Env$
$\texttt{in}\ C[\texttt{case}_T\ x_m\ \ldots\ (c_{T,i}\ \overrightarrow{z_i} \to t)\ \ldots]$
$\to \texttt{letrec}\ x_1 = (c_{T,i}\ \overrightarrow{y_i}), \{y_i = t_i\}_{i=1}^n, \{x_i = x_{i-1}\}_{i=2}^m, Env$
$\texttt{in}\ C[(\texttt{letrec}\ \{z_i = y_i\}_{i=1}^n\ \texttt{in}\ t)]$

(case-in) for the case $\mathrm{ar}(c_{T,i}) = 0$:
$\texttt{letrec}\ x_1 = c_{T.i}, \{x_i = x_{i-1}\}_{i=2}^m, Env\ \texttt{in}\ C[\texttt{case}_T\ x_m\ \ldots(c_{T,i} \to t)\ldots]$
$\to \texttt{letrec}\ x_1 = c_{T,i}, \{x_i = x_{i-1}\}_{i=2}^m, Env\ \texttt{in}\ C[t]$

(case-e) for the case $\mathrm{ar}(c_{T,i}) = n \geq 1$: Let y_i be fresh variables, then
$\texttt{letrec}\, x_1 = (c_{T,i}\ \overrightarrow{t_i}), \{x_i = x_{i-1}\}_{i=2}^m, Env,$
$u = C[\texttt{case}_T\ x_m\ \ldots(c_{T,i}\ \overrightarrow{z_i} \to r_1)\ldots]$
$\texttt{in}\ r_2$
$\to \texttt{letrec}\ x_1 = (c_{T,i}\ \overrightarrow{y_i}), \{y_i = t_i\}_{i=1}^n, \{x_i = x_{i-1}\}_{i=2}^m, Env,$
$u = C[(\texttt{letrec}\ \{z_i = y_i\}_{i=1}^n\ \texttt{in}\ r_1)]$
$\texttt{in}\ r_2$

(case-e) for the case $\mathrm{ar}(c_{T,i}) = 0$:
$\texttt{letrec}\ x_1 = c_{T,i}, \{x_i = x_{i-1}\}_{i=2}^m, Env,$
$u = C[\texttt{case}_T\ x_m\ \ldots\ (c_i \to r_1)\ldots]\ \texttt{in}\ r_2$
$\to \texttt{letrec}\ x_1 = c_{T,i}, \{x_i = x_{i-1}\}_{i=2}^m, Env, u = C[r_1]\ \texttt{in}\ r_2$

Fig. 2. Reduction rules of the calculus (continued)

variants of (case) for constants are necessary to ensure not to introduce empty letrec-environments and hence the reduction rules generate only syntactically correct expressions.

2.3 Normal Order Reduction

Let R be a maximal reduction context for a term t and $t \equiv R[s]$. The normal order reduction applies a reduction rule of Definition 2.5 to s or to the direct superterm of s. For establishing understanding we start with describing how a position of a normal order redex can be reached by using a nondeterministic unwinding algorithm $\mathcal{UW}$. After that we will define the normal order reduction.

Let s be a term. If $s \equiv (\texttt{letrec}\ Env\ \texttt{in}\ s')$ apply uw to the pair $(s', (\texttt{letrec}\ Env\ \texttt{in}\ [\cdot]))$, otherwise apply uw to the pair $(s, [\cdot])$.

$\texttt{uw}((s\ t), R) \rightarrow \texttt{uw}(s, R[([\cdot]\ t)])$

$\texttt{uw}((\texttt{seq}\ s\ t), R) \rightarrow \texttt{uw}(s, R[(\texttt{seq}\ [\cdot]\ t)])$

$\texttt{uw}((\texttt{case}\ s\ alts), R) \rightarrow \texttt{uw}(s, R[(\texttt{case}\ [\cdot]\ alts)])$

$\texttt{uw}((\texttt{amb}\ s\ t), R) \rightarrow \texttt{uw}(s, R[(\texttt{amb}\ [\cdot]\ t)])$ **or** $\texttt{uw}(t, R[(\texttt{amb}\ s\ [\cdot])])$

$\texttt{uw}(x, (\texttt{letrec}\ x = s, Env\ \texttt{in}\ R^-)) \rightarrow \texttt{uw}(s, (\texttt{letrec}\ x = [\cdot], Env\ \texttt{in}\ R^-[x]))$

$\texttt{uw}(x, (\texttt{letrec}\ y = R^-, x = s, Env\ \texttt{in}\ t))$
$\quad \rightarrow \texttt{uw}(s, (\texttt{letrec}\ x = [\cdot], y = R^-[x], Env\ \texttt{in}\ t))$

$\texttt{uw}(s, R) \rightarrow (s, R)$ if no other rule is applicable

If a term contains a cycle, it may be the case that the algorithm does not terminate, e.g. for the term $(\texttt{letrec}\ x = y, y = x\ \texttt{in}\ x)$:

$$\begin{array}{l}\texttt{uw}(x, (\texttt{letrec}\ x = y, y = x\ \texttt{in}\ [\cdot])) \rightarrow \texttt{uw}(y, (\texttt{letrec}\ x = [\cdot], y = x\ \texttt{in}\ x)) \\ \rightarrow \texttt{uw}(x, (\texttt{letrec}\ x = y, y = [\cdot]\ \texttt{in}\ x)) \rightarrow \texttt{uw}(y, (\texttt{letrec}\ x = [\cdot], y = x\ \texttt{in}\ x)) \\ \rightarrow \ldots\end{array}$$

If the algorithm starting with term s terminates, the result is a pair (s', R), where R is a maximal reduction context for s and $R[s'] \equiv s$.

Definition 2.6. *We say the unwinding algorithm* visits *a subterm during execution, if there is a step, where the subterm is the first argument of the pair, to which* uw *is applied, or if the subterm is the whole term.*

Lemma 2.7. *During evaluation the unwinding algorithm visits only subterms that are in a reduction context. If $s \equiv R[s']$ then there exists an execution (by making the right decision if the algorithm crosses an* amb*-expression) that visits s'.*

We now define the normal order reduction. We apply a reduction rule by using a maximal reduction context for the term that should be reduced. It may be the case that the unwinding algorithm finds a maximal reduction context, but no reduction is possible. E.g. that happens, if the first argument of a `case`-expression has the wrong type, or if a free variable occurs inside the maximal reduction context.

Definition 2.8 (Normal Order Reduction). *Let t be an expression. Let R be a maximal reduction context for t, i.e. $t \equiv R[t']$ for some t'. The normal order reduction $\xrightarrow{\mathbf{no}}$ is defined by one of the following cases:*

If t' is a `letrec`*-expression* ($\texttt{letrec}\ Env_1\ \texttt{in}\ t''$)*, and $R \not\equiv [\cdot]$, then there are the following cases, where R_0 is a reduction context:*

1. $R \equiv R_0[(\texttt{seq}\ [\cdot]\ r)]$. *Reduce* $(\texttt{seq}\ t'\ r)$ *using rule* (lseq)*.*
2. $R \equiv R_0[([\cdot]\ r)]$. *Reduce* $(t'\ r)$ *using rule* (lapp)*.*
3. $R \equiv R_0[(\texttt{case}_T\ [\cdot]\ alts)]$. *Reduce* $(\texttt{case}_T\ t'\ alts)$ *using rule* (lcase)*.*
4. $R \equiv R_0[(\texttt{amb}\ [\cdot]\ s)]$. *Reduce* $(\texttt{amb}\ t'\ s)$ *using rule* (lamb-l)*.*
5. $R \equiv R_0[(\texttt{amb}\ s\ [\cdot])]$. *Reduce* $(\texttt{amb}\ s\ t')$ *using rule* (lamb-r)*.*
6. $R \equiv (\texttt{letrec}\ Env_2\ \texttt{in}\ [\cdot])$. *Reduce t using rule* (llet-in) *resulting in* $(\texttt{letrec}\ Env_1, Env_2\ \texttt{in}\ t'')$.
7. $R \equiv (\texttt{letrec}\ x = [\cdot], Env_2\ \texttt{in}\ t''')$. *Reduce t using* (llet-e) *resulting in* $(\texttt{letrec}\ x = t'', Env_1, Env_2\ \texttt{in}\ t''')$.

If t' is a value, then there are the following cases:

8. $R \equiv R_0[\texttt{case}_T\ [\cdot]\ \ldots]$, $t' \equiv (c_T\ \ldots)$*, i.e. the top constructor of t' belongs to type T. Then apply (case-c) to* $(\texttt{case}_T\ t'\ \ldots)$*.*
9. $R \equiv \texttt{letrec}\ x_1 = [\cdot], \{x_i = x_{i-1}\}_{i=2}^m, Env\ \texttt{in}\ R_0^-[\texttt{case}_T\ x_m\ (c_{T,j}\ \overrightarrow{y_i} \to r)\ alts]$ *and* $t' \equiv (c_{T,j}\ \overrightarrow{t_i})$*. Then apply* (case-in) *resulting in*
   $\texttt{letrec}\ x_1 = (c_{T,j}\ \overrightarrow{z_i}), \{x_i = x_{i-1}\}_{i=2}^m, \{z_i = t_i\}_{i=1}^n, Env$
   $\texttt{in}\ R_0^-[(\texttt{letrec}\ \{y_i = z_i\}_{i=1}^n\ \texttt{in}\ r)]$
10. $R \equiv \texttt{letrec}\ x_1 = [\cdot], \{x_i = x_{i-1}\}_{i=2}^m, Env\ \texttt{in}\ R_0^-[\texttt{case}_T\ x_m\ (c_{T,j} \to r)\ alts]$ *and* $t' \equiv c_{T,j}$*. Apply* (case-in) *resulting in* $\texttt{letrec}\ x_1 = c_{T,j}, \{x_i = x_{i-1}\}_{i=2}^m, Env\ \texttt{in}\ R_0^-[r]$.
11. $R \equiv \texttt{letrec}\ x_1 = [\cdot], \{x_i = x_{i-1}\}_{i=2}^m, Env, y = R_0^-[\texttt{case}_T\ x_m\ (c_{T,j}\ \overrightarrow{y_i} \to r)\ alts]\ \texttt{in}\, r'$,
    and $t' \equiv (c_{T,j}\ \overrightarrow{t_i})$*, and y is in a reduction context. Then apply* (case-e) *resulting in*
    $\texttt{letrec}\ \ x_1 = (c_{T,j}\ \overrightarrow{z_i}), \{x_i = x_{i-1}\}_{i=2}^m, \{z_i = t_i\}_{i=1}^n, Env,$
    $\quad y = R_0^-[(\texttt{letrec}\ \{y_i = z_i\}_{i=1}^n\ \texttt{in}\ r)]$
    $\texttt{in}\ r'$
12. $R \equiv \texttt{letrec}\ x_1 = [\cdot], \{x_i = x_{i-1}\}_{i=2}^m, Env, y = R_0^-[\texttt{case}_T\ x_m\ (c_{T,j} \to r)\ alts]\ \texttt{in}\ r'$*, and* $t' \equiv c_{T,j}$*, and y is in a reduction context. Then apply* (case-e) *resulting in*
    $\texttt{letrec}\ x_1 = c_{T,j}, \{x_i = x_{i-1}\}_{i=2}^m, Env, y = R_0^-[r]\ \texttt{in}\ r'$.
13. $R \equiv R_0[([\cdot]\ s)]$ *where R_0 is a reduction context and t' is an abstraction. Then apply* (lbeta) *to* $(t'\ s)$*.*

14. $R \equiv$ (letrec $x_1 = [\cdot], \{x_i = x_{i-1}\}_{i=2}^m, Env$ in $R_0^-[x_m]$) *where* R_0^- *is a weak reduction context and* t' *is an abstraction. Then apply (*cp-in*) and copy* t' *to the indicated position, resulting in* (letrec $x_1 = t', \{x_i = x_{i-1}\}_{i=2}^m, Env$ in $R_0^-[t']$).
15. $R \equiv$ (letrec $x_1 = [\cdot], \{x_i = x_{i-1}\}_{i=2}^m, Env, y = R_0^-[x_m]$ in r) *where* R_0^- *is a weak reduction context, y is in a reduction context and* t' *is an abstraction. Then apply (*cp-e*) resulting in* (letrec $x_1 = t', \{x_i = x_{i-1}\}_{i=2}^m, Env, y = R_0^-[t']$ in r).
16. $R \equiv R_0[$(seq $[\cdot]$ r)$]$. *Then apply (*seq-c*) to* (seq t' r) *resulting in* r.
17. $R \equiv$ (letrec $x_1 = [\cdot], \{x_i = x_{i-1}\}_{i=2}^m, Env$ in $R_0^-[$(seq x_m r)$]$), *and* t' *is a constructor application. Then apply (*seq-in*) resulting in* (letrec $x_1 = t', \{x_i = x_{i-1}\}_{i=2}^m, Env$ in $R_0^-[r]$).
18. $R \equiv$ (letrec $x_1 = [\cdot], \{x_i = x_{i-1}\}_{i=2}^m, Env, y = R_0^-[$(seq x_m r)$]$ in r') *where y is in a reduction context, and* t' *is a constructor application. Then apply (*seq-e*) resulting in* (letrec $x_1 = t', \{x_i = x_{i-1}\}_{i=2}^m, Env, y = R_0^-[r]$ in r').
19. $R \equiv R_0[$(amb $[\cdot]$ r)$]$. *Then apply (*amb-l-c*) to* (amb t' r).
20. $R \equiv R_0[$(amb r $[\cdot]$)$]$. *Then apply (*amb-r-c*) to* (amb r t').
21. $R \equiv$ (letrec $x_1 = [\cdot], \{x_i = x_{i-1}\}_{i=2}^m, Env$ in $R_0^-[$(amb x_m r)$]$), *and* t' *is a constructor application. Then apply (*amb-l-in*) resulting in* (letrec $x_1 = t', \{x_i = x_{i-1}\}_{i=2}^m, Env$ in $R_0^-[x_m]$).
22. $R \equiv$ (letrec $x_1 = [\cdot], \{x_i = x_{i-1}\}_{i=2}^m, Env$ in $R_0^-[$(amb r x_m)$]$), *and* t' *is a constructor application. Then apply (*amb-r-in*) resulting in* (letrec $x_1 = t', \{x_i = x_{i-1}\}_{i=2}^m, Env$ in $R_0^-[x_m]$).
23. $R \equiv$ (letrec $x_1 = [\cdot], \{x_i = x_{i-1}\}_{i=2}^m, Env, y = R_0^-[$(amb x_m r)$]$ in r') *where y is in a reduction context, and* t' *is a constructor application. Then apply (*amb-l-e*) resulting in* (letrec $x_1 = t', \{x_i = x_{i-1}\}_{i=2}^m, Env, y = R_0^-[x_m]$ in r').
24. $R \equiv$ (letrec $x_1 = [\cdot], \{x_i = x_{i-1}\}_{i=2}^m, Env, y = R_0^-[$(amb r x_m)$]$ in r') *where y is in a reduction context, and* t' *is a constructor application. Then apply (*amb-r-e*) resulting in* (letrec $x_1 = t', \{x_i = x_{i-1}\}_{i=2}^m, Env, y = R_0^-[x_m]$ in r').

The normal order redex *is defined as the subexpression to which the reduction rule is applied. This includes the* letrec*-expression that is mentioned in the reduction rules, for example in (*cp-e*).*

Some of our proofs will use induction on specific lengths of sequences of normal order reductions which are defined as follows:

Definition 2.9. *The number of reductions of a finite sequence RED consisting of normal order reductions is denoted with* $\mathtt{rl}(RED)$. *With* $\mathtt{rl}_{(\backslash a)}(RED)$ *we denote the number of non-a reductions in RED where a is a specific reduction.*

Example 2.10. Let $RED = \xrightarrow{no,seq}\xrightarrow{no,lapp}\xrightarrow{no,lbeta}\xrightarrow{no,llet}$. Then $\mathtt{rl}(RED) = 4$, and e.g. $\mathtt{rl}_{(\backslash lll)}(RED)$ is number of non-lll in RED, i.e. $\mathtt{rl}_{(\backslash lll)}(RED) = 2$.

2.4 Encoding of Non-deterministic and Parallel Operators

Fig. 3 shows the encoding of other non-deterministic or parallel operators within our language. The operator `par` activates the concurrent evaluation of its first

$$
\begin{array}{ll}
\texttt{par} & \equiv \lambda x.\lambda y.\texttt{amb}\ (\texttt{seq}\ x\ y)\ (\texttt{seq}\ y\ y) \\
\texttt{spar} & \equiv \lambda x.\lambda y.\texttt{amb}\ (\texttt{seq}\ x\ (\texttt{seq}\ y\ (\texttt{Pair}\ x\ y)))\ (\texttt{seq}\ y\ (\texttt{seq}\ x\ (\texttt{Pair}\ x\ y))) \\
\texttt{dchoice} & \equiv \lambda x.\lambda y.\texttt{amb}\ (\texttt{seq}\ x\ (\texttt{seq}\ y\ x))\ (\texttt{seq}\ y\ (\texttt{seq}\ x\ y)) \\
\texttt{choice} & \equiv \lambda x.\lambda y.((\texttt{amb}\ (\lambda z_1.x)\ (\lambda z_2.y))\ \texttt{True}) \\
\texttt{or} & \equiv \lambda x.\lambda y.(\texttt{amb}\ (\texttt{if}\ x\ \texttt{then True else}\ y)\ (\texttt{if}\ y\ \texttt{then True else}\ x)) \\
\texttt{merge} & \equiv \texttt{letrec}\ m = \lambda xs.\lambda ys.\texttt{amb}\ (\texttt{case}_{List}\ xs\ ([] \to ys)\ (z : zs \to z : m\ zs\ ys)) \\
 & \qquad\qquad\qquad\qquad\qquad\quad (\texttt{case}_{List}\ ys\ ([] \to xs)\ (z : zs \to z : m\ xs\ zs)) \\
 & \quad \texttt{in}\ m
\end{array}
$$

Fig. 3. Encoding of Operators

argument, but has the value of its second argument (Glasgow parallel Haskell has such an operator, see e.g. [THLP98]). The operator `spar` evaluates both arguments in parallel and returns the pair of values (e.g. this is the `par` operator suggested in [JH93]). The locally demonic `dchoice` non-deterministically chooses one of its arguments if and only if both arguments converge. Erratic `choice` non-deterministically chooses one if its arguments before evaluating the arguments. The parallel `or` is non-strict in both of its arguments, i.e. if one of the arguments evaluates to `True` then the `or`-expression evaluates to `True`. The `merge`-operator implements bottom-avoiding merge of two lists.

2.5 Convergence and Divergence

The notion of a weak head normal form will be required:

Definition 2.11. *An expression t is a* weak head normal form (WHNF) *if one of the following conditions holds:*

- *t is a value, or*
- *t is of the form* (`letrec` *Env* `in` v)*, where v is a value,*
- *or t is of the form* (`letrec` $x_1 = c_{T,i}\ t_1 \ldots t_{\mathrm{ar}(c_{T,i})}, x_2 = x_1, \ldots x_m = x_{m-1}, Env$ `in` x_m)

Lemma 2.12. *A WHNF has no normal order reduction.*

We now introduce predicates for may- and must-convergence as well as may- and must-divergence. Informally, a term *may-converge* if the it may evaluate to a WHNF using normal order reductions. If the opposite holds, a term *must-diverge*. I.e., there does not exist an evaluation in normal order that ends in a WHNF.

In difference to e.g. [Mor98] the existence of an infinite evaluation is *not* a sufficient criterion for *may-divergence*. In $\Lambda^{\texttt{let}}_{\texttt{amb}}$ a term *may-diverge* if there exists an evaluation that leads to a must-divergent term. Hence, it may happen that a term has infinite evaluation, but every contractum has the ability to converge. If there exists such an infinite evaluation for a term, [CHS05] say the term is *weakly divergent*.

The predicate for *must-convergence* is the opposite of *may-divergence*, hence weakly divergent terms are must-convergent in $\Lambda^{\texttt{let}}_{\texttt{amb}}$. The advantage of reasoning with the chosen predicates is that our semantics fulfils a fairness property, without explictly using scheduling of concurrent evaluations (see section 3). We now formally define the predicates:

Definition 2.13 (May- and Must-Convergence). *For a term t, we write $t\downarrow$ iff there exists a sequence of normal order reductions starting from t that ends in a WHNF, i.e.*

$$t\downarrow := \exists s : (t \xrightarrow{\mathbf{no},*} s \wedge s \text{ is a WHNF})$$

If $t\downarrow$, we say that t may-converge*. The set of finite sequences of normal order reductions of an expression t ending in a WHNF is denoted with $\mathcal{CON}(t)$, i.e.*

$$\mathcal{CON}(t) := \{RED \mid t \xrightarrow{RED} s, s \text{ is a WHNF}, \\ RED \text{ contains only normal order reductions}\}$$

We allow finite sequences of normal order reductions to be empty, i.e. if t is a WHNF then $\mathcal{CON}(t)$ contains an empty reduction sequence.

For a term t must-convergence *is defined as*

$$t\Downarrow := \forall s : (t \xrightarrow{\mathbf{no},*} s \implies s\downarrow)$$

Definition 2.14 (May and Must Divergence). *For a term t we write $t\Uparrow$ iff there exists no sequence of normal order reductions starting with t that ends in a WHNF. Then we say t* must-diverges*, i.e.*

$$t\Uparrow := \forall s : (t \xrightarrow{\mathbf{no},*} s \implies s \text{ is not a WHNF})$$

Let $\mathcal{NC}$ be the set of all terms that must-diverge i.e. $\mathcal{NC} = \{s \mid s\Uparrow\}$.

For a term t, we say t may-diverge*, denoted with $t\uparrow$ iff t may reduce to a term that must-diverge, i.e.*

$$t\uparrow := \exists s : (t \xrightarrow{\mathbf{no},*} s \wedge s\Uparrow)$$

For a term we define the set of all finite sequences of normal order reductions that lead to a term that must-diverge as follows:

$$\mathcal{DIV}(t) := \{RED \mid t \xrightarrow{RED} s, s\Uparrow, RED \text{ contains only normal order reductions}\}$$

We allow those sequences to be empty, i.e. if $t\Uparrow$ then $\mathcal{DIV}(t)$ contains an empty sequence.

The following lemma shows some relations between convergence and divergence.

Lemma 2.15. *Let t be a term, then $(t\Downarrow \iff \neg(t\uparrow))$, $(t\downarrow \iff \neg(t\Uparrow))$, $(t\Downarrow \implies t\downarrow)$ and $(t\Uparrow \implies t\uparrow)$.*

2.5.1 An Alternative Definition of Divergence Inspired from [Gor95] we give a co-inductive definition of the terms that must-diverge.

Definition 2.16. *Let*

$$\mathcal{MD}(X) := \{s \mid (\forall t : (s \xrightarrow{\mathbf{no}} t \implies t \in X)) \wedge s \textit{ is not a WHNF}\}$$

We define the set $\mathcal{BOT}$ as the greatest fixed point of $\mathcal{MD}$, i.e. $\mathcal{BOT} := \mathtt{gfp}(\mathcal{MD})$.

We inductively define the sets $\mathbf{md}^i$ for all $i \in \mathbb{N}_0$:

$$\begin{aligned}\mathbf{md}^0 &= \Lambda^{\mathtt{let}}_{\mathtt{amb}}\\ \mathbf{md}^i &= \mathcal{MD}(\mathbf{md}^{i-1})\end{aligned}$$

We now prove some properties of terms in $\mathcal{BOT}$.

Lemma 2.17. *The operator $\mathcal{MD}$ is monotonous w.r.t. set-inclusion.*

Proof. We need to show: $X \subseteq Y \implies \mathcal{MD}(X) \subseteq \mathcal{MD}(Y)$. Let $X \subseteq Y$ hold and there is $s \in \mathcal{MD}(X)$. We split into two cases:

- s has no normal order reduction. Then $s \in Y$ and hence $s \in \mathcal{MD}(Y)$
- For all t with $s \xrightarrow{\mathbf{no}} t$ follows that $t \in X$. Since $X \subseteq Y$, we have for all t: $t \in Y$. Thus, $s \in \mathcal{MD}(Y)$.

□

Lemma 2.18. $s \in \mathcal{BOT}$ *iff* $\forall j : s \in \mathbf{md}^j$.

Proof. Since $\mathcal{MD}$ is monotonous and set-inclusion forms a complete lattice, the greatest fixed point can be represented as $\mathtt{gfp}(\mathcal{MD}) = \bigcap_i \mathbf{md}^i$. □

Lemma 2.19. *Let s, t be terms with $s \xrightarrow{\mathbf{no}} t$. If $s \in \mathcal{BOT}$ then $t \in \mathcal{BOT}$.*

Proof. We use Lemma 2.18. From s in $\mathcal{BOT}$ we have, for all j: $s \in \mathbf{md}^j$. Since $s \xrightarrow{\mathbf{no}} t$, we have $\forall j > 0 : t \in \mathbf{md}^j$. It remains to prove $t \in \mathbf{md}^0$, but that holds by definition. □

Corollary 2.20. *If $s \xrightarrow{\mathbf{no},*} t$ with $s \in \mathcal{BOT}$. Then $t \in \mathcal{BOT}$*

Lemma 2.21. $\mathcal{BOT} = \mathcal{NC}$

Proof. $\mathcal{BOT} \subseteq \mathcal{NC}$: By definition of $\mathcal{BOT}$ we have that $\mathcal{BOT}$ does not contain terms in WHNF. Then Corollary 2.20 shows the claim.

$\mathcal{NC} \subseteq \mathcal{BOT}$: By co-induction it is sufficient to prove that $\mathcal{NC}$ is $\mathcal{MD}$-dense, i.e. $\mathcal{NC} \subseteq \mathcal{MD}(\mathcal{NC})$. Let $s \in \mathcal{NC}$ then we split into two cases:

- s has no normal order reduction. Since s cannot be in WHNF $s \in \mathcal{MD}(\mathcal{NC})$
- s has at least one normal order reduction. For every t with $s \xrightarrow{\mathbf{no}} t$ we have that t cannot have a terminating normal order reduction, otherwise there would $\mathcal{CON}(s)$ would not be empty. Since all such t have no terminating normal order reduction we have $\forall t : s \xrightarrow{\mathbf{no}} t \implies t \in \mathcal{NC}$ and since s is not in WHNF we have $s \in \mathcal{MD}(\mathcal{NC})$.

□

3 Fair Normal Order Reduction

In this section we show that the defined normal order reduction causes the same notions for may- and must-convergence as a fair reduction strategy does. Informally, a fair reduction strategy never does not reduce a normal order redex in infinite reduction sequences. Note, that for our normal order reduction this does not hold, e.g. the term $t \equiv (\texttt{amb}\ (\texttt{letrec}\ x = \lambda y.(y\ y)\ \texttt{in}\ (x\ x))\ \texttt{True})$ has an (amb-r)-redex, but normal order reduction may never reduce this redex. Nevertheless, t is must-convergent in our calculus. Hence, our notion of convergence already introduces a kind of fairness. A similar observation has already been made in [CHS05] for a call-by-name calculus with amb.

For implementing fair evaluation, we use resource annotations for amb-expressions:

Definition 3.1. *An* annotated variant *of a term s is s with all* amb*-expressions being annotated with a pair $\langle m, n\rangle$ of non-negative integers, denoted with* $\texttt{amb}_{\langle m,n\rangle}$*. The set of all annotated variants of a term s is denoted with $ann(s)$. With $ann_0(s)$ we denote the annotated variant of s with all pairs being $\langle 0, 0\rangle$. If s is an annotated variant of term t, then let $da(s) = t$.*

Informally, a (inner) redex within the subterm s (t, respectively) of the expression $(\texttt{amb}_{\langle m,n\rangle}\ s\ t)$ can only be reduced if resource m (n, respectively) is non-zero. Any reduction inside s decreases the annotation m by 1. Fairness emerges from the fact that resources can only be increased if both resources m and n are 0, and the increase for both resources must be strictly greater than 0.

We extend the notions of contexts and WHNFs to annotated variants:

Definition 3.2. *If C is an annotated variant of a term with a hole, then C is a context iff $da(s)$ is a context. An annotated variant s of a term is a WHNF iff $da(t)$ is a WHNF.*

We now give a description of a non-deterministic unwinding algorithm $\mathcal{UWF}$ that leads to fair evaluation. The algorithm performs four tasks: It finds a position where a normal order reduction can be applied, it decreases the annotations for the path that leads to this position and if necessary it performs scheduling by increasing the annotations. Furthermore, the unwinding algorithm decreases the annotation for a subterm that cannot be reduced, since it has a typing error (e.g. $\texttt{case}_{List}\ \texttt{True}\ \ldots$) or it is a term with a blackhole, e.g. $(\texttt{letrec}\ x = x\ \texttt{in}\ x)$. [Mor98] uses an additional reduction rule for this cases. We decrease the annotation by executing the unwinding algorithm again with another variant of the same term, where annotations are decreased.

Let s be an annotated variant of a term. If $s \equiv (\texttt{letrec}\ Env\ \texttt{in}\ s')$ then apply uwf to the pair $(s', (\texttt{letrec}\ Env\ \texttt{in}\ [\cdot]))$, otherwise apply uwf to the pair $(s, [\cdot])$.

$\texttt{uwf}((s\ t), R) \to \texttt{uwf}(s, R[([\cdot]\ t)])$

$\texttt{uwf}((\texttt{seq}\ s\ t), R) \to \texttt{uwf}(s, R[(\texttt{seq}\ [\cdot]\ t)])$

$\texttt{uwf}((\texttt{case}\ s\ alts), R) \to \texttt{uwf}(s, R[(\texttt{case}\ [\cdot]\ alts)])$

$\texttt{uwf}((\texttt{amb}_{\langle m+1,n\rangle}\ s\ t), R) \to \texttt{uwf}(s, R[(\texttt{amb}_{\langle m,n\rangle}\ [\cdot]\ t)])$

$\texttt{uwf}((\texttt{amb}_{\langle m,n+1\rangle}\ s\ t), R) \to \texttt{uwf}(t, R[(\texttt{amb}_{\langle m,n\rangle}\ s\ [\cdot])])$

$\texttt{uwf}((\texttt{amb}_{\langle 0,0\rangle}\ s\ t), R) \to \texttt{uwf}((\texttt{amb}_{\langle m,n\rangle}\ s\ t), R),$ where $m, n > 0$

$\texttt{uwf}(x, (\texttt{letrec}\ x = s, Env\ \texttt{in}\ R^-)) \to \texttt{uwf}(s, (\texttt{letrec}\ x = [\cdot], Env\ \texttt{in}\ R^-[x]))$

$\texttt{uwf}(x, (\texttt{letrec}\ y = R^-, x = s, Env\ \texttt{in}\ t))$
$\to \texttt{uwf}(s, (\texttt{letrec}\ x = [\cdot], y = R^-[x], Env\ \texttt{in}\ t))$

$\texttt{uwf}(s, R) \to (s, R)$ if no other rule is applicable

Now, it may happen that the unwinding algorithms loops, e.g. for the subterm $(\texttt{letrec}\ x = y, y = x\ \texttt{in}\ x)$. Another case is that the algorithm terminates, but for the maximal reduction context found no normal order reduction is applicable, e.g. this holds for the pair $((\texttt{letrec}\ x = x\ \texttt{in}\ x), \texttt{amb}_{\langle m,n\rangle}\ [\cdot]\ s)$. In both cases the annotations for all arguments of amb-expressions that have been visited are decreased and then with the modified annotations, the algorithms performs a new search: we assume that the fair unwinding algorithm has a loop detection which starts a new search when visiting a binding for a second time during execution, i.e. we add the rule:

$\texttt{uwf}(x, (\texttt{letrec}\ y = R^-, x = s, Env\ \texttt{in}\ t))$
$\to$ execute $\mathcal{UWF}$ with $(\texttt{letrec}\ y = R^-[x], x = s, Env\ \texttt{in}\ t)$ if there was a preceding step with intermediate result $\texttt{uwf}(s', (\texttt{letrec}\ Env', x = [\cdot]\ \texttt{in}\ t'))$.

Moreover, if $\mathcal{UWF}$ terminates with result (s, R), but no normal order reduction is applicable to $R[s]$ (using R as maximal reduction context for determining the normal order reduction) then execute $\mathcal{UWF}$ with $R[s]$.

Lemma 3.3. *If $s \in ann(t)$ and $\texttt{uwf}(s', R)$ is an intermediate result of $\mathcal{UWF}$, then $R[s'] \in ann(t)$. The same holds for the resulting pair of $\mathcal{UWF}$.*

Proof. This holds, since $\mathcal{UWF}$ only changes the annotations. □

Lemma 3.4. *Let s be an annotated variant and (s', R) be an result of executing $\mathcal{UWF}$ starting with s. Then there exists an execution of the unwinding algorithm $\mathcal{UW}$ for $da(s)$ that has result $(da(s'), da(R'))$.*

Proof. Use the steps of the last search of the execution of $\mathcal{UWF}$. By replacing all $\texttt{amb}_{\langle m,n\rangle}$ constructs with amb, we can perform every step with the algorithm $\mathcal{UW}$, too. □

Fair normal order reduction $\xrightarrow{\mathbf{fno}}$ on annotated variants is defined as follows:

Definition 3.5 (Fair Normal Order Reduction). *Let s be an annotated variant of a term. If s is a WHNF, then no fair normal order reduction is possible. Otherwise, execute $\mathcal{UWF}$ starting with s. If the execution terminates with (s'', R), then apply a normal order reduction to $R[s'']$ with maximal reduction context R, where annotations are inherited like labels in labeled reduction (see [Bar84]). Let the result be t. Then $s \xrightarrow{\mathbf{fno}} t$.*

A fair normal order reduction sequence (denoted with F as subscript) is a sequence consisting of fair normal order reductions. Note, that it may happen, that the execution of the normal order reduction does not terminate, then there is no fair normal order reduction defined.

Definition 3.6. *Fair May- and must-convergence and -divergence for annotated variants is defined as:*

$$\begin{aligned}
t{\downarrow_F} &:= \exists s : (t \xrightarrow{\mathbf{fno},*} s \wedge s \text{ is a WHNF})\\
t{\Downarrow_F} &:= \forall s : (t \xrightarrow{\mathbf{fno},*} s \implies s{\downarrow_F})\\
t{\Uparrow_F} &:= \forall s : (t \xrightarrow{\mathbf{fno},*} s \implies s \text{ is not a WHNF})\\
t{\uparrow_F} &:= \exists s : (t \xrightarrow{\mathbf{fno},*} s \wedge s{\Uparrow_F})
\end{aligned}$$

A term t fair may-converge *(denoted with $t{\downarrow_F}$) iff $ann_0(t){\downarrow_F}$, a term t* fair must-converge *(denoted with $t{\Downarrow_F}$) iff $ann_0(t){\Downarrow_F}$.*

Lemma 3.7. *If $s \xrightarrow{\mathbf{fno}} t$, then $da(s) \xrightarrow{\mathbf{no}} da(t)$.*

Proof. Follows from Lemma 3.4 and the definition of fair normal order reduction. □

Corollary 3.8. *Let s be a term, $s' \in (ann(s))$ and $s'{\downarrow_F}$, then $s{\downarrow}$.*

Lemma 3.9. *If $s \xrightarrow{\mathbf{no},*} t$, then there exists $t' \in ann(t)$ with with $ann_0(s) \xrightarrow{\mathbf{fno},*} t'$.*

Proof. Let $s \xrightarrow{RED} t$, where RED is a sequence of normal order reductions. If RED is empty then the claim follows with $t' = ann_0(t)$. Otherwise, let $RED = red_1 \dots red_k$. From the definition of the normal order reduction it follows, that before every red_i there is an execution of $\mathcal{UW}$ that finds a maximal reduction context, which is used to apply red_i. Let s_i be the term to which red_i is applied, and let (R_i, s'_i) be the corresponding maximal reduction context and term in the hole, i.e. $R_i[s'_i] \equiv s_i$. Further, let $e_1, \dots, e_k$ be the executions of $\mathcal{UW}$ that terminate with result (s'_i, R_i). Obviously, it is sufficient to show that for every red_i there is an execution of $\mathcal{UWF}$, that terminates with (s''_i, R''_i) where $R''_i[s''_i] \in ann(s_i)$. Now, let m be the sum of all steps $\mathtt{uw}((\mathtt{amb}\ t_1\ t_2), R) \to \dots$ that happens in the executions $e_1, \dots, e_k$. Now for every e_i build an execution e'_i of $\mathcal{UWF}$, by performing the corresponding steps, with the difference that when arriving at $\mathtt{uwf}(\mathtt{amb}_{\langle 0,0\rangle}, R')$ then insert the step $\mathtt{uwf}(\mathtt{amb}_{\langle 0,0\rangle}, R') \to \mathtt{uwf}(\mathtt{amb}_{\langle m,m\rangle}, R')$. Now it can be seen easily that all other steps are possible, since there are always enough resources to apply the corresponding steps. Note, that since there is always a normal order reduction possible using (R_i, s'_i), $\mathcal{UWF}$ never needs to restart. ☐

Corollary 3.10. *Let s be a term with $s\downarrow$, then $s\downarrow_F$.*

Proposition 3.11. *For all terms t: $t\downarrow$ iff $t\downarrow_F$.*

Proof. Follows from Corollaries 3.10 and 3.8. ☐

Lemma 3.12. *Let s be term, $s' \in ann(s)$ and $s\Uparrow$, then $s'\Uparrow_F$.*

Proof. Assume the claim is false, then $s'\downarrow_F$ and hence there exists a sequence of fair normal order reductions, that lead from s' to a WHNF, but then with Lemma 3.7 it follows, that $s\downarrow$. Hence a contradiction. ☐

Lemma 3.13. *Let s be a term with $s\uparrow$ then $ann_0(s)\uparrow_F$.*

Proof. Let $RED \in \mathcal{DIV}(s)$, then $s \xrightarrow{RED} t$ and $t\Uparrow$. Lemma 3.9 shows that $ann_0(s) \xrightarrow{\mathbf{fno},*} t'$, with $t' \in ann(t)$. From Lemma 3.12 we have $t'\Uparrow_F$ and thus $ann_0(s)\Uparrow_F$. ☐

For the remaining part, i.e. to show that if s must-converge then $ann_0(s)\Downarrow_F$ we will use co-inductive definitions of must-convergence. Let $\mathcal{ANN}(\Lambda^{\mathtt{let}}_{\mathtt{amb}}) = \{s \mid s' \in \Lambda^{\mathtt{let}}_{\mathtt{amb}}, s \in ann(s')\}$.

Definition 3.14. *The operators $\mathcal{MC} : \Lambda^{\mathtt{let}}_{\mathtt{amb}} \to \Lambda^{\mathtt{let}}_{\mathtt{amb}}$ and $\mathcal{MC}_F : \mathcal{ANN}(\Lambda^{\mathtt{let}}_{\mathtt{amb}}) \to \mathcal{ANN}(\Lambda^{\mathtt{let}}_{\mathtt{amb}})$ are defined as:*

$$\mathcal{MC}(X) = \left\{ s \in X \;\middle|\; \begin{array}{c} (\forall s' : s \xrightarrow{\mathbf{no}} s' \implies s'\downarrow \wedge s' \in X) \\ \wedge \\ (s \textit{ has no normal order reduction} \implies s \textit{ is a WHNF}) \end{array} \right\}$$

$$\mathcal{MC}_F(X) = \left\{ s \in X \;\middle|\; \begin{array}{c} (\forall s' : s \xrightarrow{\mathbf{fno}} s' \implies s'\downarrow_F \wedge s' \in X) \\ \wedge \\ (s \textit{ has no normal order reduction} \implies s \textit{ is a WHNF}) \end{array} \right\}$$

Lemma 3.15. *The operators $\mathcal{MC}$ and $\mathcal{MC}_F$ are monotonous w.r.t. set inclusion.*

Proof. We only show the property for $\mathcal{MC}$, the proof for $\mathcal{MC}_F$ is analogous. We need to show: $X \subseteq Y \implies \mathcal{MC}(X) \subseteq \mathcal{MC}(Y)$. Let $X \subseteq Y$ hold and there is $s \in \mathcal{MC}(X)$. We split into two cases:

- s has no normal order reduction and is in WHNF. Then $s \in X$ and hence $s \in Y$ and finally $s \in \mathcal{MC}(Y)$.
- For all t with $s \xrightarrow{\mathbf{no}} t$ follows that $t\downarrow, t \in X$. Since $X \subseteq Y$, we have for all t: $t \in Y$. Thus, $s \in \mathcal{MC}(Y)$.

☐

Definition 3.16. *Let* $\mathbf{mc}^i$ *and* $\mathbf{mc}^i_f$ *be inductively defined as*

$$\mathbf{mc}^0 = \Lambda^{\mathtt{let}}_{\mathtt{amb}} \qquad \mathbf{mc}^0_f = \mathcal{ANN}(\Lambda^{\mathtt{let}}_{\mathtt{amb}})$$
$$\mathbf{mc}^i = \mathcal{MC}(\mathbf{mc}^{i-1}) \qquad \mathbf{mc}^i_f = \mathcal{MC}(\mathbf{mc}^{i-1}_f)$$

Lemma 3.17. *The greatest fixed point of* $\mathcal{MC}$ *or* $\mathcal{MC}_F$*, respectively can be presented as the infinite intersection of all* $\mathbf{mc}^i$ *or* $\mathbf{mc}^i_f$*, respectively. I.e.*

- $s \in \mathtt{gfp}(\mathcal{MC})$ *iff* $\forall j : s \in \mathbf{mc}^j$.
- $s \in \mathtt{gfp}(\mathcal{MC}_F)$ *iff* $\forall j : s \in \mathbf{mc}^j_F$.

Proof. Since $\mathcal{MC}$ and $\mathcal{MC}_F$ are monotonous and set-inclusion forms a complete lattice, their greatest fixed points can be represented as $\bigcap_i \mathbf{mc}^i$, or $\bigcap_i \mathbf{mc}^i_F$, respectively. □

Lemma 3.18. *Let* s *be a term with* $s{\Downarrow}, s \xrightarrow{\mathbf{no}} s'$ *then* $s'{\Downarrow}$.

Proof. Assume the claim is false, i.e. $s{\Downarrow}, s \xrightarrow{\mathbf{no}} s'$ but $s'{\uparrow}$, then there exists t' with $s' \xrightarrow{\mathbf{no},*} t'$ and $t'{\Uparrow}$. By combining the reductions, we have $s \xrightarrow{\mathbf{no},*} t'$ and $\neg(t'{\downarrow})$. Hence, we have a contradiction. □

Using the same reasoning we can show:

Lemma 3.19. *Let* $s \in \mathcal{ANN}(\Lambda^{\mathtt{let}}_{\mathtt{amb}})$ *with* $s{\Downarrow_F}, s \xrightarrow{\mathbf{fno}} s'$ *then* $s'{\Downarrow_F}$.

Lemma 3.20. *Let* s *be a term with* $s \in \mathtt{gfp}(\mathcal{MC})$ *and* $s \xrightarrow{\mathbf{no}} s'$*. Then* $s' \in \mathtt{gfp}(\mathcal{MC})$.

Proof. From $s \in \mathtt{gfp}(\mathcal{MC})$ follows for all i: $s \in \mathcal{MC}^i(\Lambda^{\mathtt{let}}_{\mathtt{amb}})$ and thus $\forall i \geq 0 : s' \in \mathcal{MC}^{i-1}(\Lambda^{\mathtt{let}}_{\mathtt{amb}})$. □

Using the same arguments we can show:

Lemma 3.21. *Let* $s \in \mathcal{ANN}(\Lambda^{\mathtt{let}}_{\mathtt{amb}})$ *with* $s \in \mathtt{gfp}(\mathcal{MC}_F)$ *and* $s \xrightarrow{\mathbf{fno}} s'$*. Then* $s' \in \mathtt{gfp}(\mathcal{MC}_F)$.

Lemma 3.22. *For all* $s : s{\Downarrow}$ *iff* $s \in \mathtt{gfp}(\mathcal{MC})$

Proof. We show by induction on i, that $\forall s : s{\Downarrow} \implies s \in \mathbf{mc}^i$. Obviously, the claim holds for $i = 0$. For $i > 0$, by the definition of must-convergence together with Lemma 3.18 it follows that all $s{\Downarrow} \implies (\forall s' : s \xrightarrow{\mathbf{no}} s', s'{\Downarrow})$. Hence, these s' are all in $\mathbf{mc}^{i-1}$. The second condition that, if s is irreducible then s is a WHNF, follows obviously.

Now we show the other direction. Let $s \in \mathtt{gfp}(\mathcal{MC})$. Let $s \xrightarrow{\mathbf{no},k} t$, with $k \geq 0$. We show $t{\downarrow}$ by induction on k. If $k = 0$, then we need to show $s{\downarrow}$. Since $s \in \mathtt{gfp}(\mathcal{MC})$ either s is a WHNF and thus $s{\downarrow}$, or $s \xrightarrow{\mathbf{no}} s'$ and $s'{\downarrow}$ and hence $s{\downarrow}$. If $k > 0$, then let $s \xrightarrow{\mathbf{no}} s_1 \xrightarrow{\mathbf{no}} \ldots \xrightarrow{\mathbf{no}} s_k$. From Lemma 3.20 we have $s_1 \in \mathtt{gfp}(\mathcal{MC})$. Using the induction hypothesis we have $t{\downarrow}$. □

Lemma 3.23. *For all* $s \in \mathcal{ANN}(\Lambda^{\texttt{let}}_{\texttt{amb}}) : s{\Downarrow}_F$ *iff* $s \in \texttt{gfp}(\mathcal{MC}_F)$

Proof. We show by induction on i, that $\forall s \in \mathcal{ANN}(\Lambda^{\texttt{let}}_{\texttt{amb}}) : s{\Downarrow}_F \implies s_F \in \mathbf{mc}^i_F$. Obviously, the claim holds for $i = 0$. For $i > 0$, by the definition of must-convergence together with Lemma 3.19 it follows that all $s{\Downarrow}_F \implies (\forall s' : s \xrightarrow{\mathbf{fno}} s', s'{\Downarrow}_F)$. Hence these s' are all in $\mathbf{mc}^{i-1}_F$. The second condition that, if s is irreducible then s is a WHNF, follows obviously.

Now we show the other direction. Let $s \in \texttt{gfp}(\mathcal{MC}_F)$. Let $s \xrightarrow{\mathbf{fno},k} t$, with $k \geq 0$. We show $t{\downarrow}_F$ by induction on k. If $k = 0$, i.e. we need to show $s{\downarrow}_F$. Since $s \in \texttt{gfp}(\mathcal{MC}_F)$ either s is a WHNF and thus $s{\downarrow}$, or $s \xrightarrow{\mathbf{fno}} s'$ and $s'{\downarrow}$ and hence $s{\downarrow}$. If $k > 0$, then let $s \xrightarrow{\mathbf{fno}} s_1 \xrightarrow{\mathbf{fno}} \ldots \xrightarrow{\mathbf{fno}} s_k$. From Lemma 3.21 we have $s_1 \in \texttt{gfp}(\mathcal{MC}_{\texttt{F}})$. Using the induction hypothesis we have $t{\downarrow}_F$. □

Lemma 3.24. *Let* $s \in \texttt{gfp}(\mathcal{MC})$ *and* $s' \in ann(s)$*. Then* $s' \in \texttt{gfp}(\mathcal{MC}_F)$*.*

Proof. We show by induction on i that $s' \in \mathbf{mc}^i_F$ if $da(s') \in \texttt{gfp}(\mathcal{MC})$. If $i = 0$ then this is obvious. If $i > 0$ then for showing $s' \in \mathbf{mc}^i_F$ it is sufficient to prove:

1. $s' \in \mathbf{mc}^{i-1}_F$.
2. If s' has no fair normal order reduction then s' is a WHNF.
3. $\forall s'' : s' \xrightarrow{\mathbf{fno}} s'' \implies s''{\downarrow}_F \wedge s'' \in \mathbf{mc}^{i-1}_F$.

1 follows from the induction hypothesis. 2 holds, since $s' \in \mathbf{mc}_{i-1}$ and thus if s' has no fair normal order reduction, it must be a WHNF. For proving 3 let $s' \xrightarrow{\mathbf{fno}} s''$. Since $s' \in \mathbf{mc}^{i-1}_F$ we have $s''{\downarrow}_F$. With Lemma 3.7 we have $da(s') \xrightarrow{\mathbf{no}} da(s'')$. Since $da(s') \in \texttt{gfp}(\mathcal{MC})$, using Lemma 3.20 we have $da(s'') \in \texttt{gfp}(\mathcal{MC})$. By the induction hypothesis we have $s'' \in \mathbf{mc}^{i-1}_F$. □

Corollary 3.25. *Let* s *be a term with* $s{\Downarrow}$*, then* $s{\Downarrow}_F$*.*

Theorem 3.26. *For all terms* t*: (*$t{\downarrow}$ *iff* $t{\downarrow}_F$*) and (*$t{\Downarrow}$ *iff* $t{\Downarrow}_F$*).*

Proof. Follows from Proposition 3.11, Lemma 3.13 and Corollary 3.25. □

4 Contextual Equivalence and Proof Tools

4.1 Preorders for May- and Must-Convergence

We define different preorders resulting in a combined preorder which tests for may-convergence and must-convergence in all contexts. Contextual equivalence is then the symmetrisation of the combined preorder.

Definition 4.1. *Let* s, t *be terms. We define the following relations:*

$$s \leq_c^{\downarrow} t \text{ iff } (\forall C \in \mathcal{C} :\ C[s]{\downarrow} \Rightarrow C[t]{\downarrow})$$

$$s \leq_c^{\Downarrow} t \text{ iff } (\forall C \in \mathcal{C} :\ C[s]{\Downarrow} \Rightarrow C[t]{\Downarrow})$$

$$s \leq_c t \text{ iff } s \leq_c^{\downarrow} t \wedge s \leq_c^{\Downarrow} t$$

The contextual equivalence is then defined as:

$$s \sim_c t \text{ iff } s \leq_c t \wedge t \leq_c s$$

Note, that for all three preorders $C[s]$ may be an open term. Our contextual equivalence is the same as [CHS05] use for their call-by-name calculus where so-called weak divergences are not considered. This is in contrast to [HM95,Mor98,Las05] where may-divergence includes terms that have an infinite normal order reduction but never lose the ability to converge. A consequence is that our equational theory is different from the one of [Mor98]:

Example 4.2. The example of [CHS05, p.453] is applicable to our calculus. Let the identity function $\mathbf{I}$, a fixed-point operator $\mathbf{Y}$ and a must-divergent term Ω be defined as

$$\begin{aligned}\mathbf{I} &\equiv \lambda x.x\\ \mathbf{Y} &\equiv (\texttt{letrec}\ y = \lambda f.(f\ (y\ f))\ \texttt{in}\ y)\\ \Omega &\equiv (\texttt{letrec}\ x = x\ \texttt{in}\ x)\end{aligned}$$

then

$$\mathbf{I} \sim_c \mathbf{Y}\ (\lambda x.(\texttt{choice}\ x\ \mathbf{I})) \not\sim_c \texttt{choice}\ \Omega\ \mathbf{I}.$$

Now, we consider a contextual equivalence $\sim_M$ that is the same as $\sim_c$ with the only difference that a term t must-converge iff all sequences of normal order reductions that start with t are finite and lead to a WHNF. The relation $\sim_M$ is analogous to the contextual equivalence used by [Mor98]. Then

$$\mathbf{I} \not\sim_M \mathbf{Y}\ (\lambda x.(\texttt{choice}\ x\ \mathbf{I})) \sim_M \texttt{choice}\ \Omega\ \mathbf{I}.$$

A well-known property (see [LLP05]) for lambda calculi with locally bottom-avoiding choice holds for $\Lambda^{\texttt{let}}_{\texttt{amb}}$, too:

Example 4.3. Ω is not least w.r.t. $\leq_c$. This follows, since for the context $C \equiv (\texttt{amb}\ (\lambda x.\lambda y.x)\ [\cdot])\ \Omega)$ the term $C[\mathbf{I}]$ may-diverge whereas $C[\Omega]$ must-converge, hence $\Omega \not\leq_c \mathbf{I}$.

A *precongruence* $\preceq$ is a preorder on expressions, such that $s \preceq t \Rightarrow C[s] \preceq C[t]$ for all contexts C. A *congruence* is a precongruence that is also an equivalence relation.

Proposition 4.4. *$\leq_c$ is a precongruence, and $\sim_c$ is a congruence.*

Proof. We firstly show that $\leq_c$ is transitive. Let $s \leq_c t, t \leq_c r$, let C_1, C_2 be contexts such that $C_1[s]\downarrow$ and $C_2[s]\Downarrow$. From $s \leq_c t$ then follows $C_1[t]\downarrow$ and $C_2[t]\Downarrow$ and with $t \leq_c r$ we have $C_1[r]\downarrow$ and $C_2[r]\Downarrow$, i.e. $s \leq_c r$

Further let $s \leq_c t$ and let C_1 be a context. To show $C_1[s] \leq_c C_1[t]$, let C_2 be an arbitrary context. If $C_2[C_1[s]]\downarrow$, use the context $C_2[C_1[\cdot]]$, then with $s \leq_c t$ it follows that $C_2[C_1[t]]\downarrow$. If $C_2[C_1[s]]\Downarrow$ we can reason in the same way. □

The following lemma will be used during the proofs of correctness of reductions. By using the contrapositive of the implication in the preorder for may-convergence and Lemma 2.15 the following is true:

Lemma 4.5. *Let s, t be terms, then $s \leq_c^{\downarrow} t$ iff $\forall C \in \mathcal{C} : C[t]\Uparrow \implies C[s]\Uparrow$.*

Let $s \leq_c^{\downarrow} t$, i.e. $\forall C \in \mathcal{C} : C[s]{\downarrow} \implies C[t]{\downarrow}$. The contrapositive is then $\forall C \in \mathcal{C} : \neg(C[t]{\downarrow}) \implies \neg(C[s]{\downarrow})$. With Lemma 2.15 we have $\forall C \in \mathcal{C} : C[t]{\Uparrow} \implies C[s]{\Uparrow}$.

As an important part of this paper we will prove that all deterministic reduction rules are correct program transformations:

Definition 4.6 (Correct Program Transformation). *A binary relation ν on terms is a* correct program transformation *iff $\forall$ terms $s, t : s \; \nu \; t \implies s \sim_c t$.*

In the remaining subsections we develop some tools that will help us to prove correctness of program transformations.

4.2 Context Lemmas

The goal of this section is to prove a "context lemma" which enables to prove contextual equivalence of two terms by observing their termination behaviour only in all reduction contexts instead of all contexts of class $\mathcal{C}$. [Mor98] also provides a context lemma for his call-by-need calculus, but only for may-convergence. We improve upon his work by providing a context lemma for may- as well as must-convergence.

The structure of this section is as follows: We firstly show some properties that will be necessary for proving a context lemma for may-convergence and a context lemma for must-convergence. The section ends with a context lemma for the combined precongruence.

In this section we will use *multicontexts*, i.e. terms with several (or no) holes $\cdot_i$, where 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 placed into the holes $\cdot_i$, then we denote the resulting term as $C[s_1, \ldots, s_n]$.

Lemma 4.7. *Let C be a multicontext with n holes then the following holds:*

If there are terms s_i with $i \in \{1, \ldots, n\}$ such that $C[s_1, \ldots, s_{i-1}, \cdot_i, s_{i+1}, \ldots, s_n]$ is a reduction context, then there exists a hole $\cdot_j$, such that for all terms $t_1, \ldots, t_n$ $C[t_1, \ldots, t_{j-1}, \cdot_j, t_{j+1}, \ldots, t_n]$ is a reduction context.

Proof. We assume there is a multicontext C with n holes and there are terms $s_1, \ldots, s_n$ with $R_i \equiv C[s_1, \ldots, s_{i-1}, \cdot_i, s_{i+1}, \ldots, s_n]$ being a reduction context. Since R_i is a reduction context, there is an execution of the unwinding algorithm $\mathcal{UW}$ starting with $C[s_1, \ldots, s_n]$ which visits s_i (see Lemma 2.7). We fix this execution and apply the same evaluation to $C[\cdot_1, \ldots, \cdot_n]$ and stop when we arrive at a hole. Either the evaluation stops at hole $\cdot_i$ or earlier at some hole $\cdot_j$. Since the unwinding algorithm visits only positions in a reduction context, the claim follows. □

Lemma 4.8. *Let s, t be expressions, σ be a permutation on variables, then*

- $(\forall R \in \mathcal{R} : R[s]{\downarrow} \implies R[t]{\downarrow}) \implies (\forall R \in \mathcal{R} : R[\sigma(s)]{\downarrow} \implies R[\sigma(t)]{\downarrow})$, *and*
- $(\forall R \in \mathcal{R} : R[s]{\uparrow} \implies R[t]{\uparrow}) \implies (\forall R \in \mathcal{R} : R[\sigma(s)]{\uparrow} \implies R[\sigma(t)]{\uparrow})$.

Proof. Let s ,t be terms that the precondition holds, i.e. $\forall R : R[s]{\downarrow} \implies R[t]{\downarrow}$. Let σ be a permutation on variables names. Let R_0 be a reduction context with $R_0[\sigma(s)]{\downarrow}$. We can construct a reduction context R_1 by renaming these bound variables in R_0 which capture a free variable in $\sigma(s)$ or $\sigma(t)$ such that $R_1[s]{\downarrow}$, with the precondition we have $R_1[t]{\downarrow}$. By undoing the renaming of R_1 we can observe that $R_0[\sigma(t)]{\downarrow}$. The proof for the other part is analogous. □

We now prove a lemma using multicontexts which is more general than needed, since the context lemma for may-convergence (Lemma 4.10) is a specialisation of the claim.

Lemma 4.9. *For $i = 1,\dots,k$, $k \geq n$ let s_i, t_i be expressions. Then*

$$\forall i : \forall R \in \mathcal{R} : (R[s_i]{\downarrow} \Rightarrow R[t_i]{\downarrow})$$
$$\Longrightarrow$$
$$\forall \text{ multicontexts } C : C[s_1,\dots,s_n]{\downarrow} \Rightarrow C[t_1,\dots,t_n]{\downarrow}.$$

Proof. Let $RED \in \mathcal{CON}(C[s_1,\dots,s_n])$, we use induction on the following lexicographic ordering of pairs (l, n):

1. $l = \mathtt{rl}(RED)$
2. n=The number of holes in C.

The claim holds for all pairs $(l, 0)$, since if C has no holes there is nothing to show. Now, let $(l, n) > (0, 0)$. For the induction step, we assume the claim holds for all pairs (l', n') that are strictly smaller than (l, n). We assume that there exist s_i, t_i with $i \in \{1,\dots,n\}$ such that the precondition holds, i.e. $\forall i : \forall R \in \mathcal{R} : (R[s_i]{\downarrow} \Rightarrow R[t_i]{\downarrow})$. Let C be a multicontext with n holes and $RED \in \mathcal{CON}(C[s_1,\dots,s_n])$ with $\mathtt{rl}(RED) = l$, we split into two cases:

- At least one hole of C is in a reduction context. We assume hole $\cdot_j$ is in a reduction context. With Lemma 4.7 we have: There is a hole $\cdot_i$ with $R_1 \equiv C[s_1,\dots,s_{i-1},\cdot_i,s_{i+1},\dots,s_n]$ and $R_2 \equiv C[t_1,\dots,t_{i-1},\cdot_i,t_{i+1},\dots,t_n]$ being reduction contexts. Let $C_1 \equiv C[\cdot_1,\dots,\cdot_{i-1},s_i,\cdot_{i+1},\dots,\cdot_n]$. From $C[s_1,\dots,s_n] \equiv C_1[s_1,\dots,s_{i-1},s_{i+1},\dots,s_n]$ we have $RED \in \mathcal{CON}(C_1[s_1,\dots,s_{i-1},s_{i+1},\dots,s_n])$. Since C_1 has $n-1$ holes, we can use the induction hypothesis and derive $C_1[t_1,\dots,t_{i-1},t_{i+1},\dots,t_n]{\downarrow}$, i.e. $C[t_1,\dots,t_{i-1},s_i,t_{i+1},\dots,t_n]{\downarrow}$. From that we have $R_2[s_i]{\downarrow}$. Using the precondition we derive $R_2[t_i]{\downarrow}$, i.e. $C[t_1,\dots,t_n]{\downarrow}$.
- No hole of C is in a reduction context. If $l = 0$, then $C[s_1,\dots,s_n]$ is a WHNF and since no hole is in a reduction context, $C[t_1,\dots,t_n]$ is also a WHNF, hence $C[t_1,\dots,t_n]{\downarrow}$. If $l > 0$ then the first normal order reduction of RED can also be used for $C[t_1,\dots,t_n]$. We now argue that this normal order reduction can modify the context C, the number of occurrences of the terms s_i and the positions of the terms s_i and the elimination, duplication, variable permutation for every s_i is the same for every t_i. More formal, we show: For $i = 1,\dots,m$ let $(s'_i, t'_i) \equiv (s_j, t_j)$ or $(s'_i, t'_i) \equiv (\rho(s_j), \rho(t_j))$ for some j, where

ρ is a permutation on variables. If $C[s_1,\ldots,s_n] \xrightarrow{\mathbf{no},a} C'[s'_1,\ldots,s'_m]$ then $C[t_1,\ldots,t_n] \xrightarrow{\mathbf{no},a} C'[t'_1,\ldots,t'_m]$. We can firstly verify by going through the cases of Definition 2.8 that the modified part of a normal order reduction is also in a reduction context. Now we consider what can happen to the terms s_i and t_i:

- If $\cdot_i$ is in an alternative of `case`, that is discarded by a (case)-reduction, or $\cdot_i$ is in the argument of a `seq`- or `amb`-expression that is discarded by a (seq)- or (amb)-reduction, then s_i and t_i are both eliminated.
- If the normal order reduction is not a (cp)-reduction that copies a superterm of s_i or t_i, and s_i and t_i are not eliminated as mentioned in the previous bullet, then s_i and t_i can only move their position.
- If the normal order reduction is a (cp)-reduction that copies a superterm of s_i or t_i, then renamed copies $\rho_{s,i}(s_i)$ and $\rho_{t,i}(t_i)$ of s_i and t_i will occur, where $\rho_{s,i}, \rho_{t,i}$ are permutations on variables. W.l.o.g. for all i: $\rho_{s,i} = \rho_{t,i}$. Free variables of s_i or t_i can also be renamed in $\rho_{s,i}(s_i)$ and $\rho_{t,i}(t_i)$ if they are bound in the copied superterm. But with Lemma 4.8 we have: The precondition also holds for $\rho_{s,i}(s_i)$ and $\rho_{t,i}(t_i)$, i.e. $\forall R \in \mathcal{R}$: $R[\rho_{s,i}(s_i)]\downarrow \implies R[\rho_{t,i}(t_i)]\downarrow$.

Now we can use the induction hypothesis: Since $C'[s'_1,\ldots,s'_m]$ has a terminating sequence of normal order reductions of length $l-1$ we also have $C'[t'_1,\ldots,t'_m]\downarrow$. With $C[t_1,\ldots,t_n] \xrightarrow{\mathbf{no},a} C'[t'_1,\ldots,t'_m]$ we have $C[t_1,\ldots,t_n]\downarrow$. □

Lemma 4.10 (Context Lemma for May Convergence). *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. $s \le_c^{\downarrow} t$.*

Proof. The Lemma is a specialisation of Lemma 4.9. □

Lemma 4.11 (Context Lemma for Must-Convergence). *Let s, t be terms, then*

$$\Big(\big(\forall R \in \mathcal{R} : R[s]\Downarrow \implies R[t]\Downarrow\big) \wedge \big(\forall R \in \mathcal{R} : R[s]\downarrow \implies R[t]\downarrow\big)\Big)$$
$$\implies$$
$$(\forall C : C[s]\Downarrow \implies C[t]\Downarrow)$$

Proof. We prove a more general claim using multicontexts and the contrapositive of the first of the inner implications: Let s_i, t_i be terms, then

$$\Big(\big(\forall R \in \mathcal{R} : R[t_i]\uparrow \implies R[s_i]\uparrow\big) \wedge \big(\forall R \in \mathcal{R} : R[s_i]\downarrow \implies R[t_i]\downarrow\big)\Big)$$
$$\implies$$
$$\forall \text{ multicontexts } C : C[t_1,\ldots,t_n]\uparrow \implies C[s_1,\ldots,s_n]\uparrow$$

We use induction on the lexicographical ordering of tuples, with the components:

- $\mathtt{rl}(RED)$ where $RED \in \mathcal{DIV}(C[t_1,\ldots,t_n])$.
- The number of holes in C.

The base case holds, since if C has no holes, there is nothing to show. For the induction step we assume that claim holds for all pairs strictly smaller than (l, m).

Let the precondition hold, i.e. for all $R \in \mathcal{R} : R[t_i]{\uparrow} \implies R[s_i]{\uparrow}$ as well as for all $R \in \mathcal{R} : R[s_i]{\downarrow} \implies R[t_i]{\downarrow}$. Let $C[t_1, \ldots, t_m] \xrightarrow{\mathbf{no},l} t'$ with $t'{\Uparrow}$. We split into two cases:

- At least one hole in C is in a reduction context. Then there is a hole $\cdot_j$ with $R_1 \equiv C[t_1, \ldots, t_{j-1}, \cdot, t_{j+1}, \ldots, t_m]$ and $R_2 \equiv C[s_1, \ldots, s_{j-1}, \cdot, s_{j+1}, \ldots, s_m]$ being reduction contexts. Let $C' = C[\cdot_1, \ldots, \cdot_{j-1}, t_j, \cdot_{j+1}, \ldots, \cdot_m]$. Since $C'[t_1, \ldots, t_{j-1}, t_{j+1}, \ldots, t_m] \xrightarrow{\mathbf{no},l} t'$ and C' has $m - 1$ holes, we can use the induction hypothesis and have $C'[s_1, \ldots, s_{j-1}, s_{j+1}, \ldots, s_m]{\uparrow}$ and hence $R_2[t_i]{\uparrow}$. Using the precondition we have $R_2[s_i]{\uparrow}$, thus $C[s_1, \ldots, s_m]{\uparrow}$.
- No hole of C is in a reduction context. Then we split into two subcases:
  - $l > 0$: Then we can perform the same first reduction for $C[t_1, \ldots, t_m]$ also for $C[s_1, \ldots, s_m]$. With the same reasoning as in the proof of Lemma 4.10, we have that the results of the reduction are $C'[t'_1, \ldots, t'_{m'}]$ and $C'[s'_1, \ldots s'_{m'}]$, where $(t'_i, s'_i) = (\rho(t'_j), \rho(s'_j))$ for a variable permutation ρ. With Lemma 4.8 we have that the precondition also holds for s'_i, t'_i. Since $C'[t'_1, \ldots, t'_{m'}] \xrightarrow{\mathbf{no},l-1} t'$ we can use the induction hypothesis and have $C[s'_1, \ldots, s'_m]{\uparrow}$. Since $C[s_1, \ldots, s_m] \xrightarrow{\mathbf{no}} C[s'_1, \ldots, s'_m]$, we have $C[s_1, \ldots, s_m]{\uparrow}$.
  - $l = 0$: Then $C[t_1, \ldots, t_m]{\Uparrow}$. We assume that $\neg(C[s_1, \ldots, s_m]{\uparrow})$, i.e. $C[s_1, \ldots, s_m]{\Downarrow}$. Then we also have $C[s_1, \ldots, s_m]{\downarrow}$. Using the precondition and Lemma 4.9, we have $C[t_1, \ldots, t_m]{\downarrow}$ which is a contradiction. □

Corollary 4.12. *If* $s \leq_c^{\downarrow} t$ *and for all* $R \in \mathcal{R} : R[t]{\uparrow} \implies R[s]{\uparrow}$ *then* $s \leq_c^{\Downarrow} t$

Proof. Let s, t be terms such that the precondition holds. From $s \leq_c^{\downarrow} t$ we have $\forall R \in \mathcal{R} : R[s]{\downarrow} \implies R[t]{\downarrow}$. The second part of the precondition is equivalent to $\forall R \in \mathcal{R} : R[s]{\Downarrow} \implies R[t]{\Downarrow}$. Using Lemma 4.11 we have $s \leq_c^{\Downarrow} t$. □

By combining the context lemma for may-convergence and the context lemma for must-convergence we derive the following lemma.

Lemma 4.13 (Context Lemma). *Let s, t be terms, then*

$$\Big(\big(\forall R \in \mathcal{R} : R[s]{\Downarrow} \implies R[t]{\Downarrow}\big) \wedge \big(\forall R \in \mathcal{R} : R[s]{\downarrow} \implies R[t]{\downarrow}\big) \Big) \implies s \leq_c t$$

Proof. Follows from Lemma 4.10 and Lemma 4.11. □

4.3 Properties of the (lll)-Reduction

The following lemma shows that `letrec`s in reduction contexts can be moved to the top level environment by a sequence of normal order reductions.

Lemma 4.14. *The following holds:*

1. *For all terms of the form* (letrec Env_1 in $R_1^-[(\texttt{letrec } Env_2 \texttt{ in } t)]$) *where* R_1^- *is a weak reduction context, there exists a sequence of normal order* (lll)-*reductions with*

$$(\texttt{letrec } Env_1 \texttt{ in } R_1^-[(\texttt{letrec } Env_2 \texttt{ in } t)]) \xrightarrow{\textbf{no},\mathsf{lll},+} (\texttt{letrec } Env_1, Env_2 \texttt{ in } R_1^-[t]).$$

2. *For all terms of the form*

$$\texttt{letrec } Env_1, x_1 = R_1^-[(\texttt{letrec } Env_2 \texttt{ in } t)], \{x_i = R_i^-[x_{i-1}]\}_{i=2}^m \texttt{ in } R_{m+1}^-[x_m]$$

where R_j^-, $j = 1, \ldots, m+1$, *are weak reduction contexts there exists a sequence of normal order* (lll)-*reductions with*

$$\texttt{letrec } Env_1, x_1 = R_1^-[(\texttt{letrec } Env_2 \texttt{ in } t)], \{x_i = R_i^-[x_{i-1}]\}_{i=2}^m \texttt{ in } R_{m+1}^-[x_m]$$
$$\xrightarrow{\textbf{no},\mathsf{lll},+} (\texttt{letrec } Env_1, Env_2, x_1 = R_1^-[t], \{x_i = R_i^-[x_{i-1}]\}_{i=2}^m \texttt{ in } R_{m+1}^-[x_m])$$

3. *For all terms of the form* $R_1^-[(\texttt{letrec } Env \texttt{ in } t)]$ *where* R_1^- *is a weak reduction context, there exists a sequence of normal order* (lll)-*reductions with*

$$R_1^-[(\texttt{letrec } Env \texttt{ in } t)] \xrightarrow{\textbf{no},\mathsf{lll},*} (\texttt{letrec } Env \texttt{ in } R_1^-[t])$$

Proof. This follows by induction on the main depth of the context R_1^-. □

Another property of the (lll)-reduction is that every reduction sequence consisting only of (lll)-reductions must be finite.

Definition 4.15. *For a given term t, the measure* $\mu_{lll}(t)$ *is a pair* $(\mu_1(t), \mu_2(t))$, *ordered lexicographically. The measure* $\mu_1(t)$ *is the number of* letrec-*subexpressions in t, and* $\mu_2(t)$ *is the sum of* lrdepth(s,t) *of all* letrec-*subexpressions s of t, where* lrdepth *is defined as follows:*

$$\texttt{lrdepth}(s,s) = 0$$
$$\texttt{lrdepth}(s, C_1[C_2[s]]) = \begin{cases} 1 + \texttt{lrdepth}(s, C_2[s]) & \text{if } C_1 \text{ is a context of main depth 1, and not a } \texttt{letrec} \\ \texttt{lrdepth}(s, C_2[s]) & \text{if } C_1 \text{ is a context of main depth 1, and it is a } \texttt{letrec} \end{cases}$$

Example 4.16. Let $s \equiv (\texttt{letrec } x = ((\lambda y.y)\ (\texttt{letrec } z = \texttt{True in } z)) \texttt{ in } x)$ then $\mu_{lll}(s) = (2,1)$ where

$$\texttt{lrdepth}(s, (\texttt{letrec } z = \texttt{True in } z))$$
$$= \texttt{lrdepth}(((\lambda y.y)\ (\texttt{letrec } z = \texttt{True in } z)), (\texttt{letrec } z = \texttt{True in } z))$$
$$= 1 + \texttt{lrdepth}((\texttt{letrec } z = \texttt{True in } z), (\texttt{letrec } z = \texttt{True in } z)) = 1 + 0 = 1$$

Proposition 4.17. *The reduction* (lll) *is terminating, I.e. there are no infinite reductions sequences consisting only of* (C, lll)*-reductions.*

Proof. The proposition holds, since $s \xrightarrow{C,\mathsf{lll}} t$ implies $\mu_{lll}(s) > \mu_{lll}(t)$, and $\forall\ t$: $\mu_{lll}(t) \geq (0, 0)$. □

4.4 Complete Sets of Commuting and Forking Diagrams

For proving correctness of the reduction rules and of further program transformations we introduce complete sets of commuting diagrams and complete sets of forking diagrams. They have already been successfully used in [Kut00,SS03,Sab03,SSSS04,SSSS05,Man05] as a proof tool for proving contextual equivalence of program transformations.

We start with defining so-called *internal* reductions:

Definition 4.18. *Let* s*,* t *be terms,* $\mathcal{X}$ *be a context class. A reduction* $s \xrightarrow{X,red} t$ *is called* $\mathcal{X}$-internal*, if it is not a normal order reduction for* s*, and* $X \in \mathcal{X}$*. We denote* $\mathcal{X}$*-internal reductions with* $s \xrightarrow{i\mathcal{X},red} t$*.*

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 or some other program transformation. In the following we introduce transformations on reduction sequences, by using the notation

$$\xrightarrow{X,red} . \xrightarrow{\mathbf{no},a_1} \ldots \xrightarrow{\mathbf{no},a_k} \quad \rightsquigarrow \quad \xrightarrow{\mathbf{no},b_1} \ldots \xrightarrow{\mathbf{no},b_m} . \xrightarrow{X,red_1} \ldots . \xrightarrow{X,red_h},$$

where $\xrightarrow{X,red}$ is a reduction inside a context of a specific class like $\mathcal{C}$ or an internal reduction inside such a context class (e.g. $i\mathcal{C}$).

Such a transformation rule *matches* the prefix of the reduction sequence RED, if the prefix is: $s \xrightarrow{X,red} t_1 \xrightarrow{\mathbf{no},a_1} \ldots t_k \xrightarrow{\mathbf{no},a_k} t$. The transformation rule is *applicable* to the prefix of a reduction sequence RED with prefix $s \xrightarrow{X,red} x_1 \xrightarrow{\mathbf{no},a_1} \ldots x_k \xrightarrow{\mathbf{no},a_k} t$, iff the following holds:

$$\exists r_1, \ldots, r_{m+h-1} : s \xrightarrow{\mathbf{no},b_1} r_1 \ldots \xrightarrow{\mathbf{no},b_m} r_m \xrightarrow{X,red_1} r_{m+1} \ldots r_{m+h-1} \xrightarrow{X,red_h} t.$$

The transformation consists in replacing the prefix of RED with the result, i.e. leading to $s \xrightarrow{\mathbf{no},b_1} r_1 \ldots \xrightarrow{\mathbf{no},b_m} r_m \xrightarrow{X,red_1} r_{m+1} \ldots r_{m+h-1} \xrightarrow{X,red_h} t$.

Since we will use sets of transformation rules, it may be the case that there is a transformation rule in the set, that matches a prefix of a reduction sequence, but it is not applicable, since the right hand side cannot be constructed. But in a complete set there is always at least one diagram that is applicable.

Definition 4.19 (Complete Sets of Commuting / Forking Diagrams). *A* complete set of commuting diagrams for the reduction $\xrightarrow{X,red}$ *is a set of transformation rules on reduction sequences of the form*

$$\xrightarrow{X,red} . \xrightarrow{\mathbf{no},a_1} \ldots \xrightarrow{\mathbf{no},a_k} \quad \rightsquigarrow \quad \xrightarrow{\mathbf{no},b_1} \ldots \xrightarrow{\mathbf{no},b_m} . \xrightarrow{X,red_1} \ldots . \xrightarrow{X,red_{k'}},$$

depicted with a diagram of the form shown in Fig. 4 (a), where $k, k' \geq 0, m \geq 1$, *such that in every reduction sequence* $t_0 \xrightarrow{X,red} t_1 \xrightarrow{\mathbf{no}} \ldots \xrightarrow{\mathbf{no}} t_h$, *where* t_0 *is not a WHNF, at least one of the transformation rules is applicable to a prefix of the sequence.*

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

$$\xleftarrow{\mathbf{no},a_1} \ldots \xleftarrow{\mathbf{no},a_k} . \xrightarrow{X,red} \quad \rightsquigarrow \quad \xrightarrow{X,red_1} . \ldots . \xrightarrow{X,red_{k'}} . \xleftarrow{\mathbf{no},b_1} \ldots \xleftarrow{\mathbf{no},b_m},$$

depicted by a diagram of the form shown in Fig. 4 (b), where $k, k' \geq 0, m \geq 1$, *such that for every reduction sequence* $t_h \xleftarrow{\mathbf{no}} \ldots t_2 \xleftarrow{\mathbf{no}} t_1 \xrightarrow{X,red} t_0$, *where* t_1 *is not a WHNF and* $t_1 \xrightarrow{X,red} t_0$ *is not the same reduction with the same (inner) redex as* $t_0 \xrightarrow{\mathbf{no}} t_1$, *at least one of the transformation rules from the set is applicable to a suffix of the sequence.*

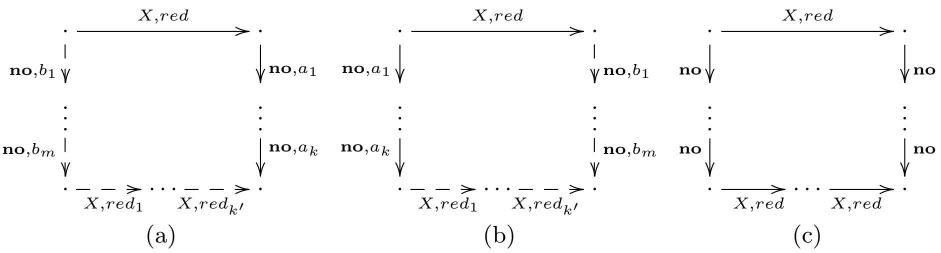


Fig. 4. Commuting and Forking Diagrams and their common representation

The two different kinds of diagrams are required for two different parts of the proof for the contextual equivalence of two terms. Commuting and forking diagrams often have a common representation (see Fig. 4 (c)). We will give the diagrams only in the common representation if the corresponding commuting and forking diagrams can be read off obviously.

We abbreviate k reductions of type a with $\xrightarrow{a,k}$. As another notation, we use the $*$- and $+$-notation of regular expressions for the diagrams. The interpretation is an infinite set of diagrams constructed as follows: Repeat the following step as long as diagrams with reductions labeled with $*$ or $+$ exist.

For a reduction $\xrightarrow{a,*}$ ($\xrightarrow{a,+}$, respectively) of a diagram insert diagrams for all $i \in \mathbb{N}_0$ ($i \in \mathbb{N}$) with $\xrightarrow{a,*}$ ($\xrightarrow{a,+}$) replaced by $\xrightarrow{a,i}$ reductions into the set.

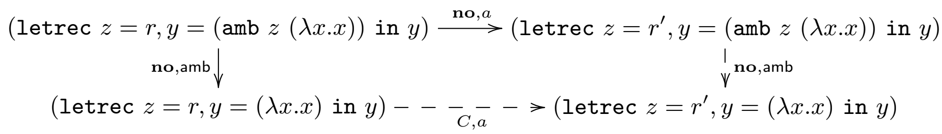
5 Correctness of (lbeta), (case-c), (seq-c)

In this section we use the context lemmas together with complete sets of commuting and forking diagrams to prove that (lbeta), (case-c) and (seq-c) are correct program transformations.

Lemma 5.1. *Let red* $\in$ {lbeta, case-c, seq-c, amb-c, lapp, lcase, lseq, lamb}. *If* $s \xrightarrow{red} t$ *then* $t \leq_c^{\downarrow} s$.

Proof. Let $red \in$ {lbeta, case-c, seq-c, amb-c, lapp, lcase, lseq, lamb}, $s \xrightarrow{red} t$ and $R[t]\!\downarrow$. Then there exists $RED \in \mathcal{CON}(R[t])$. Since every reduction red of the kind mentioned in the lemma inside a reduction context is a normal order reduction, we have $R[s] \xrightarrow{\mathbf{no},a} R[t]$. By appending RED to $R[s] \xrightarrow{R,red} R[t]$ we have $R[s]\!\downarrow$. Hence, $\forall R \in \mathcal{R} : R[t]\!\downarrow \implies R[s]\!\downarrow$. Now, the context lemma for may-convergence shows the claim. ☐

Since the defined normal order reduction may reduce inside the arguments of amb-expressions, the normal order reduction is not unique. I.e., normal order reductions can overlap with other normal order reductions. To treat this situations it is not sufficient to use diagrams for $(\mathbf{no}, a)$, where a is a deterministic reduction with all other normal order reductions. The diagrams cannot be closed in general, this also holds if we regard all reduction contexts, e.g. if an $(\mathbf{no}, a)$-reduction overlaps with an $(\mathbf{no}, \mathsf{amb})$-reduction, we require a non-normal order a-reduction inside a context, that is not a reduction context:



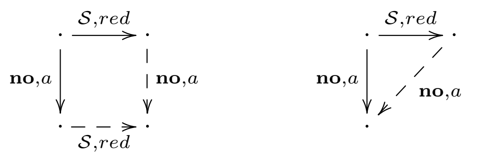
The context C is not a reduction context, but a *surface context*, which are contexts where the hole is not in the body of an abstraction.

Definition 5.2 (Surface Context). *The class* $\mathcal{S}$ *of* surface contexts *is defined as follows:*

$$\begin{aligned}\mathcal{S} ::= {} & [\cdot] \mid (\mathcal{S}\ E) \mid (E\ \mathcal{S}) \mid (\texttt{seq}\ E\ \mathcal{S}) \mid (\texttt{seq}\ \mathcal{S}\ E) \\ & \mid (c_{T,i}\ E_1 \ldots E_{i-1}\ \mathcal{S}\ E_{i+1} \ldots E_{\mathrm{ar}(c_{T,i})}) \mid (\texttt{case}_T\ \mathcal{S}\ alts) \\ & \mid (\texttt{case}_T\ E\ \ldots (Pat\ \to\ \mathcal{S}) \ldots) \mid (\texttt{amb}\ \mathcal{S}\ E) \mid (\texttt{amb}\ E\ \mathcal{S}) \\ & \mid (\texttt{letrec}\ Env\ \texttt{in}\ \mathcal{S}) \mid (\texttt{letrec}\ \ldots, x_i = \mathcal{S}, Env\ \texttt{in}\ E)\end{aligned}$$

Note that every reduction context is also a surface context, i.e. $\mathcal{R} \subset \mathcal{S}$.

Lemma 5.3. *A complete set of forking diagrams for* $\xrightarrow{\mathcal{S},red}$ *with red* $\in$ {lbeta, case-c, seq-c} *is:*



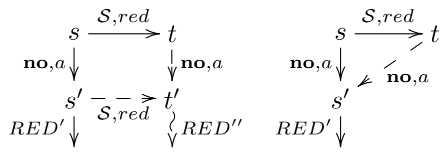
a arbitrary $\qquad$ $a \in$ {case, seq, amb-l, amb-r}

Proof. This follows by inspecting all cases where an $(\mathbf{no}, a)$-reduction overlaps with an (S, red)-reduction with $red \in \{\mathsf{lbeta}, \mathsf{case\text{-}c}, \mathsf{seq\text{-}c}\}$. The first diagram is applicable, if both reductions are performed independently and hence can be commuted. The second diagram is applicable if the redex of *red* is inside an unused alternative of a `case`-expression, inside the first argument of a `seq`-expression or inside an argument of an `amb`-expression, that is discarded by an $(\mathbf{no}, \mathsf{case})$-, $(\mathbf{no}, \mathsf{seq\text{-}c})$- or $(\mathbf{no}, \mathsf{amb})$-reduction, respectively. □

Lemma 5.4. *Let* $red \in \{\mathsf{lbeta}, \mathsf{case\text{-}c}, \mathsf{seq\text{-}c}\}$ *and* $s \xrightarrow{i\mathcal{S},red} t$ *then* s *is a WHNF iff* t *is a WHNF.*

Lemma 5.5. *Let* $red \in \{\mathsf{lbeta}, \mathsf{case\text{-}c}, \mathsf{seq\text{-}c}\}$ *and* $s \xrightarrow{red} t$ *then* $s \leq_c^{\downarrow} t$

Proof. By using the context lemma for may-convergence, we need to show, that if $s_0 \xrightarrow{red} t_0$, then for all reduction contexts $R : R[s_0]\downarrow \implies R[t_0]\downarrow$. We will show the same statement for all surface contexts. This is sufficient, since every reduction context is also a surface context. Let $s \equiv S[s_0]$, $t \equiv S[t_0]$ with $s \xrightarrow{S,red} t$. Further let $RED \in \mathcal{CON}(s)$. By induction on the length of RED we show that $t\downarrow$. The base case is covered by Lemma 5.4. Let RED be of length $l > 0$. If the first reduction of RED is the same reduction as the (S, red)-reduction then there is nothing to show. In all other cases we can apply a diagram from the complete set of forking diagrams of Lemma 5.3 to a suffix of $\xleftarrow{RED} s \xrightarrow{S,red} t$. Let RED' be the suffix of RED of length $l - 1$, then we have the following two possibilities:



(1) (2)

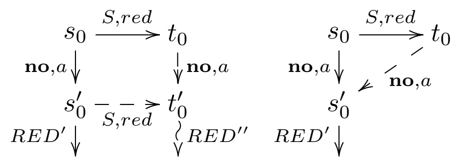
(1) We use the induction hypothesis for RED', thus $t'\downarrow$. With $t \xrightarrow{\mathbf{no},a} t'$ we have $t\downarrow$.
(2) If the second diagram is applicable, we can append RED' to $t \xrightarrow{\mathbf{no},a} s'$, i.e. $t\downarrow$. □

Lemma 5.6. *If* $s \xrightarrow{red} t$ *with* $red \in \{\mathsf{lbeta}, \mathsf{case\text{-}c}, \mathsf{seq\text{-}c}\}$ *then* $s \leq_c^{\Downarrow} t$.

Proof. We use Corollary 4.12. We have $s \leq_c^{\downarrow} t$ from Lemma 5.5. It remains to show $\forall R \in \mathcal{R} : R[t]\uparrow \implies R[s]\uparrow$. Let R be an reduction context, $R[s] \xrightarrow{R,red} R[t]$ and $R[t]\uparrow$. Since every reduction $red \in \{\mathsf{lbeta}, \mathsf{case\text{-}c}, \mathsf{seq\text{-}c}\}$ in a reduction context is also a normal order reduction, we have $R[s]\uparrow$. □

Lemma 5.7. *If* $s \xrightarrow{red} t$ *with* $red \in \{\mathsf{lbeta}, \mathsf{case\text{-}c}, \mathsf{seq\text{-}c}\}$ *then* $t \leq_c^{\Downarrow} s$.

Proof. We use Corollary 4.12. From Lemma 5.1 we have $t \leq_c^{\downarrow} s$. It remains to show $\forall R \in \mathcal{R} : R[s]\uparrow \implies R[t]\uparrow$. We will show the statement for all surface contexts. Let $s_0 \equiv S[s]$, $t_0 = S[t]$, $s \xrightarrow{red} t$ and $s_0\uparrow$. We use induction on the length k of a sequence $RED \in \mathcal{DIV}(s_0)$. If $k = 0$, i.e. $s_0\Uparrow$, then the claim follows from Lemma 5.1 using Lemma 4.5. Now, let $k > 0$. Since s_0 cannot be a WHNF, we can apply a forking diagram to a suffix of $\xleftarrow{RED} s_0 \xrightarrow{S,red} t_0$. Let RED' be the suffix of RED of length $k-1$. Then we have two cases:



(1) (2)

(1) We can apply the induction hypothesis to $\xleftarrow{RED'} s_0' \xrightarrow{S,red} t_0'$ and hence have $t_0'\uparrow$. With $t_0 \xrightarrow{\mathbf{no},a} t_0'$, we have $t_0\uparrow$.
(2) For this case we have $t_0\uparrow$, since $t_0 \xrightarrow{\mathbf{no},a} s_0'$. □

Proposition 5.8. *The reductions* (lbeta)*,* (case-c) *and* (seq-c) *keep contextual equivalence, i.e. if* $s \xrightarrow{red} t$ *with* $red \in \{\text{lbeta}, \text{case-c}, \text{seq-c}\}$ *then* $s \sim_c t$*.*

Proof. This follows from Lemma 5.1, 5.5, 5.7 and 5.6. □

6 Additional Correct Program Transformations

We now define some additional program transformations, which will be necessary during the proofs of the correctness of the remaining reduction rules of $\Lambda^{\texttt{let}}_{\texttt{amb}}$ and are also useful compile- or run-time optimisations.

Definition 6.1. *In Fig. 5 some additional transformation rules are defined. We define the following unions:*

$$\begin{array}{ll}
(\mathsf{gc}) & := (\mathsf{gc1}) \cup (\mathsf{gc2}) \\
(\mathsf{cpx}) & := (\mathsf{cpx\text{-}in}) \cup (\mathsf{cpx\text{-}e}) \\
(\mathsf{cpcx}) & := (\mathsf{cpcx\text{-}in}) \cup (\mathsf{cpcx\text{-}e}) \\
(\mathsf{opt}) & := (\mathsf{gc}) \cup (\mathsf{cpx}) \cup (\mathsf{cpcx}) \cup (\mathsf{abs}) \cup (\mathsf{xch})
\end{array}$$

The transformation (gc) performs garbage collection by removing unnecessary bindings, (cpx) copies variables, (cpcx) abstract a constructor application and then copy it, the rule (abs) abstracts a constructor application by sharing the arguments through new `letrec`-bindings and the rule (xch) restructures two bindings in an `letrec`-environment by reversing an indirection and the corresponding binding.

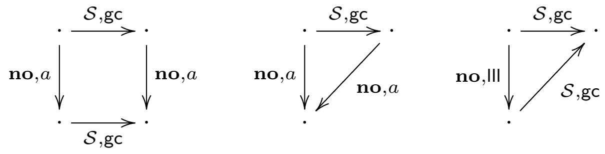
We will now develop complete sets of commuting and complete sets of forking diagrams for all additional transformations. After this we will combine the sets to derive complete sets for (opt) and then prove correctness of (opt).

(gc1)	$(\texttt{letrec}\ x_1 = s_1, \ldots, x_n = s_n\ \texttt{in}\ t) \to t$ if $x_i, i = 1, \ldots, n$ does not occur free in t
(gc2)	$(\texttt{letrec}\ x_1 = s_1, \ldots, x_n = s_n, y_1 = t_1, \ldots, y_m = t_m\ \texttt{in}\ t)$ $\to (\texttt{letrec}\ y_1 = t_1, \ldots, y_m = t_m\ \texttt{in}\ t)$ if $x_i, i = 1, \ldots, n$ does not occur free in t nor in any t_j, for $j = 1, \ldots, m$
(cpx-in)	$(\texttt{letrec}\ x = y, Env\ \texttt{in}\ C[x]) \to (\texttt{letrec}\ x = y, Env\ \texttt{in}\ C[y])$ if $x \not\equiv y$ and y is a variable
(cpx-e)	$(\texttt{letrec}\ x = y, z = C[x], Env\ \texttt{in}\ s) \to (\texttt{letrec}\ x = y, z = C[y], Env\ \texttt{in}\ s)$ if $x \not\equiv y$ and y is a variable
(cpcx-in)	$(\texttt{letrec}\ x = (c\ \overrightarrow{s_i}), Env\ \texttt{in}\ C[x])$ $\to (\texttt{letrec}\ x = (c\ \overrightarrow{y_i}), \{y_i = s_i\}_{i=1}^{\mathrm{ar}(c)}, Env\ \texttt{in}\ C[(c\ \overrightarrow{y_i})])$ where y_i are fresh variables
(cpcx-e)	$(\texttt{letrec}\ x = (c\ \overrightarrow{s_i}), z = C[x], Env\ \texttt{in}\ r)$ $\to (\texttt{letrec}\ x = (c\ \overrightarrow{y_i}), \{y_i = s_i\}_{i=1}^{\mathrm{ar}(c)}, Env, z = C[(c\ \overrightarrow{y_i})]\ \texttt{in}\ r)$ where y_i are fresh variables
(abs)	$(\texttt{letrec}\ x = (c\ \overrightarrow{s_i}), Env\ \texttt{in}\ r)$ $\to (\texttt{letrec}\ x = (c\ \overrightarrow{y_i}), \{y_i = s_i\}_{i=1}^{\mathrm{ar}(c)}, Env\ \texttt{in}\ r)$ where y_i are fresh variables
(xch)	$(\texttt{letrec}\ x = s, y = x, Env\ \texttt{in}\ r) \to (\texttt{letrec}\ x = y, y = s, Env\ \texttt{in}\ r)$

Fig. 5. Additional Transformation Rules

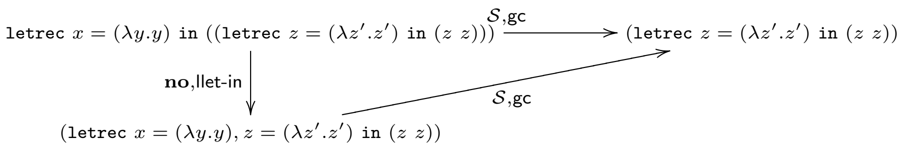
6.1 Diagrams for (gc)

Lemma 6.2. *A complete set of forking diagrams and a complete set of commuting diagrams for* (S, gc) *can be read off the following diagrams:*



a arbitrary $a \in \{\mathsf{case}, \mathsf{seq}, \mathsf{amb}\}$ *a arbitrary*

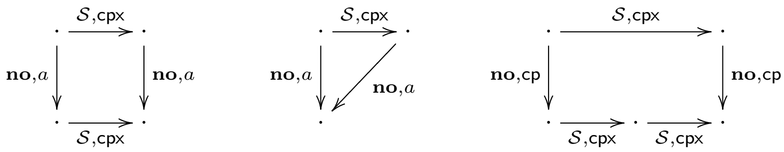
Proof. This follows by inspecting all cases where an (S, gc)-transformation overlaps with a normal order reduction or is followed by a normal order reduction. The first diagram covers the cases where both rules are performed independently and hence can be commuted. If the redex of the (S, gc)-transformation is discarded by an $(\mathbf{no}, \mathsf{case})$-, $(\mathbf{no}, \mathsf{seq})$- or $(\mathbf{no}, \mathsf{amb})$-reduction then the second diagram is applicable. The last diagram covers the cases where an $(\mathbf{no}, \mathsf{lll})$ redex is eliminated by an (S, gc)-transformation, e.g.



□

6.2 Diagrams for (cpx)

Lemma 6.3. *A complete set of forking diagrams and a complete set of commuting diagrams for* (S, cpx) *can be read off the following diagrams:*



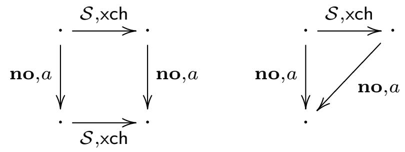
a arbitrary $a \in \{\mathsf{case}, \mathsf{seq}, \mathsf{amb}, \mathsf{cp}\}$

Proof. By case analysis we have the following possibilities for the overlappings of (S, cpx) and a normal order reduction:

- The first diagram describes the cases, where the reductions can be commuted.
- The second diagram is applicable, if the redex or inner redex of the (S, cpx)-transformation is discarded by an $(\mathbf{no}, \mathsf{case})$-, $(\mathbf{no}, \mathsf{seq})$- or $(\mathbf{no}, \mathsf{amb})$-reduction or if the target variable of (S, cpx) and an $(\mathbf{no}, \mathsf{cp})$-reduction are identical.
- The last diagram covers the cases where the target of the (S, cpx)-transformation is inside the body of an abstraction that is copied by an $(\mathbf{no}, \mathsf{cp})$-reduction. □

6.3 Diagrams for (xch)

Lemma 6.4. *A complete set of forking diagrams and a complete set of commuting diagrams for* (S, xch) *can be read off the following diagrams:*

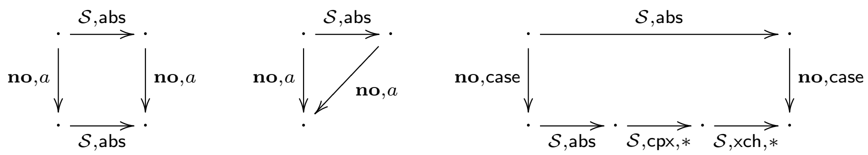


a *arbitrary* $\quad a \in \{\mathsf{case}, \mathsf{seq}, \mathsf{amb}\}$

Proof. Either an (S, xch)-transformation and a normal order reduction commute, or the redex of (S, xch) is discarded by an $(\mathbf{no}, \mathsf{case})$-, $(\mathbf{no}, \mathsf{seq})$- or an $(\mathbf{no}, \mathsf{amb})$-reduction. □

6.4 Diagrams for (abs)

Lemma 6.5. *A complete set of forking and a complete set of commuting diagrams for* (S, abs) *can be read off the following diagrams:*

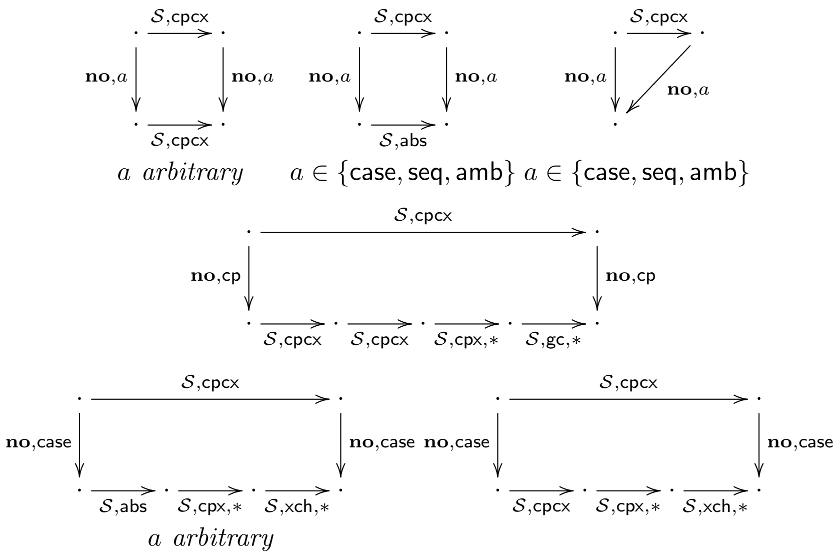


a *arbitrary* $\quad a \in \{\mathsf{case}, \mathsf{seq}, \mathsf{amb}\}$

Proof. By inspecting the overlappings between an $(\mathbf{no}, a)$-reduction and an (S, abs)-transformation we derive the three kinds of diagrams. If the rules are performed independently, then the first diagram is applicable. The second diagram describes the cases where the redex of (S, abs) is discarded by a normal order reduction. The last diagram covers the cases where a normal order (case)-reduction uses the same constructor application that is abstracted by the (S, abs)-transformation. □

6.5 Diagrams for (cpcx)

Lemma 6.6. *A complete set of forking diagrams and a complete set of commuting diagrams for* (S, cpcx) *can be read off the following diagrams:*



Proof. The sets follow by case analysis of the overlappings between an $(\mathbf{no}, a)$-reduction and an (S, cpcx)-transformation. The first diagram describes the cases where the rules commute. The second diagram is applicable if the target variable of the (S, cpcx)-transformation is discarded by an $(\mathbf{no}, \mathsf{case})$-, $(\mathbf{no}, \mathsf{seq})$- or $(\mathbf{no}, \mathsf{amb})$-reduction, but the binding for the constructor application is not discarded.

The third diagram covers the cases where the redex of the (S, cpcx)-transformation is discarded by an $(\mathbf{no}, a)$-reduction or where the (S, cpcx)-transformation copies a constant into the first argument of a `case`-expression.

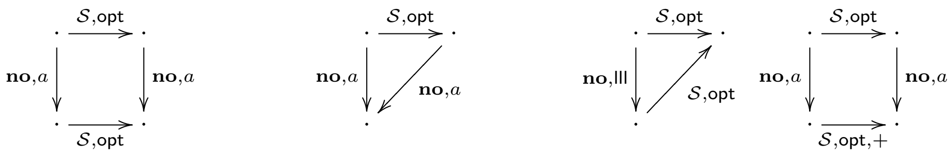
The 4^{th} diagram is applicable if the target of the (S, cpcx)-transformation is inside the body of an abstraction which is copied by an $(\mathbf{no}, \mathsf{cp})$-reduction.

The last two diagrams cover the cases where the same constructor application is used by an $(\mathbf{no}, \mathsf{case})$-reduction and the (S, cpcx)-transformation: the 5^{th} diagram is applicable if the target is the to-be-cased variable; the 6^{th} diagram covers the cases where the target of the (cpcx)-transformation is inside an unused alternative of the `case`-expression. ∎

6.6 Correctness of (opt)

We now combine the diagrams for all transformations of (opt).

Lemma 6.7. *A complete set of forking diagrams and a complete set of commuting diagrams for* (S, opt) *can be read off the following diagrams:*



a arbitrary $a \in \{\mathsf{case}, \mathsf{seq}, \mathsf{amb\text{-}l}, \mathsf{amb\text{-}r}, \mathsf{cp}\}$ $a \in \{\mathsf{cp}, \mathsf{case}\}$

Proof. Follows from Lemma 6.2, 6.3, 6.4, 6.5 and 6.6. □

Lemma 6.8. *Let* $s \xrightarrow{S,\mathsf{opt}} t$ *then the following hold:*

- *If* s *is a WHNF then* t *is a WHNF.*
- *If* t *is a WHNF then either* s *is a WHNF or also in case of an* (S, gc)*-transformation* $s \xrightarrow{\mathbf{no},\mathsf{llet}} s'$ *where* s' *is a WHNF.*

Proof. This follows from the Definition 2.11. For (S, gc) there are three special cases:

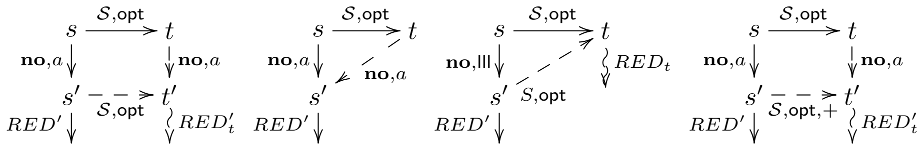
- $s \equiv (\texttt{letrec}\ Env\ \texttt{in}\ s')$ where s' is a WHNF.
- $s \equiv (\texttt{letrec}\ Env_2\ \texttt{in}\ (\texttt{letrec}\ Env\ \texttt{in}\ s'))$ where $(\texttt{letrec}\ Env_2\ \texttt{in}\ s')$ is a WHNF.
- $s \equiv (\texttt{letrec}\ Env_2, x = (\texttt{letrec}\ Env\ \texttt{in}\ r)\ \texttt{in}\ s')$, where $(\texttt{letrec}\ Env_2, x = r\ \texttt{in}\ s')$ is a WHNF.

In all cases an $(\mathbf{no}, \mathsf{llet})$-reduction for s leads to a WHNF. □

The claims that an application of (opt) inside surface contexts keeps may- and must-convergence cannot be proved directly, since the inductions used in the proofs would not work. Hence, we will prove stronger statements by using different lengths of sequences of normal order reductions.

Lemma 6.9. *If* $s \xrightarrow{S,\mathsf{opt}} t$*, then for all* $RED_s \in \mathcal{CON}(s)$ *there exists* $RED_t \in \mathcal{CON}(t)$ *with* $\mathtt{rl}(RED_t) \leq \mathtt{rl}(RED_s)$

Proof. Let $s \xrightarrow{S,\mathsf{opt}} t$ and $RED_s \in \mathcal{CON}(s)$ with length l. We use induction on l to show the existence of $RED_t \in \mathcal{CON}(t)$ with $\mathtt{rl}(RED_t) \leq l$. If $l = 0$ then the claim follows from the Lemma 6.8. If $l > 0$, we apply a forking diagram for (S, opt) from the complete set of Lemma 6.7 to the sequence $\xleftarrow{RED_s} s \xrightarrow{S,\mathsf{opt}} t$. With RED' being the suffix of RED_s of length $l - 1$, we have the cases:



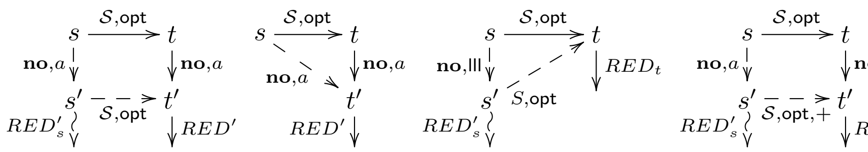
(1) (2) (3) (4)

(1) We can apply the induction hypothesis to RED' and hence have $RED'_t \in \mathcal{CON}(t')$ with $\mathtt{rl}(RED'_t) \leq \mathtt{rl}(RED')$. By appending RED'_t to $t \xrightarrow{\mathbf{no},a} t'$ we have $RED_t \in \mathcal{CON}(t)$ with $\mathtt{rl}(RED_t) \leq \mathtt{rl}(RED_s)$.
(2) There exists $RED_t = \xrightarrow{\mathbf{no},a} . \xrightarrow{RED'}$ and $\mathtt{rl}(RED_t) = l$.
(3) By the induction hypothesis we have $RED_t \in \mathcal{CON}(t)$ with $\mathtt{rl}(RED_t) \leq \mathtt{rl}(RED')$. With $\mathtt{rl}(RED') = \mathtt{rl}(RED_s) + 1$ the claim follows.
(4) We apply the induction hypothesis firstly for RED' and then for every derived normal order reduction leading to $RED'_t \in \mathcal{CON}(t')$ with $\mathtt{rl}(RED'_t) \leq \mathtt{rl}(RED')$. By appending RED'_t to $t \xrightarrow{\mathbf{no},a} t'$ we have $RED_t \in \mathcal{CON}(t)$ with $\mathtt{rl}(RED_t) \leq l$. □

Lemma 6.10. *If $s \xrightarrow{S,\mathsf{opt}} t$ then for all $RED_t \in \mathcal{CON}(t)$ there exists $RED_s \in \mathcal{CON}(s)$ with $\mathtt{rl}_{(\setminus \mathsf{lll})}(RED_s) \leq \mathtt{rl}_{(\setminus \mathsf{lll})}(RED_t)$.*

Proof. We use induction on the following measure μ on reduction sequences $s \xrightarrow{S,\mathsf{opt}} t \xrightarrow{RED}$ with $\mu(s \xrightarrow{S,\mathsf{opt}} t \xrightarrow{RED}) = (\mathtt{rl}_{(\setminus \mathsf{lll})}(RED), \mu_{lll}(s))$. We assume the measure to be ordered lexicographically. If $\mu(s \xrightarrow{S,\mathsf{opt}} t \xrightarrow{RED_t}) = (0,(0,0))$, then from $\mu_{lll}(s) = (0,0)$ follows $\mu_{lll}(t) = (0,0)$ since an (S,opt) transformation does not introduce new `letrec`-expressions. Thus RED_t must be empty and t be a WHNF. From Lemma 6.8 we have that either s is also a WHNF or $s \xrightarrow{\mathbf{no},\mathsf{lll}} s'$ where s' is a WHNF. In both cases we have a (possibly empty) $RED_s \in \mathcal{CON}(S)$ with $\mathtt{rl}_{(\setminus \mathsf{lll})}(RED_s) \leq \mathtt{rl}_{(\setminus \mathsf{lll})}(RED_t)$.

Now, let $\mu(s \xrightarrow{S,\mathsf{opt}} t \xrightarrow{RED_t}) = (l,m) > (0,(0,0))$. W.l.o.g. we assume that RED_t is nonempty, hence we can apply a commuting diagram from the complete set of Lemma 6.7. Let RED' be the suffix of RED_t of length $l-1$, we have the following cases:



(1) (2) (3) (4)

(1) We split into two cases:
   - If the $(\mathbf{no},a)$-reduction is an $(\mathbf{no},\mathsf{lll})$-reduction, then $\mu_{lll}(s') < m$ and $\mathtt{rl}_{(\setminus \mathsf{lll})}(RED') = \mathtt{rl}_{(\setminus \mathsf{lll})}(RED_t)$. Hence we can apply the induction hypothesis to $s' \xrightarrow{S,\mathsf{opt}} t' \xrightarrow{RED'}$ and have $RED'_s \in \mathcal{CON}(s')$ with $\mathtt{rl}_{(\setminus \mathsf{lll})}(RED'_s) \leq \mathtt{rl}_{(\setminus \mathsf{lll})}(RED')$. By appending RED'_s to $s \xrightarrow{\mathbf{no},a} s'$ we have $RED_s \in \mathcal{CON}(s)$ with $\mathtt{rl}_{(\setminus \mathsf{lll})}(RED_s) \leq \mathtt{rl}_{(\setminus \mathsf{lll})}(t)$.
   - If the $(\mathbf{no},a)$-reduction is not an $(\mathbf{no},\mathsf{lll})$-reduction, then $\mathtt{rl}_{(\setminus \mathsf{lll})}(RED') < \mathtt{rl}_{(\setminus \mathsf{lll})}(RED_t)$ and we can apply the induction hypothesis to $s' \xrightarrow{S,\mathsf{opt}} t' \xrightarrow{RED'}$ and have $RED'_s \in \mathcal{CON}(s')$ with $\mathtt{rl}_{(\setminus \mathsf{lll})}(RED'_s) \leq \mathtt{rl}_{(\setminus \mathsf{lll})}(RED')$. Again, we append RED'_s to $s \xrightarrow{\mathbf{no},a} s'$

and have a terminating normal order reduction RED_s for s with $\mathtt{rl}_{(\backslash\mathsf{lll})}(RED_s) \leq \mathtt{rl}_{(\backslash\mathsf{lll})}(RED_t)$.

(2) We have $RED_s = \xrightarrow{\mathbf{no},a}\xrightarrow{RED'}$ and $\mathtt{rl}_{(\backslash\mathsf{lll})}(RED_s) = \mathtt{rl}_{(\backslash\mathsf{lll})}(RED_t)$.

(3) Since $\mu_{lll}(s') < \mu_{lll}(s)$,we can apply the induction hypothesis to $s' \xrightarrow{S,\mathsf{opt}} t \xrightarrow{RED_t}$ and have $RED'_s \in \mathcal{CON}(s')$ with $\mathtt{rl}_{(\backslash\mathsf{lll})}(RED'_s) \leq \mathtt{rl}_{(\backslash\mathsf{lll})}(RED_t)$. By appending RED'_s to $s \xrightarrow{\mathbf{no},\mathsf{lll}} s'$ the claim follows.

(4) Since $\mathtt{rl}_{(\backslash\mathsf{lll})}(RED') < \mathtt{rl}_{(\backslash\mathsf{lll})}(RED_t)$, we can apply the induction hypothesis multiple times for every (S, opt)-transformation leading to $RED'_s \in \mathcal{CON}(s')$ with $\mathtt{rl}_{(\backslash\mathsf{lll})}(RED'_s) \leq l - 1$. By appending RED'_s to $s \xrightarrow{\mathbf{no},a} s'$ we have $RED_s \in \mathcal{CON}(s)$ with $\mathtt{rl}_{(\backslash\mathsf{lll})}(RED_s) \leq l$. □

Lemma 6.11. *If $s \xrightarrow{\mathsf{opt}} t$ then $s \leq_c^{\downarrow} t$ and $t \leq_c^{\downarrow} s$.*

Proof. Let $s \xrightarrow{\mathsf{opt}} t$. Using the context lemma for may-convergence it is sufficient to show $\forall S \in \mathcal{S}$: $S[s]\downarrow \implies S[t]\downarrow$ and $\forall S \in \mathcal{S}$: $S[t]\downarrow \implies S[s]\downarrow$. The first part follows from Lemma 6.9, the second part follows from Lemma 6.10. □

Lemma 6.12. *If $s \xrightarrow{S,\mathsf{opt}} t$ then $s\Uparrow$ iff $t\Uparrow$.*

Proof. Follows from Lemma 6.11 using Lemma 4.5. □

Lemma 6.13. *If $s \xrightarrow{S,\mathsf{opt}} t$, then for all $RED_s \in \mathcal{DIV}(s)$ there exists $RED_t \in \mathcal{DIV}(t)$ with $\mathtt{rl}(RED_t) \leq \mathtt{rl}(RED_s)$*

Proof. Let $s = S[s_0], t = S[t_0]$ with $s_0 \xrightarrow{\mathsf{opt}} t_0$. Let $RED_s \in \mathcal{DIV}(s)$ with $l = \mathtt{rl}(RED_s)$. We show by induction on l that there exists $RED_t \in \mathcal{DIV}(t)$ with $\mathtt{rl}(RED_t) \leq l$. For the base case let RED_s be empty, i.e. $s\Uparrow$, then Lemma 6.12 shows the claim. The induction step uses the same arguments as the proof of Lemma 6.9. □

Lemma 6.14. *If $s \xrightarrow{S,\mathsf{opt}} t$ then for all $RED_t \in \mathcal{DIV}(t)$ there exists $RED_s \in \mathcal{DIV}(s)$ with $\mathtt{rl}_{(\backslash\mathsf{lll})}(RED_s) \leq \mathtt{rl}_{(\backslash\mathsf{lll})}(RED_t)$.*

Proof. The claim follows by induction on the measure μ on reduction sequences $s \xrightarrow{S,\mathsf{opt}} t \xrightarrow{RED}$ with $\mu(s \xrightarrow{S,\mathsf{opt}} t \xrightarrow{RED}) = (\mathtt{rl}_{(\backslash\mathsf{lll})}(RED), \mu_{lll}(s))$. Let the measure be ordered lexicographically. The base case is covered by Lemma 6.12. I.e., let $s = S[s_0]$, $t = S[t_0]$ and $s_0 \xrightarrow{\mathsf{opt}} t_0$, then we have $\mu(s \xrightarrow{S,\mathsf{opt}} t \xrightarrow{RED_t}) = (0, (0, 0))$. Since $\mu_{lll}(s) = (0, 0)$ we have that $\mu_{lll}(t) = (0, 0)$ since an (S, opt) transformation does not introduce new `letrec`-expressions. Thus $\mathtt{rl}(RED_t) = 0$, i.e. $t\Uparrow$ and Lemma 6.12 shows the claim. The induction step uses the same arguments as the proof of Lemma 6.10. □

Lemma 6.15. *If $s \xrightarrow{\mathsf{opt}} t$ then $s \leq_c^{\Downarrow} t$ and $t \leq_c^{\Downarrow} s$.*

Proof. We use Corollary 4.12. We already have $s \leq_c^{\downarrow} t$ and $t \leq_c^{\downarrow} s$ from Lemma 6.11. Hence, it is sufficient to show $\forall S \in \mathcal{S} : S[t]\uparrow \implies S[s]\uparrow$ and $\forall S \in \mathcal{S} : S[s]\uparrow \implies S[t]\uparrow$. Let $s \xrightarrow{\mathsf{opt}} t$ and S be a context with $S[t]\uparrow$. Then with Lemma 6.14 follows that $S[s]\uparrow$. Now, let S be context with $S[s]\uparrow$, then Lemma 6.13 shows that $S[t]\uparrow$. □

Proposition 6.16. (opt) *is a correct program transformation, i.e. if* $s \xrightarrow{\mathsf{opt}} t$ *then* $s \sim_c t$.

Proof. Follows from Lemma 6.11 and Lemma 6.15. □

7 Correctness of Deterministic Reduction Rules

In this section we prove the correctness of the remaining reduction rules.

7.1 Correctness of (case)

With the correctness of (opt) and (case-c) we show the following proposition.

Proposition 7.1. (case) *is a correct program transformation, i.e. if* $s \xrightarrow{\mathsf{case}} t$ *then* $s \sim_c t$.

Proof. From Propositions 6.16 and 5.8 we have that (cpx), (cpcx) and (case-c) keep contextual equivalence. Let $\{x_i = x_{i-1}\}_{i=1}^m$ be the chain which is used by a ($\mathcal{C}$, case-in)- or ($\mathcal{C}$, case-e)-reduction, then every (case-in)-reduction can be replaced by the sequence $\xrightarrow{\mathcal{C},\mathsf{cpx},m-1}\xrightarrow{\mathcal{C},\mathsf{cpcx}}\xrightarrow{\mathcal{C},\mathsf{case\text{-}c}}$:

$$
\begin{array}{ll}
 & \mathtt{letrec}\ x_1 = c\ \overrightarrow{t_i}, \{x_i = x_{i-1}\}_{i=2}^m, Env\ \mathtt{in}\ C[\mathtt{case}_T\ x_m\ \ldots(c\ \overrightarrow{z_i} \to r)\ldots] \\
\xrightarrow{\mathsf{cpx},m-1} & \mathtt{letrec}\ x_1 = c\ \overrightarrow{t_i}, \{x_i = x_{i-1}\}_{i=2}^m, Env\ \mathtt{in}\ C[\mathtt{case}_T\ x_1\ \ldots(c\ \overrightarrow{z_i} \to r)\ldots] \\
\xrightarrow{\mathsf{cpcx}} & \mathtt{letrec}\ x_1 = c\ \overrightarrow{y_i}, \{y_i = t_i\}_{i=1}^{\mathrm{ar}(c)}, \{x_i = x_{i-1}\}_{i=2}^m, Env\ \mathtt{in}\ C[\mathtt{case}_T\ (c\ \overrightarrow{y_i})\ \ldots] \\
\xrightarrow{\mathsf{case\text{-}c}} & \mathtt{letrec}\ x_1 = c\ \overrightarrow{y_i}, \{y_i = t_i\}_{i=1}^{\mathrm{ar}(c)}, \{x_i = x_{i-1}\}_{i=2}^m, Env \\
 & \mathtt{in}\ C[\mathtt{letrec}\ \{z_i = y_i\}_{i=1}^{\mathrm{ar}(c)}\ \mathtt{in}\ r]
\end{array}
$$

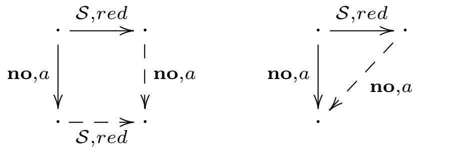
Every ($\mathcal{C}$, case-e)-reduction can also be replaced by the sequence $\xrightarrow{\mathcal{C},\mathsf{cpx},m-1}\xrightarrow{\mathcal{C},\mathsf{cpcx}}\xrightarrow{\mathcal{C},\mathsf{case\text{-}c}}$, where the transformation is analogous to the transformation for ($\mathcal{C}$, case-in). □

7.2 Correctness of (lll)

We will develop complete sets of diagrams for the reductions (lapp), (lcase), (lseq), (lamb) and (llet). Then we combine them to derive complete sets of diagrams for (lll) and finally prove the correctness of (lll)

7.2.1 Diagrams for (lapp), (lcase) and (lseq)

Lemma 7.2. *Let red* $\in$ {lapp, lcase, lseq}*, a complete set of forking diagrams for* $\xrightarrow{S,red}$ *is*

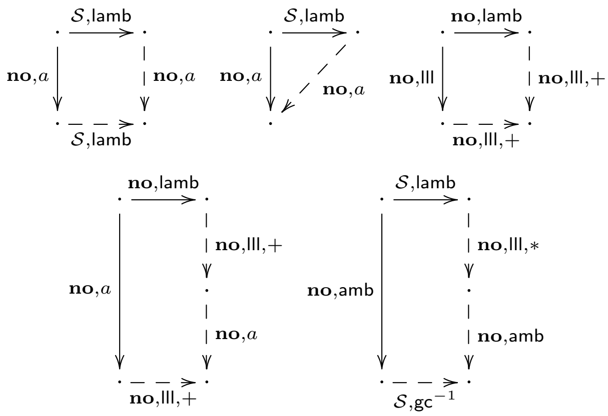


a arbitrary $\quad$ $a \in$ {case, seq, amb}

Proof. A case analysis of all overlappings shows that the reductions either commute, or the redex of the reduction (S, red) is discarded by an normal order reduction. □

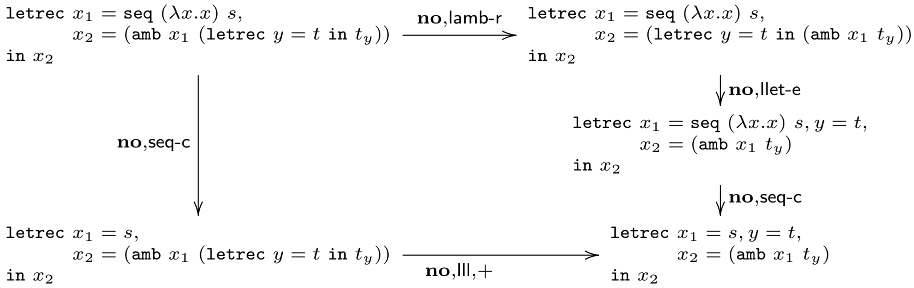
7.2.2 Diagrams for (lamb)

Lemma 7.3. *A complete set of forking diagrams for* $(\mathcal{S}$, lamb$)$ *is*

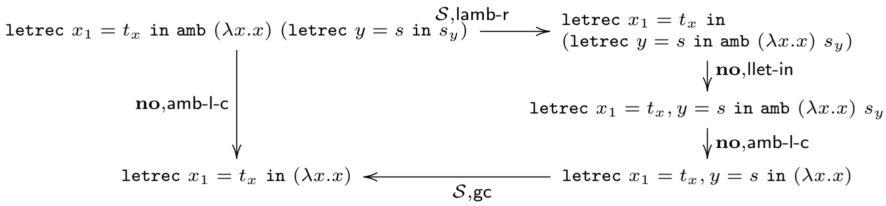


where for the first diagram a is arbitrary, for second diagram $a \in$ {case, seq, amb} *and for the* 4^{th} *diagram* $a \in$ {case, lbeta, cp, seq, amb}.

Proof. Follows by inspecting all cases where an $(S$, lamb$)$-reduction overlaps with a normal order reduction. The first diagram describes the commuting case. The second diagram covers the cases where the $(S$, lamb$)$ redex is discarded by an (**no**, case)-, (**no**, seq)- or (**no**, amb)-reduction. If after performing the $(S$, lamb$)$-reduction, the (inner) redex of the (**no**, a) is no longer inside a reduction context, then the third or the fourth diagram is applicable. Note that in all these case the $(S$, lamb$)$-reduction is also a normal order reduction. An example for these cases is:

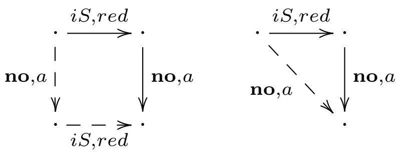


The last diagram covers the cases, where the (S, lamb) redex and the redex of an $(\mathbf{no}, \mathsf{amb})$-reduction are identical, e.g.



□

Lemma 7.4. *A complete set of commuting diagrams for* (iS, red) *with* $red \in \{\mathsf{lapp}, \mathsf{lcase}, \mathsf{lseq}, \mathsf{lamb}\}$ *is*

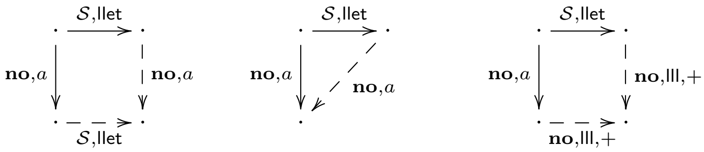


a *arbitrary* $\quad a \in \{\mathsf{case}, \mathsf{seq}, \mathsf{amb}\}$

Proof. This follows by checking all cases where (iS, a) with $a \in \{\mathsf{lapp}, \mathsf{lseq}, \mathsf{lcase}, \mathsf{lamb}\}$ is followed by a normal order reduction. The first diagram covers the cases where the reductions commute. If the normal order reduction is a $(\mathsf{case})-$, $(\mathsf{seq})-$ or (amb)-reduction that discards the `letrec`-expression which is the result of the (iS, a)-reduction, then the second diagram is applicable. □

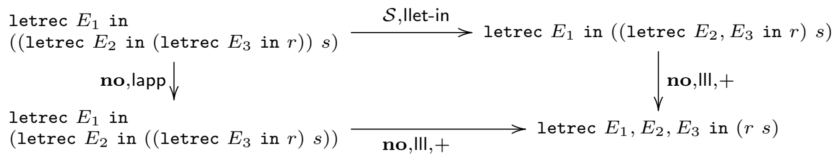
7.2.3 Diagrams for (llet)

Lemma 7.5. *A complete set of forking diagrams for* $(\mathcal{S}, \mathsf{llet})$ *is*



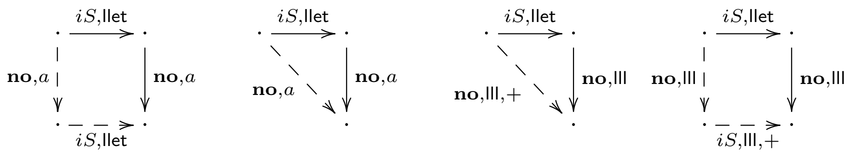
a *arbitrary* $\quad a \in \{\mathsf{case}, \mathsf{seq}, \mathsf{amb}\}$ $a \in \{\mathsf{lapp}, \mathsf{lcase}, \mathsf{lseq}, \mathsf{lamb}\}$

Proof. This follows by inspecting all cases where an (S, llet)-reduction overlaps with a normal order reduction. The first diagram is applicable if the reductions can be commuted. The cases where the (S, llet)-redex is discarded by an $(\mathbf{no}, \mathsf{case})$-, $(\mathbf{no}, \mathsf{seq})$- or $(\mathbf{no}, \mathsf{amb})$-reduction are covered by the second diagram. The third diagram is necessary for the cases that after the (S, llet)-reduction the redex of the $(\mathbf{no}, a)$ with $a \in \{\mathsf{lapp}, \mathsf{lcase}, \mathsf{lseq}, \mathsf{lamb}\}$ is no longer in a reduction context. An example for this case is:



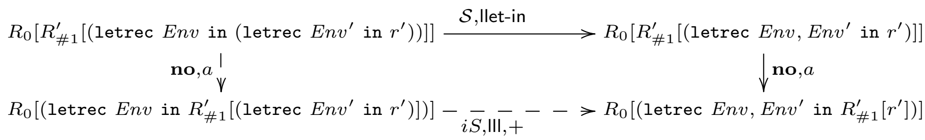
□

Lemma 7.6. *A complete set of commuting diagrams for* $(i\mathcal{S}, \mathsf{llet})$ *is:*

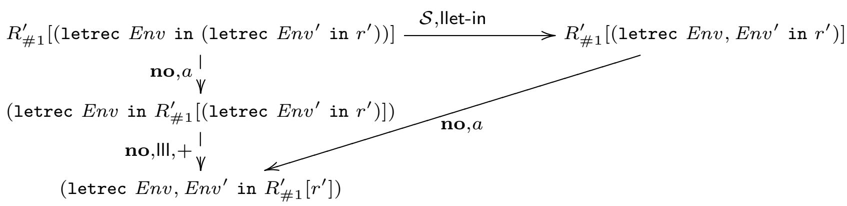


a *arbitrary* $\quad a \in \{\mathsf{case}, \mathsf{seq}, \mathsf{amb}\}$

Proof. This follows by checking all cases where an $\mathcal{S}$-internal (llet)-reduction is followed by a normal order reduction. The first diagram covers the cases where the reductions commute. If the contractum of the (llet)-reduction is discarded by an $(\mathbf{no}, \mathsf{case})$-, $(\mathbf{no}, \mathsf{seq})$- or $(\mathbf{no}, \mathsf{amb})$-reduction, then the second diagram is applicable. The 4th diagram covers the cases where an $(\mathbf{no}, \mathsf{lll})$-reduction overlaps with the (iS, llet)-reduction and the `letrec`-environment needs to be adjusted.



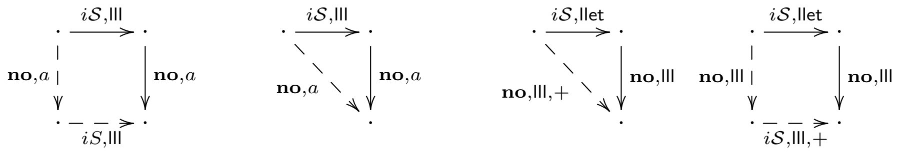
The third diagram is for the same case except that the existential quantified (lll)-reductions are normal order. An example for this case is:



□

7.2.4 Proving Correctness of (lll) Now we can combine the diagrams to derive complete sets of forking and commuting diagrams for (lll).

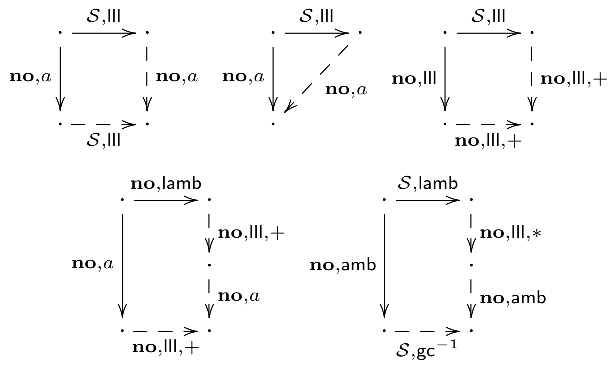
Lemma 7.7. *A complete set of commuting diagrams for* $(i\mathcal{S},\mathsf{lll})$ *is:*



a arbitrary $a \in \{\mathsf{case},\mathsf{seq},\mathsf{amb\text{-}l},\mathsf{amb\text{-}r}\}$

Proof. Follows from Lemma 7.4 and Lemma 7.6. □

Lemma 7.8. *A complete set of forking diagrams for* $(\mathcal{S},\mathsf{lll})$ *is:*



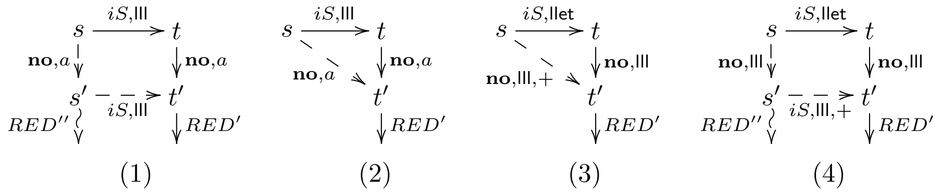
where for the first diagram a is arbitrary, for the second diagram $a \in \{\mathsf{case},\mathsf{seq},\mathsf{amb\text{-}l},\mathsf{amb\text{-}r}\}$ *and for the* 4^{th} *diagram* $a \in \{\mathsf{case},\mathsf{lbeta},\mathsf{cp},\mathsf{seq},\mathsf{amb\text{-}l},\mathsf{amb\text{-}r}\}$.

Proof. Follows from Lemma 7.2, Lemma 7.3 and Lemma 7.5. □

Lemma 7.9. *If* $s \xrightarrow{iS,\mathsf{lll}} t$ *then s is a WHNF iff t is a WHNF.*

Lemma 7.10. *If* $s \xrightarrow{iS,\mathsf{lll}} t$ *then for all* $RED_t \in \mathcal{CON}(t)$ *there exists* $RED_s \in \mathcal{CON}(s)$ *with* $\mathtt{rl}_{(\setminus \mathsf{lll})}(RED_s) \leq \mathtt{rl}_{(\setminus \mathsf{lll})}(RED_t)$.

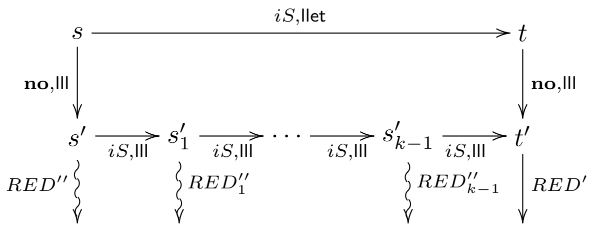
Proof. Let the measure μ on reduction sequences be defined as $\mu(s \xrightarrow{iS,\mathsf{lll}} t \xrightarrow{RED} r) = (\mathtt{rl}_{(\setminus \mathsf{lll})}(RED), \mu_{lll}(s))$ and let μ be ordered lexicographically. For the base case let $\mu(s \xrightarrow{iS,\mathsf{lll}} t \xrightarrow{RED_t} r) = (0,(1,1))$ The measure $\mu_{lll}(s)$ cannot be smaller than (0,(1,1)), since otherwise no (iS,lll)-reduction would be possible. If $\mu(s \xrightarrow{iS,\mathsf{lll}} t \xrightarrow{RED_t} r) = (0,(1,1))$ the sequence RED cannot contain $(\mathbf{no},\mathsf{lll})$ reductions and thus RED is empty. Then t is a WHNF and with Lemma 7.9 s is also a WHNF. Now, let $\mu(s \xrightarrow{iS,\mathsf{lll}} t \xrightarrow{RED_t}) > (0,(1,1))$. We apply a diagram from the complete set of commuting diagrams for (iS, lll) to a prefix of $s \xrightarrow{iS,\mathsf{lll}} t \xrightarrow{RED_t}$. With RED' being RED_t without the first reduction, the cases are:



Since cases (2) and (3) are trivial we only show the other cases:

(1) If the $(\mathbf{no}, a)$-reduction is an $(\mathbf{no}, \mathsf{lll})$-reduction then the we can apply the induction hypothesis to the sequence $s' \xrightarrow{iS,\mathsf{lll}} t' \xrightarrow{RED'}$ since from $s \xrightarrow{\mathbf{no},\mathsf{lll}} s'$ follows that $\mu_{lll}(s') < \mu_{lll}(s)$ and $\mathtt{rl}_{(\setminus \mathsf{lll})}(RED') = \mathtt{rl}_{(\setminus \mathsf{lll})}(RED_t)$. Hence, we have $RED'' \in \mathcal{CON}(s')$ with $\mathtt{rl}_{(\setminus \mathsf{lll})}(RED'') \leq \mathtt{rl}_{(\setminus \mathsf{lll})}(RED')$. By appending RED'' to $s \xrightarrow{\mathbf{no},\mathsf{lll}} s'$ we have $RED_s \in \mathcal{CON}(s)$ with $\mathtt{rl}_{(\setminus \mathsf{lll})}(RED_s) = \mathtt{rl}_{(\setminus \mathsf{lll})}(RED'')$.
If the $(\mathbf{no}, a)$-reduction is not an $(\mathbf{no}, \mathsf{lll})$-reduction, then we also can apply the induction hypothesis to the sequence $s' \xrightarrow{iS,\mathsf{lll}} t' \xrightarrow{RED'}$ since $\mathtt{rl}_{(\setminus \mathsf{lll})}(RED) > \mathtt{rl}_{(\setminus \mathsf{lll})}(RED')$. Hence we have $RED'' \in \mathcal{CON}(s')$ with $\mathtt{rl}_{(\setminus \mathsf{lll})}(RED'') \leq \mathtt{rl}_{(\setminus \mathsf{lll})}(RED')$. By appending RED'' to $s \xrightarrow{\mathbf{no},a} s'$ we have $RED_s \in \mathcal{CON}(s)$ with $\mathtt{rl}_{(\setminus \mathsf{lll})}(RED_s) \leq \mathtt{rl}_{(\setminus \mathsf{lll})}(RED_t)$.

(4) If the 4^{th} diagram is applied, then the prefix $s \xrightarrow{iS,\mathsf{llet}} t \xrightarrow{\mathbf{no},\mathsf{lll}}$ is replaced by $s \xrightarrow{\mathbf{no},\mathsf{lll}} s' \xrightarrow{iS,\mathsf{lll},k} t'$ for some $k > 0$, i.e. there exist terms $s'_1 \dots s'_{k-1}$ with



It holds that $\mu_{lll}(s) > \mu_{lll}(s') > \mu_{lll}(s'_1) > \ldots > \mu_{lll}(s'_{k-1}) > \mu_{lll}(t')$. Since $\mathtt{rl}_{(\setminus \mathsf{lll})}(RED_t) = \mathtt{rl}_{(\setminus \mathsf{lll})}(RED')$, we can apply the induction hypothesis to the sequence $s'_{k-1} \xrightarrow{iS,\mathsf{lll}} t' \xrightarrow{RED'}$ and have $RED''_{k-1} \in \mathcal{CON}(s'_{k-1})$ with $\mathtt{rl}_{(\setminus \mathsf{lll})}(RED''_{k-1}) \leq \mathtt{rl}_{(\setminus \mathsf{lll})}(RED')$. Now, we can apply the induction hypothesis multiple times for every s'_i and finally for s', i.e. we have $RED'' \in \mathcal{CON}(s')$ with $\mathtt{rl}_{(\setminus \mathsf{lll})}(RED'') \leq \mathtt{rl}_{(\setminus \mathsf{lll})}(RED')$. By appending RED'' to $s \xrightarrow{\mathbf{no},\mathsf{lll}} s'$, we have $RED_s \in \mathcal{CON}(s)$ with $\mathtt{rl}_{(\setminus \mathsf{lll})}(RED_s) \leq \mathtt{rl}_{(\setminus \mathsf{lll})}(RED)$.

□

Lemma 7.11. *If $s \xrightarrow{S,\mathsf{lll}} t$ then for all $RED_s \in \mathcal{CON}(s)$ there exists $RED_t \in \mathcal{CON}(t)$ with $\mathtt{rl}_{(\setminus \mathsf{lll})}(RED_t) \leq \mathtt{rl}_{(\setminus \mathsf{lll})}(RED_s)$.*

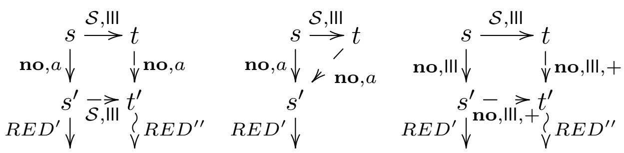
Proof. We use induction on a measure μ on reduction sequences with $\mu(\xleftarrow{RED_s} s \xrightarrow{S,\mathsf{lll}} t) = (\mathtt{rl}_{(\backslash \mathsf{lll})}(RED_s), \mu_{lll}(s))$ and the measure is ordered lexicographically.

Let $s \xrightarrow{S,\mathsf{lll}} t$ and $RED_s \in \mathcal{CON}(s)$ with $\mathtt{rl}_{(\backslash \mathsf{lll})}(RED_s) = l$. If $\mu_{lll}(s) = (1,1)$ then either s is already in WHNF and Lemma 7.9 shows the claim, or RED_s consists of exactly one $(\mathbf{no}, \mathsf{lll})$-reduction which leads to t, i.e. the reduction is the same as the (S, lll)-reduction, then the claim trivially follows.

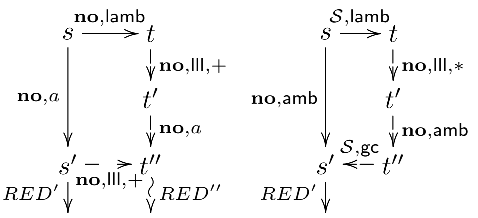
For the induction step, we use the complete set of forking diagrams for (S, lll) of Lemma 7.8. If no diagram is applicable to RED_s then there are two possibilities:

- s is already in WHNF, then the claim follows from Lemma 7.9 and Lemma 2.12.
- The first reduction of RED_s is the same reduction as the (S, lll)-reduction. By dropping the first reduction from RED_s we have $RED_t \in \mathcal{CON}(t)$ with $\mathtt{rl}_{(\backslash \mathsf{lll})}(RED_t) = l$.

Otherwise, with RED' being the suffix of RED_s with length $l-1$ we have the cases:



(1) (2) (3)



(4) (5)

(1) For the first diagram we split into two cases:
   - The reduction $s \xrightarrow{\mathbf{no},a} s'$ is not an $(\mathbf{no}, \mathsf{lll})$-reduction. Since $\mathtt{rl}_{(\backslash \mathsf{lll})}(RED') = 1 + \mathtt{rl}_{(\backslash \mathsf{lll})}(RED_s)$, we can apply the induction hypothesis, i.e. there exists $RED'' \in \mathcal{CON}(t')$ with $\mathtt{rl}_{(\backslash \mathsf{lll})}(RED'') \leq \mathtt{rl}_{(\backslash \mathsf{lll})}(RED')$. By appending RED'' to $t \xrightarrow{\mathbf{no},a} t'$ we have $RED_t \in \mathcal{CON}(t)$ with $\mathtt{rl}_{(\backslash \mathsf{lll})}(RED_t) \leq \mathtt{rl}_{(\backslash \mathsf{lll})}(RED_s)$.
   - The reduction $s \xrightarrow{\mathbf{no},a} s'$ is an $(\mathbf{no}, \mathsf{lll})$-reduction. Since $\mathtt{rl}_{(\backslash \mathsf{lll})}(RED_s) = \mathtt{rl}_{(\backslash \mathsf{lll})}(RED')$ and $\mu_{lll}(s) > \mu_{lll}(s')$ we can apply the induction hypothesis and have a reduction sequence $RED'' \in \mathcal{CON}(t')$ with $\mathtt{rl}_{(\backslash \mathsf{lll})}(RED'') \leq \mathtt{rl}_{(\backslash \mathsf{lll})}(RED')$.By appending RED'' to $t \xrightarrow{\mathbf{no},a} t'$ we have $RED_t \in \mathcal{CON}(t)$ with $\mathtt{rl}_{(\backslash \mathsf{lll})}(RED_t) \leq \mathtt{rl}_{(\backslash \mathsf{lll})}(RED_s)$.

(2) For the second diagram the existence of $RED_t \in \mathcal{CON}(t)$ with $\mathtt{rl}_{(\backslash \mathsf{lll})}(RED_t) \leq \mathtt{rl}_{(\backslash \mathsf{lll})}(RED_s)$ is obvious.
(3) Since $\mathtt{rl}_{(\backslash \mathsf{lll})}(RED) = \mathtt{rl}_{(\backslash \mathsf{lll})}(RED')$ but $\mu_{lll}(s') < \mu_{lll}(s)$ and (lll) decreases the measure μ_{lll} strictly, we can an apply the induction hypothesis multiple times for every ($\mathbf{no}, \mathsf{lll}$)-reduction in the sequence $s' \xrightarrow{\mathbf{no},\mathsf{lll},+} t'$. Thus, we have $RED'' \in \mathcal{CON}(t')$ with $\mathtt{rl}_{(\backslash \mathsf{lll})}(RED'') \leq \mathtt{rl}_{(\backslash \mathsf{lll})}(RED')$. By appending RED'' to $t \xrightarrow{\mathbf{no},\mathsf{lll},+} t'$ we have $RED_t \in \mathcal{CON}(t)$ with $\mathtt{rl}_{(\backslash \mathsf{lll})}(RED_t) \leq l$.
(4) Since the first reduction of RED_s is not an (lll)-reduction, $\mathtt{rl}_{(\backslash \mathsf{lll})}(RED') = l - 1$. Hence, we can use the induction hypothesis multiple times for every (lll)-reduction in $s' \xrightarrow{\mathbf{no},\mathsf{lll},+} t''$ and derive $RED'' \in \mathcal{CON}(t')$ with $\mathtt{rl}_{(\backslash \mathsf{lll})}(RED'') \leq l - 1$. By appending RED'' to $t \xrightarrow{\mathbf{no},\mathsf{lll},+} t' \xrightarrow{\mathbf{no},a} t''$ we have $RED_t \in \mathcal{CON}(t)$ with $\mathtt{rl}_{(\backslash \mathsf{lll})}(RED_t) \leq l$.
(5) We can construct the reduction sequence $t \xrightarrow{\mathbf{no},\mathsf{lll},*} t' \xrightarrow{\mathbf{no},\mathsf{amb}} t'' \xrightarrow{S,\mathsf{gc}} s' \xrightarrow{RED'}$ where $RED' \in \mathcal{CON}(s')$. From Lemma 6.10 we have that there exists $RED'' \in \mathcal{CON}(t'')$ with $\mathtt{rl}_{(\backslash \mathsf{lll})}(RED'') \leq \mathtt{rl}_{(\backslash \mathsf{lll})}(RED')$. Hence, we have the sequence $RED_t = t \xrightarrow{\mathbf{no},\mathsf{lll},*} t' \xrightarrow{\mathbf{no},\mathsf{amb}} t'' \xrightarrow{RED''}$ with $\mathtt{rl}_{(\backslash \mathsf{lll})}(RED_t) \leq \mathtt{rl}_{(\backslash \mathsf{lll})}(RED_s)$.

□

Lemma 7.12. *If $s \xrightarrow{\mathsf{lll}} t$ then $s \leq_c^{\downarrow} t$ and $t \leq_c^{\downarrow} s$.*

Proof. Using the context lemma it is sufficient to show, that if $s_0 \xrightarrow{\mathsf{lll}} t_0$ then $\forall S \in \mathcal{S}$: $S[s_0]\downarrow \implies S[t_0]\downarrow$ and $\forall S \in \mathcal{S}$: $S[t_0]\downarrow \implies S[s_0]\downarrow$. For the first part we split into two cases: If $S[s_0] \xrightarrow{\mathsf{lll}} S[t_0]$ is a normal order reduction then the claim is obvious, otherwise the claim follows from Lemma 7.11. The second part follows from Lemma 7.10. □

Lemma 7.13. *If $s \xrightarrow{S,\mathsf{lll}} t$ then $s\Uparrow$ iff $t\Uparrow$.*

Proof. Follows from Lemma 7.12 using Lemma 4.5. □

Lemma 7.14. *If $s \xrightarrow{iS,\mathsf{lll}} t$ then for all $RED_t \in \mathcal{DIV}(t)$ there exists $RED_s \in \mathcal{DIV}(s)$ with $\mathtt{rl}_{(\backslash \mathsf{lll})}(RED_s) \leq \mathtt{rl}_{(\backslash \mathsf{lll})}(RED_t)$.*

Proof. Let μ be a measure on reduction sequences $s \xrightarrow{iS,\mathsf{lll}} t \xrightarrow{RED} r$ with $\mu(s \xrightarrow{iS,\mathsf{lll}} t \xrightarrow{RED} r) = (\mathtt{rl}_{(\backslash \mathsf{lll})}(RED), \mu_{lll}(s))$. We use induction on the measure μ, ordered lexicographically. $\mu_{lll}(s)$ cannot be smaller than $(1,1)$, since otherwise no (iS, lll)-reduction would be possible. If $\mu(s \xrightarrow{iS,\mathsf{lll}} t \xrightarrow{RED}) = (0,(1,1))$ then $\mu_{lll}(t) < (1,1)$, hence the RED cannot contain ($\mathbf{no}, \mathsf{lll}$)-reductions and thus RED is empty, i.e. $t\Uparrow$. Then Lemma 7.13 shows the claim. The induction step is analogous as in the proof of Lemma 7.10. □

Lemma 7.15. *If $s \xrightarrow{S,\mathsf{lll}} t$ then for all $RED_s \in \mathcal{DIV}(s)$ there exists $RED_t \in \mathcal{DIV}(t)$ with $\mathtt{rl}_{(\backslash \mathsf{lll})}(RED_t) \leq \mathtt{rl}_{(\backslash \mathsf{lll})}(RED_s)$.*

Proof. This follows by induction on a lexicographically ordered measure μ defined as $\mu(\xleftarrow{RED_s} s \xrightarrow{S,\mathsf{lll}} t) = (\mathtt{rl}_{(\backslash \mathsf{lll})}(RED_s), \mu_{lll}(s))$. The base case follows from Lemma 7.13, and the induction uses the forking diagrams for $(\mathcal{S}, \mathsf{lll})$. $\square$

Lemma 7.16. *If $s \xrightarrow{\mathsf{lll}} t$ then $s \leq_c^{\Downarrow} t$ and $t \leq_c^{\Downarrow} s$.*

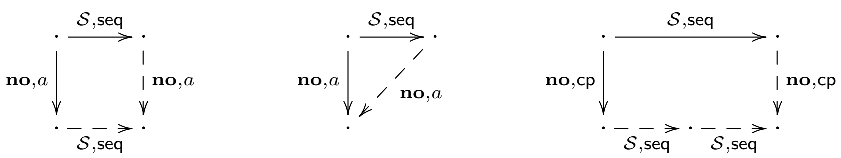
Proof. We use Corollary 4.12. We already have $s \leq_c^{\downarrow} t$ and $t \leq_c^{\downarrow} s$ from Lemma 7.12. For the remaining part we show $\forall S \in \mathcal{S} : S[t]\uparrow \implies S[s]\uparrow$ and $\forall S \in \mathcal{S} : S[s]\uparrow \implies S[t]\uparrow$. Let $s \xrightarrow{\mathsf{lll}} t$ and S be a context with $S[t]\uparrow$. If $S[s] \xrightarrow{S,\mathsf{lll}} S[t]$ is a normal order reduction, we have $S[s]\uparrow$. Otherwise, the claim follows from Lemma 7.14. Now let S be context with $S[s]\uparrow$, then Lemma 7.15 shows that $S[t]\uparrow$. $\square$

Proposition 7.17. (lll) *is a correct program transformation, i.e. if $s \xrightarrow{\mathsf{lll}} t$ then $s \sim_c t$.*

Proof. Follows from Lemma 7.12 and Lemma 7.16. $\square$

7.3 Correctness of (seq)

Lemma 7.18. *A complete set of forking diagrams for $(\mathcal{S}, \mathsf{seq})$ is*



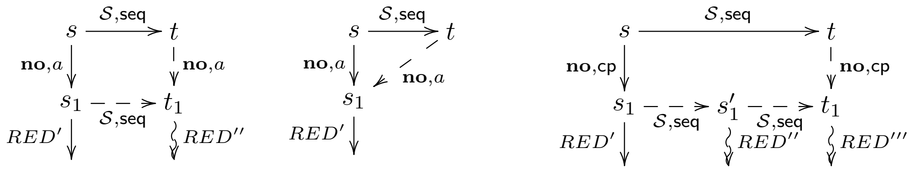
a arbitrary $a \in \{\mathsf{case}, \mathsf{seq}, \mathsf{amb\text{-}l}, \mathsf{amb\text{-}r}\}$

Proof. The reductions commute, or the (S, seq) is discarded by a normal order reduction, or if the inner redex of the (S, seq) is in the body of an abstraction, which is copied by an $(\mathbf{no}, \mathsf{cp})$-reduction, then two (S, seq)-reductions are necessary. $\square$

Lemma 7.19. *If $s \xrightarrow{iS,\mathsf{seq}} t$ then s is a WHNF iff t is a WHNF.*

Lemma 7.20. *If $s \xrightarrow{S,\mathsf{seq}} t$ then for all $RED_s \in \mathcal{CON}(s)$ there exists $RED_t \in \mathcal{CON}(t)$ with $\mathtt{rl}(RED_t) \leq \mathtt{rl}(RED_s)$.*

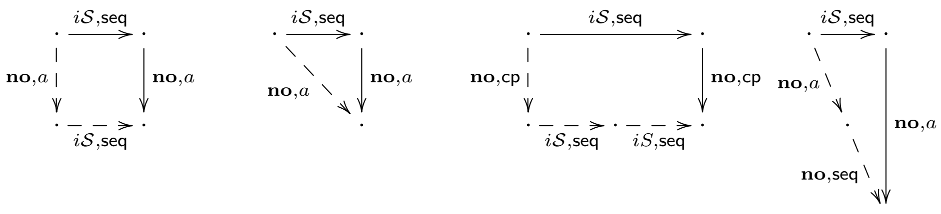
Proof. Let $s \xrightarrow{S,\mathsf{seq}} t$ and $RED_s \in \mathcal{CON}(s)$ with $\mathtt{rl}(RED_s) = l$, we use induction on l. If $l = 0$ then s is a WHNF and the claim follows from Lemma 7.19. If $l > 0$ then let RED' be the suffix of RED_s of length $l - 1$. If the first reduction of RED_s is the same as the (S, seq)-reduction, then $RED' \in \mathcal{CON}(t)$. Otherwise, we apply a forking diagram to a suffix of $\xleftarrow{RED_s} s \xrightarrow{S,\mathsf{seq}} t$ and have the cases:



(1) (2) (3)

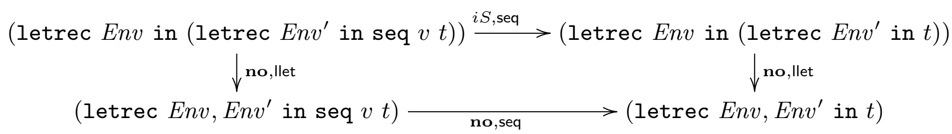
For case (1), we can apply the induction hypothesis to $\xleftarrow{RED'} s_1 \xrightarrow{S,\mathsf{seq}} t_1$. Case (2) is trivial. For case (3) we apply the induction hypothesis twice, i.e. firstly to $\xleftarrow{RED'} s_1 \xrightarrow{S,\mathsf{seq}} s_1'$ and secondly to $\xleftarrow{RED''} s_1' \xrightarrow{S,\mathsf{seq}} t_1$. □

Lemma 7.21. *A complete set of commuting diagrams for* $(i\mathcal{S}, \mathsf{seq})$ *is*



a arbitrary $a \in \{\mathsf{case}, \mathsf{seq}, \mathsf{amb\text{-}l}, \mathsf{amb\text{-}r}\}$ *a arbitrary*

Proof. The first three diagrams describe the same cases as for the forking diagrams. The second diagram is applicable, if the (iS, seq)-reduction becomes normal order, e.g.

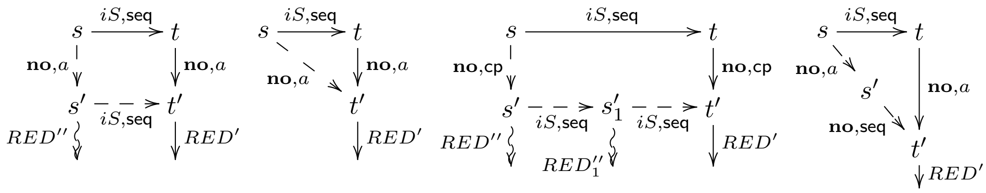


□

Lemma 7.22. *If* $s \xrightarrow{iS,\mathsf{seq}} t$ *then for every* $RED_t \in \mathcal{CON}(t)$ *there exists* $RED_s \in \mathcal{CON}(s)$ *with* $\mathtt{rl}_{(\backslash \mathsf{seq})}(RED_s) \leq \mathtt{rl}_{(\backslash \mathsf{seq})}(RED_t)$.

Proof. Let $s \xrightarrow{iS,\mathsf{seq}} t$ and $RED_t \in \mathcal{CON}(t)$, we use induction on the measure μ ordered lexicographically with $\mu(RED) = (\mathtt{rl}_{(\backslash \mathsf{seq})}(RED), \mathtt{rl}(RED))$. If $\mathtt{rl}(RED_t) = 0$ then the claim follows from Lemma 7.19. If $\mu(RED_t) = (l, m) \geq (0, 1)$, i.e. RED_t contains at least one normal order reduction, then we apply a commuting diagram from Lemma 7.21 to a prefix of $s \xrightarrow{iS,\mathsf{seq}} t \xrightarrow{RED_t}$. With

RED' being the suffix of RED_t of length $(m-1)$ we have the cases:



(1) (2) (3) (4)

For case (1) we apply the induction hypothesis to $s' \xrightarrow{iS,\mathsf{seq}} t' \xrightarrow{RED'}$. Cases (2) and (4) are trivial. For case (3) we apply the induction hypothesis twice. ◻

Lemma 7.23. *$s \xrightarrow{\mathsf{seq}} t$ then, $s \leq_c^{\downarrow} t$ and $t \leq_c^{\downarrow} s$.*

Proof. Using the context lemma for may-convergence, it is sufficient to show that $\forall S \in \mathcal{S}$: $S[s]{\downarrow} \implies S[t]{\downarrow}$ and $\forall S \in \mathcal{S}$: $S[t]{\downarrow} \implies S[s]{\downarrow}$. The first part follows from Lemma 7.20. For the second part let $S[t]{\downarrow}$. If the reduction $S[s] \xrightarrow{S,\mathsf{seq}} S[t]$ is a normal order reduction, then the claim follows trivially, otherwise the claim follows from Lemma 7.22. ◻

Lemma 7.24. *If $s \xrightarrow{S,\mathsf{seq}} t$ then $s{\Uparrow}$ iff $t{\Uparrow}$.*

Proof. Follows from Lemma 7.23 using Lemma 4.5. ◻

Lemma 7.25. *If $s \xrightarrow{S,\mathsf{seq}} t$ then for all $RED_s \in \mathcal{DIV}(s)$ there exists $RED_t \in \mathcal{DIV}(t)$ with $\mathtt{rl}(RED_t) \leq \mathtt{rl}(RED_s)$.*

Proof. Let $s = S[s_0], t = S[t_0]$, $s_0 \xrightarrow{\mathsf{seq}} t_0$ and $RED_s \in \mathcal{DIV}(s)$ with $\mathtt{rl}(RED_s) = l$. The claim follows by induction on l, where the base case is covered by Lemma 7.24 and the induction step is analogous to the proof of Lemma 7.20. ◻

Lemma 7.26. *If $s \xrightarrow{iS,\mathsf{seq}} t$ then for every $RED_t \in \mathcal{DIV}(t)$ there exists $RED_s \in \mathcal{DIV}(s)$ with $\mathtt{rl}_{(\setminus \mathsf{seq})}(RED_s) \leq \mathtt{rl}_{(\setminus \mathsf{seq})}(RED_t)$*

Proof. Let $s = S[s_0], t = S[t_0]$, $s_0 \xrightarrow{\mathsf{seq}} t_0$ and $RED_t \in \mathcal{DIV}(t)$, we use induction on the measure $\mu(RED_t)$ ordered lexicographically, where $\mu(RED) = (\mathtt{rl}_{(\setminus \mathsf{seq})}(RED), \mathtt{rl}(RED))$. If $\mathtt{rl}(RED_t) = 0$, then the claim follows from Lemma 7.24. The induction step is analogous to the proof of Lemma 7.22. ◻

Lemma 7.27. *If $s \xrightarrow{\mathsf{seq}} t$ then $s \leq_c^{\Downarrow} t$ and $t \leq_c^{\Downarrow} s$.*

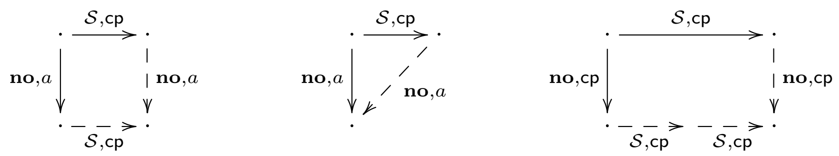
Proof. We use Corollary 4.12. We already have $s \leq_c^{\downarrow} t$ and $t \leq_c^{\downarrow} s$ from Lemma 7.23, hence it is sufficient to show $\forall S \in \mathcal{S} : S[s]{\uparrow} \implies S[t]{\uparrow}$ and $\forall S \in \mathcal{S} : S[t]{\uparrow} \implies S[s]{\uparrow}$. The first part follows from Lemma 7.25. The second part follows from Lemma 7.26 or follows trivially if $S[s] \xrightarrow{S,\mathsf{seq}} S[t]$ is a normal order reduction. ◻

Proposition 7.28. (seq) *is a correct program transformation, i.e. if* $s \xrightarrow{\mathsf{seq}} t$ *then,* $s \sim_c t$.

Proof. Follows from Lemma 7.23 and Lemma 7.27. □

7.4 Correctness of (cp)

Lemma 7.29. *A complete set of forking diagrams for* $(\mathcal{S}, \mathsf{cp})$ *is*



a arbitrary $a \in \{\mathsf{case}, \mathsf{seq}, \mathsf{amb\text{-}l}, \mathsf{amb\text{-}r}\}$

Proof. The reductions commute, or the redex of the target of the (S, cp)-reduction is discarded by a normal order reduction, or the (S, cp)-reduction copies into the body of an abstraction that is copied by an $(\mathbf{no}, \mathsf{cp})$-reduction. In detail: Follows by inspecting all cases where an (S, cp) and a normal order reduction overlaps. The first diagram is applicable if both reductions commute. If the redex or the target of the (S, cp) is discarded by an $(\mathbf{no}, \mathsf{case})$-, $(\mathbf{no}, \mathsf{seq})$- or $(\mathbf{no}, \mathsf{amb})$-reduction, then the second diagram is applicable. The third diagram covers the cases, where the (S, cp)-reduction copies into the body of an abstraction that is copied by a normal order (cp)-reduction. □

Lemma 7.30. *If* $s \xrightarrow{iS,\mathsf{cp}} t$*, then* s *is a WHNF iff* t *is a WHNF.*

Lemma 7.31. *If* $s \xrightarrow{S,\mathsf{cp}} t$*, then for all* $RED_s \in \mathcal{CON}(s)$ *there exists* $RED_t \in \mathcal{CON}(t)$ *with* $\mathtt{rl}(RED_t) \leq \mathtt{rl}(RED_s)$.

Proof. The proof is a copy of the proof of Lemma 7.20 using the complete set of forking diagrams for (S, cp) from Lemma 7.29 and using Lemma 7.30. □

For the other direction, i.e. $s \xrightarrow{\mathsf{cp}} t$, then $t \leq_c^{\downarrow} s$, we distinguish to kinds of (cp)-reductions:

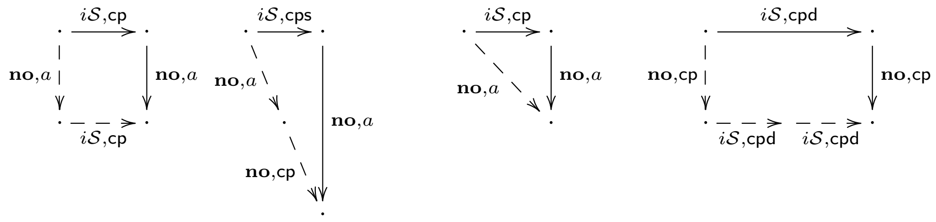
(cps) := the inner redex of the (cp) is of the form $S[x]$, i.e. the target is inside a surface context.

(cpd) := the inner redex of the (cp) is of the form $C[\lambda z.C'[x]]$, i.e. the target is inside the body of an abstraction.

Definition 7.32. *Let* s *be a term, then* $\mu_{Sx}(s)$ *is the number of occurrences of variables in* s *where the occurrence is inside a surface context.*

Lemma 7.33. *Every* $(\mathcal{S}, \mathsf{cps})$*- or* $(\mathbf{no}, \mathsf{cp})$*-reduction strictly reduces the measure* μ_{Sx}*. Every* $(\mathcal{S}, \mathsf{cpd})$*-reduction does not change the measure* μ_{Sx}.

Lemma 7.34. *A complete set of commuting diagrams for* $(i\mathcal{S}, \mathsf{cp})$ *is*



a arbitrary *a arbitrary* $a \in \{\mathsf{case}, \mathsf{seq}, \mathsf{amb\text{-}l}, \mathsf{amb\text{-}r}\}$

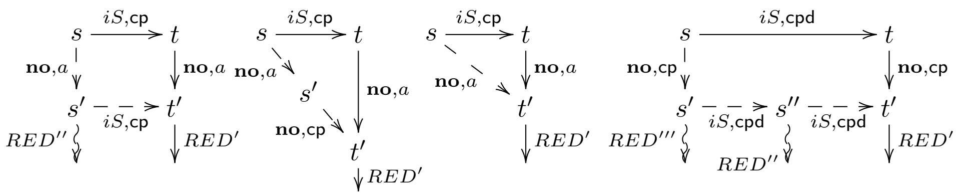
Proof. Follows by inspecting all cases where a (iS, cp)-reduction is followed by a normal order reduction. The first diagram describe the case where the reductions commute. If the $\mathcal{S}$-internal (cp)-reduction becomes normal order, then the second diagram is applicable. The third diagram describes the case where the contractum of the $(i\mathcal{S}, \mathsf{cp})$-reduction is discarded by a normal order reduction. The last diagram covers the cases, where the target of an $(i\mathcal{S}, \mathsf{cpd})$-reduction is inside the body of an abstraction that is copied by an $(\mathbf{no}, \mathsf{cp})$-reduction. An example for the second diagram is:

$$\begin{array}{ll} & \texttt{letrec } y = (\lambda x.s) \texttt{ in } (\texttt{seq } (\lambda z.z)\ y) \\ \xrightarrow{iS,\mathsf{cps}} & \texttt{letrec } y = (\lambda x.s) \texttt{ in } (\texttt{seq } (\lambda z.z)\ (\lambda x'.s[x'/x])) \\ \xrightarrow{\mathbf{no},\mathsf{seq}} & \texttt{letrec } y = (\lambda x.s) \texttt{ in } (\lambda x'.s[x'/x]) \\ \hline & \texttt{letrec } y = (\lambda x.s) \texttt{ in } (\texttt{seq } (\lambda z.z)\ y) \\ \xrightarrow{\mathbf{no},\mathsf{seq}} & \texttt{letrec } y = (\lambda x.s) \texttt{ in } y \\ \xrightarrow{\mathbf{no},\mathsf{cp}} & \texttt{letrec } y = (\lambda x.s) \texttt{ in } (\lambda x'.s[x'/x]) \end{array}$$

□

Lemma 7.35. *If* $s \xrightarrow{iS,\mathsf{cp}} t$ *then for all* $RED_t \in \mathcal{CON}(t)$ *there exists* $RED_s \in \mathcal{CON}(s)$ *with* $\mathtt{rl}_{(\backslash \mathsf{cp})}(RED_s) \leq \mathtt{rl}_{(\backslash \mathsf{cp})}(RED_t)$

Proof. Let $s \xrightarrow{iS,\mathsf{cp}} t$ and $RED_t \in \mathcal{CON}(t)$. The claim follows by induction on the measure μ on reduction sequences with $\mu(s \xrightarrow{iS,\mathsf{cp}} t \xrightarrow{RED_t}) = (\mathtt{rl}_{(\backslash \mathsf{cp})}(RED_t), \mu_{Sx}(t))$. If $\mu(s \xrightarrow{iS,\mathsf{cp}} t \xrightarrow{RED_t}) = (0,0)$, then from Lemma 7.33 we have that RED_t must be empty. Thus Lemma 7.30 shows the claim. Now, let $\mu(s \xrightarrow{iS,\mathsf{cp}} t \xrightarrow{RED_t}) = (l,m) > (0,0)$. We apply a commuting diagram to a prefix of $s \xrightarrow{iS,\mathsf{cp}} t \xrightarrow{RED_t}$. With RED' being the suffix of RED_t where the first reduction is dropped, we have the cases



(1) (2) (3) (4)

(1) For the first diagram we have
 - $\mu_{Sx}(t') < \mu_{Sx}(t)$ and $\mathtt{rl}_{(\backslash \mathsf{cp})}(RED') = l$ if the $(\mathbf{no}, a)$-reduction is an $(\mathbf{no}, \mathsf{cp})$-reduction, or
 - $\mathtt{rl}_{(\backslash \mathsf{cp})}(RED') < l$, if the $(\mathbf{no}, a)$-reduction is not an $(\mathbf{no}, \mathsf{cp})$-reduction.

 In both cases, we can apply the induction hypothesis to $s' \xrightarrow{iS,b} t' \xrightarrow{RED'}$ and have $RED'' \in \mathcal{CON}(s')$ with $\mathtt{rl}_{(\backslash \mathsf{cp})}(RED'') \leq \mathtt{rl}_{(\backslash \mathsf{cp})}(RED')$. By appending RED'' to $s \xrightarrow{\mathbf{no},a} s'$ we have $RED_s \in \mathcal{CON}(s)$ with $\mathtt{rl}_{(\backslash \mathsf{cp})}(RED_s) \leq l$.

(2) The sequence $RED_s = \xrightarrow{\mathbf{no},a}\xrightarrow{\mathbf{no},\mathsf{cp}}\xrightarrow{RED'}$ is in $\mathcal{CON}(s)$ and $\mathtt{rl}(RED_s) \leq l$.

(3) We can construct the sequence $RED_s = \xrightarrow{\mathbf{no},a}\xrightarrow{RED'}$ with $RED_s \in \mathcal{CON}(s)$ and $\mathtt{rl}_{(\backslash \mathsf{cp})}(RED_s) \leq l$.

(4) Since $t \xrightarrow{\mathbf{no},\mathsf{cp}} t'$ we have with Lemma 7.33 that $\mu_{Sx}(t') < m$. From $s' \xrightarrow{iS,\mathsf{cpd}} s''$ and $\xrightarrow{iS,\mathsf{cpd}} t'$ we also have with Lemma 7.33 that $\mu_{Sx}(s'') = \mu_{Sx}(s') < m$. Since $\mathtt{rl}_{(\backslash \mathsf{cp})}(RED') = l$ we can apply the induction hypothesis to $s'' \xrightarrow{is,\mathsf{cpd}} t' \xrightarrow{RED'}$ and have $RED'' \in \mathcal{CON}(s'')$ with $\mathtt{rl}_{(\backslash \mathsf{cp})}(RED'') \leq l$. Since $\mu(s'') < m$ we can apply the induction hypothesis for a second time to $s' \xrightarrow{iS,\mathsf{cpd}} s'' \xrightarrow{RED''}$ and have $RED''' \in \mathcal{CON}(s')$ with $\mathtt{rl}_{(\backslash \mathsf{cp})}(RED''') \leq l$. Finally we append RED''' to $s \xrightarrow{\mathbf{no},\mathsf{cp}} s'$ and have $RED_s \in \mathcal{CON}(s)$ with $\mathtt{rl}_{(\backslash \mathsf{cp})}(RED_s) \leq l$.

□

Lemma 7.36. *If $s \xrightarrow{\mathsf{cp}} t$ then $s \leq_c^{\downarrow} t$ and $t \leq_c^{\downarrow} s$.*

Proof. Let $s \xrightarrow{\mathsf{cp}} t$, using Lemma 4.10 it is sufficient to show $\forall S \in \mathcal{S} : S[s]{\downarrow} \implies S[t]{\downarrow}$ and $\forall S \in \mathcal{S} : S[t]{\downarrow} \implies S[s]{\downarrow}$. The first part follows from Lemma 7.31. The second part follows from Lemma 7.35 or follows trivially if $S[s] \xrightarrow{S,\mathsf{cp}} S[t]$ is normal order. □

Lemma 7.37. *If $s \xrightarrow{S,\mathsf{cp}} t$ then $s{\Uparrow}$ iff $t{\Uparrow}$.*

Proof. Follows from Lemma 7.36 using Lemma 4.5. □

Lemma 7.38. *If $s \xrightarrow{S,\mathsf{cp}} t$, then for all $RED_s \in \mathcal{DIV}(s)$ there exists $RED_t \in \mathcal{DIV}(t)$ with $\mathtt{rl}(RED_t) \leq \mathtt{rl}(RED_s)$*

Proof. The proof is a copy of the proof of Lemma 7.25 using the complete set of forking diagrams for (S, cp) from Lemma 7.29 and for the base case using Lemma 7.37. □

Lemma 7.39. *If $s \xrightarrow{iS,\mathsf{cp}} t$ then for all $RED_t \in \mathcal{DIV}(t)$ there exists $RED_s \in \mathcal{DIV}(s)$ with $\mathtt{rl}_{(\backslash \mathsf{cp})}(RED_s) \leq \mathtt{rl}_{(\backslash \mathsf{cp})}(RED_t)$*

Proof. The proof is analogous to the proof of Lemma 7.35 using Lemma 7.37.

□

Lemma 7.40. *If $s \xrightarrow{\mathsf{cp}} t$ then $s \leq_c^{\Downarrow} t$ and $t \leq_c^{\Downarrow} s$.*

Proof. From Lemma 7.36 we have $s \leq_c^{\downarrow} t$ and $t \leq_c^{\downarrow} s$. Using Corollary 4.12 it is sufficient to show that $\forall S \in \mathcal{S} : S[s]\uparrow \implies S[t]\uparrow$ and $\forall S \in \mathcal{S} : S[t]\uparrow \implies S[s]\uparrow$. The first part follows from Lemma 7.38. The second part follows from Lemma 7.39 if $S[s] \xrightarrow{S,\mathsf{cp}} S[t]$ is $\mathcal{S}$-internal and otherwise the claim is trivial. □

Proposition 7.41. (cp) *is a correct program transformation, i.e. if* $s \xrightarrow{\mathsf{cp}} t$ *then* $s \sim_c t$.

Proof. Follows from Lemma 7.36 and Lemma 7.40 □

8 The Standardisation Theorem and an Application

We summarise the results of the previous sections.

Theorem 8.1. *All deterministic reductions of the calculus* $\Lambda^{\mathtt{let}}_{\mathtt{amb}}$ *keep contextual equivalence, i.e. if* $s \xrightarrow{a} t$ *with* $a \in \{\mathsf{lbeta}, \mathsf{lll}, \mathsf{case}, \mathsf{seq}, \mathsf{cp}\}$ *then* $s \sim_c t$.

Proof. Follows from the Propositions 5.8, 7.17, 7.1, 7.28 and 7.41. □

We will now develop properties of the reduction (amb), that will be necessary for the proof of the Standardisation Theorem (Theorem 8.12).

8.1 Properties of the Reduction (amb)

Lemma 8.2. *If* $s \xrightarrow{i\mathcal{S},\mathsf{amb}} t$ *then s is a WHNF iff t is a WHNF.*

Proof. Follows by definition of WHNFs. □

The following lemma shows that it is sufficient to consider (amb-c)-reductions.

Lemma 8.3. *Let* s, t *be terms with* $s \xrightarrow{\mathcal{C},\mathsf{amb\text{-}in}} t$ *or* $s \xrightarrow{\mathcal{C},\mathsf{amb\text{-}e}} t$. *Then either* $s \xrightarrow{\mathcal{C},\mathsf{cp}} \xrightarrow{\mathcal{C},\mathsf{amb\text{-}c}} \xleftarrow{\mathcal{C},\mathsf{cp}} t$ *or* $s \xrightarrow{\mathcal{C},\mathsf{cpx},*} \xrightarrow{\mathcal{C},\mathsf{cpcx}} \xrightarrow{\mathcal{C},\mathsf{amb\text{-}c}} \xleftarrow{\mathcal{C},\mathsf{cpcx}} \xleftarrow{\mathcal{C},\mathsf{cpx},*} t$.

Proof. We show the transformation for a toplevel (amb-in)-reduction, the other cases are analogous: In case of a constructor application:

$$
\begin{array}{ll}
 & \mathtt{letrec}\ x_1 = c\ \overrightarrow{t_i}, \{x_i = x_{i-1}\}_{i=2}^m, \mathit{Env}\ \mathtt{in}\ C[\mathtt{amb}\ x_m\ s] \\
\xrightarrow{\mathsf{cpx},m-1} & \mathtt{letrec}\ x_1 = c\ \overrightarrow{t_i}, \{x_i = x_{i-1}\}_{i=2}^m, \mathit{Env}\ \mathtt{in}\ C[\mathtt{amb}\ x_1\ s] \\
\xrightarrow{\mathsf{cpcx}} & \mathtt{letrec}\ x_1 = c\ \overrightarrow{y_i}, \{y_i = t_i\}_{i=1}^{\mathrm{ar}(c)}, \{x_i = x_{i-1}\}_{i=2}^m, \mathit{Env}\ \mathtt{in}\ C[\mathtt{amb}\ (c\ \overrightarrow{y_i})\ s] \\
\xrightarrow{\mathsf{amb\text{-}c}} & \mathtt{letrec}\ x_1 = c\ \overrightarrow{y_i}, \{y_i = t_i\}_{i=1}^{\mathrm{ar}(c)}, \{x_i = x_{i-1}\}_{i=2}^m, \mathit{Env}\ \mathtt{in}\ C[(c\ \overrightarrow{y_i})] \\
\xleftarrow{\mathsf{cpcx}} & \mathtt{letrec}\ x_1 = c\ \overrightarrow{t_i}, \{x_i = x_{i-1}\}_{i=2}^m, \mathit{Env}\ \mathtt{in}\ C[x_1] \\
\xleftarrow{\mathsf{cpx},m-1} & \mathtt{letrec}\ x_1 = c\ \overrightarrow{t_i}, \{x_i = x_{i-1}\}_{i=2}^m, \mathit{Env}\ \mathtt{in}\ C[x_m]
\end{array}
$$

In case of an abstraction:

$$
\begin{array}{ll}
 & \mathtt{letrec}\ x_1 = (\lambda x.t), \{x_i = x_{i-1}\}_{i=2}^m, \mathit{Env}\ \mathtt{in}\ C[\mathtt{amb}\ x_m\ s] \\
\xrightarrow{\mathsf{cp}} & \mathtt{letrec}\ x_1 = (\lambda x.t), \{x_i = x_{i-1}\}_{i=2}^m, \mathit{Env}\ \mathtt{in}\ C[\mathtt{amb}\ (\lambda x.t)\ s] \\
\xrightarrow{\mathsf{amb\text{-}c}} & \mathtt{letrec}\ x_1 = (\lambda x.t), \{x_i = x_{i-1}\}_{i=2}^m, \mathit{Env}\ \mathtt{in}\ C[(\lambda x.t)] \\
\xleftarrow{\mathsf{cp}} & \mathtt{letrec}\ x_1 = (\lambda x.t), \{x_i = x_{i-1}\}_{i=2}^m, \mathit{Env}\ \mathtt{in}\ C[x_m]
\end{array}
$$

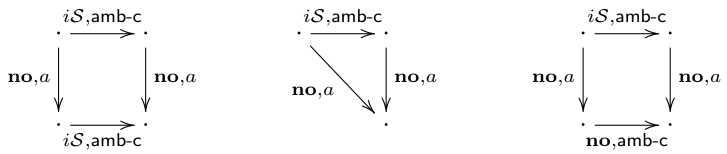
□

Lemma 8.4. *If* $s \xrightarrow{\mathsf{amb}} t$ *then* $t \leq_c^{\downarrow} s$.

Proof. For (amb-c) the claim has been proved in Lemma 5.1. For (amb-in) or (amb-e) we replace the reduction using Lemma 8.3. Then Theorem 8.1, Proposition 6.16 and Lemma 5.1 show the claim. □

Remark 8.5. An (amb)-reduction may introduce must-convergence, e.g. consider the terms $s \equiv$ `case`$_{Bool}$ (`amb True False`) (`True` $\rightarrow \Omega$) (`False` $\rightarrow$ `False`) and $t \equiv$ `case`$_{Bool}$ `False` (`True` $\rightarrow \Omega$) (`False` $\rightarrow$ `False`). While $t\Downarrow$, s may reduce to Ω, i.e. $\neg(s\Downarrow)$.

Lemma 8.6. *A complete set of commuting diagrams and a complete set of forking diagrams for* ($i\mathcal{S}$, amb-c) *can be read off the following diagrams*



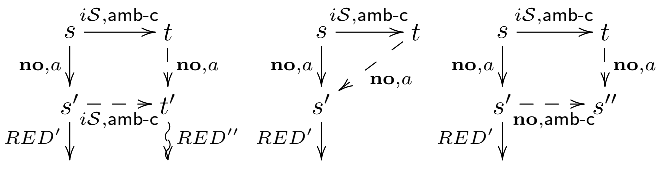
a arbitrary $\quad a \in \{\mathsf{case}, \mathsf{seq}, \mathsf{amb\text{-}l}, \mathsf{amb\text{-}r}\}$ $\quad$ *a arbitrary*

Proof. Follows by case analysis. The reductions either commute or the redex of the (S, amb-c) is discarded, or the internal (amb-c) reduction becomes normal order. □

Remark 8.7. A complete set of forking diagrams for ($\mathcal{S}$, amb-c) does not exist. There are forks that cannot be closed, e.g. $0 \xleftarrow{\mathbf{no},\mathsf{amb\text{-}l}}$ (`amb 0 1`) $\xrightarrow{\mathbf{no},\mathsf{amb\text{-}r}} 1$. Nevertheless, the following lemmas hold for (amb-c)-reductions within all surface contexts.

Lemma 8.8. *If* $s \xrightarrow{\mathcal{S},\mathsf{amb\text{-}c}} t$ *then* $s\Downarrow \implies t\downarrow$

Proof. Let $s = S[s_0], t = S[t_0]$ with $s_0 \xrightarrow{\mathsf{amb\text{-}c}} t_0$ and $s\Downarrow$. We use induction on $l = \mathtt{rl}(RED_s)$ with $RED_s \in \mathcal{CON}(s)$. If the ($S$, amb-c)-reduction is a normal order reduction, then $t\Downarrow$. Let the (amb-c)-reduction be $\mathcal{S}$-internal. If $l = 0$ then Lemma 8.2 shows the claim. If $l > 0$ then we apply a forking diagram from Lemma 8.6 to $\xleftarrow{RED_s} s \xrightarrow{i\mathcal{S},\mathsf{amb}} t$. Let RED' be the suffix of RED_s of length $l-1$, then we have the cases:



(1) (2) (3)

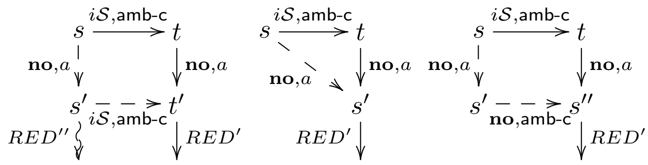
(1) From $s\Downarrow$ we have $s'\Downarrow$. Since $\mathtt{rl}(RED') = l - 1$ we can apply the induction hypothesis to $\xleftarrow{RED'} s' \xrightarrow{i\mathcal{S},\mathsf{amb\text{-}c}} t'$ and have $t'\downarrow$ and hence $t\downarrow$.

(2) From $s\Downarrow$ we have $s'\Downarrow$ and hence $t\downarrow$.
(3) From $s\Downarrow$ we have $s'\Downarrow$ as well as $s''\Downarrow$. Since $t \xrightarrow{n} s''$ we have $t\downarrow$.

□

Lemma 8.9. *If* $s \xrightarrow{\mathcal{S},\textsf{amb-c}} t$ *then* $t\uparrow \implies s\uparrow$.

Proof. Let $s \xrightarrow{\mathcal{S},\textsf{amb-c}} t$, then the claim can be shown by induction on the length of $RED \in \mathcal{DIV}(t)$ using Lemma 8.8 and the commuting diagrams for $(i\mathcal{S}, \textsf{amb-c})$. Let $s = S[s_0], t = S[t_0]$ with $s_0 \xrightarrow{\textsf{amb-c}} t_0$ and $t\uparrow$. We use induction $l = \texttt{rl}(RED_t)$ with $RED_t \in \mathcal{DIV}(t)$. If the (amb-c)-reduction is normal order, then the claim follows immediately. Let the (amb-c)-reduction be $\mathcal{S}$-internal, if $l = 0$, i.e. $t\Uparrow$, then Lemma 8.8 shows the claim. If $l > 0$ then we apply a commuting diagram from Lemma 8.6 to $s \xrightarrow{i\mathcal{S}} t \xrightarrow{RED_t}$. Let RED' be the suffix of RED_t of length $l-1$, then we have the following cases:



(1) (2) (3)

In case (1) we apply the induction hypothesis to $s' \xrightarrow{i\mathcal{S},\textsf{amb-c}} t' \xrightarrow{RED'}$ and hence have that $s'\uparrow$. With $s \xrightarrow{\mathbf{no},a} s'$ we have $s\uparrow$. Cases (2) and (3) are trivial. □

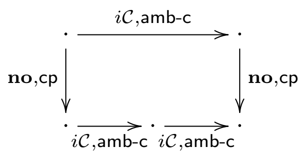
Analogously to (cps), let (ambs) be the reduction (amb) where the inner redex of (amb-in) or (amb-e) is inside a surface context.

Lemma 8.10. *If* $s \xrightarrow{\mathcal{S},\textsf{ambs}} t$ *then* $t\uparrow \implies s\uparrow$

Proof. Follows from Lemma 8.9 and 8.3 using Theorem 8.1 and Proposition 6.16.

□

Remark 8.11. We are not able to prove Lemma 8.10 for all contexts. The context lemma for must-convergence does not help, since the $s \leq_c^{\downarrow} t$ does not hold. If we used diagrams for $(i\mathcal{C}, \textsf{amb-c})$ instead of $(i\mathcal{S}, \textsf{amb-c})$, then we would have the additional diagram



Including this diagram the induction in the proof of Lemma 8.8 does not work.

8.2 The Standardisation Theorem

We define the transformation (corr) as the union of the reductions and transformations that have been shown correct. The transformation (allr) is then the union of (ambs), (corr) and the inverse of (corr):

$$\begin{aligned}(\mathsf{corr}) &:= (\mathsf{lll}) \cup (\mathsf{lbeta}) \cup (\mathsf{seq}) \cup (\mathsf{case}) \cup (\mathsf{opt})\\ (\mathsf{allr}) &:= (\mathsf{corr}) \cup (\mathsf{corr})^{-1} \cup (\mathsf{ambs})\end{aligned}$$

The next theorem shows that for every converging sequence consisting of all defined reductions and transformations there exists a normal order reduction sequence that converges and also that for every diverging sequence of reductions inside surface contexts there exists a normal order reduction sequence that diverges.

Theorem 8.12 (Standardisation).

1. *Let t be a term with $t \xrightarrow{\mathcal{C},\mathsf{allr},*} t'$ where t' is a WHNF, then $t\downarrow$.*
2. *Let t be a term with $t \xrightarrow{\mathcal{S},\mathsf{allr},*} t'$ where $t'\Uparrow$, then $t\uparrow$.*

Proof. 1. Let $t \equiv t_0 \xrightarrow{\mathcal{C},red_1} t_1 \xrightarrow{\mathcal{C},red_2} \dots \xrightarrow{\mathcal{C},red_{k-1}} t_k \equiv t'$ where t' is a WHNF. Using Theorem 8.1, Proposition 6.16 and Lemma 8.4 we have for every $t_i \xrightarrow{\mathcal{C},red_{i+1}} t_{i+1}$ that if $t_{i+1}\downarrow$ then $t_i\downarrow$. Using induction on k we can show $t_0\downarrow$.

2. Let $t \equiv t_0 \xrightarrow{\mathcal{S},red_1} t_1 \xrightarrow{\mathcal{S},red_2} \dots \xrightarrow{\mathcal{S},red_{k-1}} t_k \equiv t'$ where $t'\Uparrow$. With Theorem 8.1, Proposition 6.16 and Lemma 8.10 we have for every $t_i \xrightarrow{\mathcal{S},red_{i+1}} t_{i+1}$ that if $t_{i+1}\uparrow$ then $t_i\uparrow$. Using induction on k we can show $t_0\uparrow$. □

8.3 Proving Bottom-Avoidance

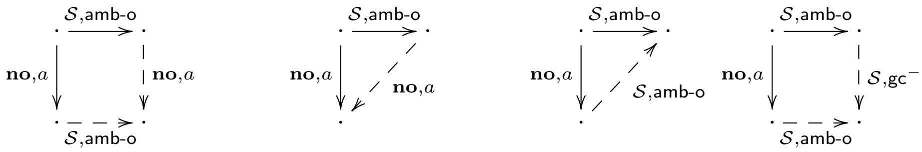
Definition 8.13 (Ω-term). *A term t is called an* Ω-term *in a context S, if either $t \equiv \Omega$ (see Example 4.2), or t is a variable x and S contains a* `letrec`*-binding $x = x$.*

We define bottom-avoidance of `amb` as a program transformation:

(amb-l-o)	(`amb` s t) $\rightarrow s$, if t is an Ω-term.
(amb-r-o)	(`amb` s t) $\rightarrow t$, if s is an Ω-term.

Let (amb-o) be the union of (amb-l-o) and (amb-r-o). Now we will show the correctness of the transformation (amb-o). We proceed again by using commuting and forking diagrams for the correctness proofs. Note, that there exists a complete set of forking diagrams for (S, amb-o) in contrast to the (amb-c)-reduction, since the case of Remark 8.7 is not possible because one argument of the `amb`-expression cannot reduce to a value. From now on we extend the definition of forking and commuting diagrams by allowing also the transformation ($\mathcal{S}$, allr) instead of normal order reductions in the existential quantified reductions on the left and on the right of the diagrams. This is sufficient since Theorem 8.12 shows that for these cases also a normal order reduction exists.

Lemma 8.14. *A complete set of forking diagrams for* $(\mathcal{S}, \mathsf{amb\text{-}o})$ *is:*



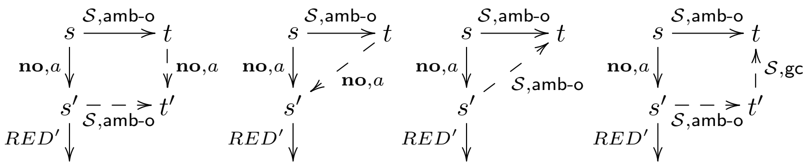
a arbitrary $\quad a \in \{\mathsf{case}, \mathsf{seq}, \mathsf{amb\text{-}l}, \mathsf{amb\text{-}r}\}$ $\quad$ *a arbitrary* $\quad$ *a arbitrary*

Proof. Follows by inspecting all cases where an $(S, \mathsf{amb\text{-}o})$-transformation and a normal order reduction overlap. The first diagram covers the cases where the reductions are performed independently. If the redex of the $(S, \mathsf{amb\text{-}o})$-transformation is discarded by a normal order (case)-, (seq)- or (amb)-reduction, then the second diagram is applicable. The third diagram covers the cases where an $(\mathbf{no}, \mathsf{lamb})$ floats out an `letrec`-environment of the argument of the `amb`-expression that is the result of the (amb-o)-reduction. The last diagram covers the cases where an `letrec`-environment is floated out of the argument of the `amb`-expression that is discarded by the (amb-o)-transformation. □

Lemma 8.15. *Let* s, t *be terms with* $s \xrightarrow{iS,\mathsf{amb\text{-}o}} t$ *then* s *is a WHNF iff* t *is a WHNF.*

Lemma 8.16. *Let* s, t *be terms with* $s \xrightarrow{S,\mathsf{amb\text{-}o}} t$ *and* $s\downarrow$ *then* $t\downarrow$

Proof. We show by induction on the length l of a reduction sequence $RED_s \in \mathcal{CON}(s)$, that there exists a sequence of $(\mathcal{S}, \mathsf{allr})$-transformations starting with t that leads to a WHNF. Then Theorem 8.12 part 1 shows that $t\downarrow$. The base case follows from Lemma 8.15. If $l > 0$ and the first reduction of RED_s is same reduction as the $(S, \mathsf{amb\text{-}o})$-transformation, the claim holds. Otherwise, we apply a forking diagram from Lemma 8.14 to $\xleftarrow{RED_s} s \xrightarrow{S,\mathsf{amb\text{-}o}}$. Let RED' be the suffix of RED_s of length $l - 1$, we have the following cases:



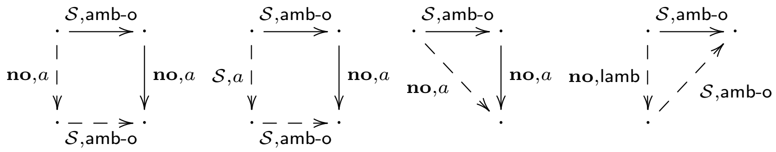
(1) (2) (3) (4)

Case (2) is trivial, for the remaining cases we apply the induction hypothesis to $\xleftarrow{RED'} s' \xrightarrow{\mathcal{S},\mathsf{amb\text{-}o}}$ and derive a sequence of $(\mathcal{S}, \mathsf{allr})$-transformations for t that ends in a WHNF. Hence, we obtain a sequence of $(\mathcal{S}, \mathsf{allr})$-transformations that starts with t'. By appending this sequence to $t \xrightarrow{\mathbf{no},a} t'$, $t \xrightarrow{\mathcal{S},a} t'$, $t \xleftarrow{\mathcal{S},\mathsf{gc}} t'$ we derive a sequence of $(\mathcal{S}, \mathsf{allr})$ reductions starting with t. □

Lemma 8.17. *If* $s \xrightarrow{\mathsf{amb\text{-}o}} t$ *then* $s \leq_c^{\downarrow} t$

Proof. Follows from Lemma 8.16 by using the context lemma for may-convergence. □

Lemma 8.18. *A complete set of commuting diagrams for* $(i\mathcal{S}, \mathsf{amb\text{-}o})$ *is:*



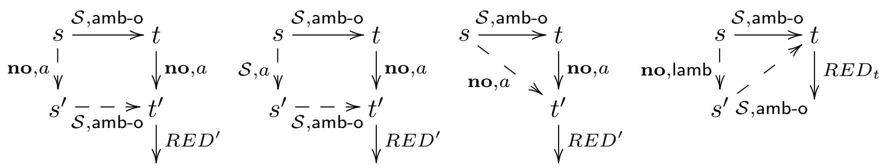
Proof. By inspecting all overlappings there are the cases: If the reductions are commutable, then the first diagram is applicable. If the normal order reduction becomes only normal order because the `amb`-expression has been eliminated, then the second diagram is applicable. The third diagram covers the cases where the the result of the $(\mathsf{amb\text{-}o})$-transformation is eliminated by a normal order reduction. The last diagram covers the case where an $(\mathbf{no}, \mathsf{llet})$ is applicable to the result of the $(\mathcal{S}, \mathsf{amb\text{-}o})$-transformation but only because the `amb`-expression has been eliminated, e.g.

$$\begin{array}{rl} & \texttt{letrec}\ \mathit{Env}_1\ \texttt{in amb}\ \Omega\ ((\texttt{letrec}\ \mathit{Env}_2\ \texttt{in}\ s)) \\ \xrightarrow{\mathcal{S},\mathsf{amb\text{-}o}} & \texttt{letrec}\ \mathit{Env}_1\ \texttt{in}\ ((\texttt{letrec}\ \mathit{Env}_2\ \texttt{in}\ s)) \\ \hline & \texttt{letrec}\ \mathit{Env}_1\ \texttt{in amb}\ \Omega\ ((\texttt{letrec}\ \mathit{Env}_2\ \texttt{in}\ s)) \\ \xrightarrow{\mathbf{no},\mathsf{lamb}} & \texttt{letrec}\ \mathit{Env}_1\ \texttt{in}\ (\texttt{letrec}\ \mathit{Env}_2\ \texttt{in amb}\ \Omega\ s) \\ \xrightarrow{\mathcal{S},\mathsf{amb\text{-}o}} & \texttt{letrec}\ \mathit{Env}_1\ \texttt{in}\ (\texttt{letrec}\ \mathit{Env}_2\ \texttt{in}\ s) \end{array}$$

□

Lemma 8.19. *Let* s, t *be terms with* $s \xrightarrow{\mathcal{S},\mathsf{amb\text{-}o}} t$ *and* $t\downarrow$, *then* $s\downarrow$.

Proof. Let $s = S[s_0]$ and $t = S[t_0]$ with $s \xrightarrow{\mathcal{S},\mathsf{amb\text{-}o}} t$ and $t\downarrow$. We show by induction on the lexicographically ordered measure (a, b) where $b = \mu_{lll}(s)$ and $a = \mathtt{rl}(RED_t)$ where $RED_t \in \mathcal{CON}(t)$, that there exists a sequence of $(\mathcal{S}, \mathsf{allr})$-transformations starting from s that ends in a WHNF. Then from Theorem 8.12 part 1 follows that $s\downarrow$. If the $(\mathcal{S}, \mathsf{amb\text{-}o})$ is also a normal order reduction, then $s\downarrow$. Otherwise, the base case is covered by Lemma 8.15 and for the induction step we apply a commuting diagram from Lemma 8.18 to $s \xrightarrow{\mathcal{S},\mathsf{amb\text{-}o}} t \xrightarrow{RED_t}$. With RED' being the suffix of RED_t of length $a - 1$ we have the cases:



(1) (2) (3) (4)

Case (3) is trivial, in cases (1) and (2) we apply the induction hypothesis to $s' \xrightarrow{\mathcal{S},\mathsf{amb\text{-}o}} t' \xrightarrow{RED'}$. For case (4), $\mu_{lll}(s') < \mu_{lll}(s)$ holds. Hence, we apply the induction hypothesis to s', and append the derived sequence for s' to the reduction $s \xrightarrow{\textbf{no},\mathsf{lamb}} s'$. □

Lemma 8.20. *If* $s \xrightarrow{\mathsf{amb\text{-}o}} t$ *then* $t \leq_c^{\downarrow} s$.

Proof. Follows from Lemma 8.19 by using the context lemma for may-convergence. □

Lemma 8.21. *If* $s \xrightarrow{\mathcal{S},\mathsf{amb\text{-}o}} t$ *then* $s{\Uparrow}$ *iff* $t{\Uparrow}$.

Proof. Follows from Lemma 8.17 and Lemma 8.20. □

Lemma 8.22. *If* $s \xrightarrow{\mathcal{S},\mathsf{amb\text{-}o}} t$ *and* $t{\uparrow}$, *then* $s{\uparrow}$.

Proof. Induction on the measure (a, b) where $b = \mu_{lll}(s)$ and $a = \mathtt{rl}(RED_t)$ with $RED_t \in \mathcal{DIV}(t)$ shows that there exists a sequence of $(\mathcal{S}, \mathsf{allr})$-transformations starting with s and ending in a term that must-diverge. The base case is covered by Lemma 8.21 and the induction step uses the commuting diagrams from Lemma 8.18. As final step Theorem 8.12 part 2 shows that $s{\uparrow}$. □

Lemma 8.23. *If* $s \xrightarrow{\mathcal{S},\mathsf{amb\text{-}o}} t$ *and* $s{\uparrow}$, *then* $t{\uparrow}$.

Proof. Induction on $\mathtt{rl}(RED_s)$ with $RED_s \in \mathcal{DIV}(s)$ shows the existence of a sequence of $(\mathcal{S}, \mathsf{allr})$-transformations from t to a term that must-diverge. The base case for this induction is covered by Lemma 8.21, the induction step uses the forking diagrams from Lemma 8.14. The last step uses Theorem 8.12 part 2 to transform the sequence of $(\mathcal{S}, \mathsf{allr})$-transformations into a normal order reduction sequence $RED_t \in \mathcal{DIV}(t)$. □

Lemma 8.24. *If* $s \xrightarrow{\mathsf{amb\text{-}o}} t$ *then* $s \leq_c^{\Downarrow} t$ *and* $t \leq_c^{\Downarrow} s$.

Proof. Both parts follow by using Corollary 4.12. For $s \leq_c^{\Downarrow} t$ we use Lemma 8.20 and 8.22, and for $t \leq_c^{\Downarrow} s$ we use Lemma 8.17 and 8.23. □

Proposition 8.25. *If* $s \xrightarrow{\mathsf{amb\text{-}o}} t$ *then* $s \sim_c t$.

Proof. Follows from Lemma 8.20, 8.17 and Lemma 8.24. □

8.4 On the Relation Between $\leq_c^{\downarrow}$ and $\leq_c^{\Downarrow}$

A consequence of the bottom-avoidance of `amb` is that $s \leq_c^{\Downarrow} t$ implies $t \leq_c^{\downarrow} s$, which we will show by similar arguments as [Mor98,Las98]. Let the context BA be defined as $BA \equiv (\mathtt{amb}\ \mathbf{I}\ (\mathtt{seq}\ [\cdot]\ (\lambda x.\Omega)))\ \mathbf{I}$.

Lemma 8.26. $BA[s]{\Downarrow}$ *iff* $s{\Uparrow}$.

Proof.
$\neg(s{\Uparrow}) \implies \neg(BA[s]{\Downarrow})$:

Let $\neg(s\Uparrow)$, i.e. $s\downarrow$. Let $RED_s \in \mathcal{CON}(s)$ and let RED_s end in a WHNF s'. We firstly perform the reduction inside the context BA, i.e. we construct the sequence $BA[s] \xrightarrow{BA,RED_s} BA[s']$. Then there are the following cases:

- s' is a value, then

$$BA[s'] \xrightarrow{\mathcal{S},\mathsf{seq}} (\mathtt{amb}\ \mathbf{I}\ (\lambda x.\Omega))\ \mathbf{I} \xrightarrow{\mathcal{S},\mathsf{amb}} (\lambda x.\Omega)\ I$$
$$\xrightarrow{\mathsf{lbeta}} (\mathtt{letrec}\ x = \mathbf{I}\ \mathtt{in}\ \Omega) \xrightarrow{\mathsf{gc}} \Omega$$

- $s' \equiv (\mathtt{letrec}\ \mathit{Env}\ \mathtt{in}\ v)$ where v is a value, then

$$\begin{aligned}
&(\mathtt{amb}\ \mathbf{I}\ (\mathtt{seq}\ (\mathtt{letrec}\ \mathit{Env}\ \mathtt{in}\ v)\ (\lambda x.\Omega)))\ \mathbf{I}\\
\xrightarrow{\mathcal{S},\mathsf{lseq}}\ &(\mathtt{amb}\ \mathbf{I}\ ((\mathtt{letrec}\ \mathit{Env}\ \mathtt{in}\ \mathtt{seq}\ v\ (\lambda x.\Omega))))\ \mathbf{I}\\
\xrightarrow{\mathcal{S},\mathsf{seq}}\ &(\mathtt{amb}\ \mathbf{I}\ ((\mathtt{letrec}\ \mathit{Env}\ \mathtt{in}\ (\lambda x.\Omega))))\ \mathbf{I}\\
\xrightarrow{\mathcal{S},\mathsf{gc}}\ &(\mathtt{amb}\ \mathbf{I}\ (\lambda x.\Omega))\ \mathbf{I}\\
\xrightarrow{\mathsf{amb}}\ &(\lambda x.\Omega)\ \mathbf{I}\\
\xrightarrow{\mathsf{lbeta}}\ &(\mathtt{letrec}\ x = \mathbf{I}\ \mathtt{in}\ \Omega)\\
\xrightarrow{\mathsf{gc}}\ &\Omega
\end{aligned}$$

- $s' \equiv (\mathtt{letrec}\ x_1 = (c_{T,i}\ \overrightarrow{s_i}), \{x_i = x_{i-1}\}_{i=2}^m, \mathit{Env}\ \mathtt{in}\ x_m)$ then perform some (cpx)-transformations such that x_m is replaced by x_1, then perform a (cpcx)-transformation such that x_1 is replaced by the constructor application, and then append the transformation from the previous bullet.

In all cases the standardisation theorem shows that $BA[s']\uparrow$ and hence $BA[s]\uparrow$.

$\neg(BA[s])\Downarrow \implies \neg(s\Uparrow)$:

Let $RED \in \mathcal{DIV}(BA[s])$ where $BA[s] \xrightarrow{RED} t$ and $t\Uparrow$. Let RED' be the prefix of RED that only changes s or shifts `letrec`-environments out of s. Note, that these `letrec` will be moved to the top, hence there is a sequence $BA[s] \xrightarrow{RED'} t' \xrightarrow{RED''} t$, with $RED = RED'RED''$ and $t' \equiv BA[s']$ or $t' \equiv (\mathtt{letrec}\ \mathit{Env}\ \mathtt{in}\ BA[s')$. If the first reduction of RED'' is an (amb-l) reduction, then either

$$BA[s'] \xrightarrow{\mathbf{no},\mathsf{amb\text{-}l}} (\mathbf{I}\,\mathbf{I}) \xrightarrow{\mathbf{no},\mathsf{lbeta}} (\mathtt{letrec}\,x = \mathbf{I}\,\mathtt{in}\,x) \xrightarrow{\mathbf{no},\mathsf{cp}} (\mathtt{letrec}\,x = \mathbf{I}\,\mathtt{in}\,\mathbf{I})$$

or

$$\begin{aligned}
&(\mathtt{letrec}\ \mathit{Env}\ \mathtt{in}\ BA[s'])\\
\xrightarrow{\mathbf{no},\mathsf{amb\text{-}l}}\ &(\mathtt{letrec}\ \mathit{Env}\ \mathtt{in}\ \mathbf{I})\ \mathbf{I}\\
\xrightarrow{\mathbf{no},\mathsf{lapp}}\ &(\mathtt{letrec}\ \mathit{Env}\ \mathtt{in}\ \mathbf{I}\ \mathbf{I})\\
\xrightarrow{\mathbf{no},\mathsf{lbeta}}\ &(\mathtt{letrec}\ \mathit{Env}\ \mathtt{in}\ (\mathtt{letrec}\ x = \mathbf{I}\ \mathtt{in}\ x))\\
\xrightarrow{\mathbf{no},\mathsf{llet}}\ &(\mathtt{letrec}\ \mathit{Env}, x = \mathbf{I}\ \mathtt{in}\ x)\\
\xrightarrow{\mathbf{no},\mathsf{cp}}\ &(\mathtt{letrec}\ \mathit{Env}, x = \mathbf{I}\ \mathtt{in}\ \mathbf{I})
\end{aligned}$$

Both cases are not possible, since the reduction sequences end in a WHNF. Hence the first reduction of RED'' must be a (seq)-reduction. We have the cases: s' is a value, or s' is a variable that is bound to a value, where the binding must be in Env. Now, we construct a normal order reduction sequence RED_s as follows: Remove all (lll)-reductions from RED' that shift `letrec`s over the context BA. Now $s \xrightarrow{RED_s} t''$, with $t'' \equiv s'$ or $t'' \equiv (\texttt{letrec}\ Env\ \texttt{in}\ s')$, i.e. t'' is a WHNF, and hence $s{\downarrow}$. □

Proposition 8.27. $\leq_c^{\Downarrow} \subseteq (\leq_c^{\downarrow})^{-1}$

Proof. Let s, t be arbitrary terms with $s \leq_c^{\Downarrow} t$ hence $\forall C \in \mathcal{C} : C[s]{\Downarrow} \implies C[t]{\Downarrow}$ and thus also $\forall C \in \mathcal{C} : BA[C[s]]{\Downarrow} \implies BA[C[t]]{\Downarrow}$. Using Lemma 8.26 this is equivalent to $\forall C \in \mathcal{C} : C[s]{\Uparrow} \implies C[t]{\Uparrow}$ and also $\forall C \in \mathcal{C} : C[t]{\downarrow} \implies C[s]{\downarrow}$, hence $t \leq_c^{\downarrow} s$. □

Let $\sim_c^{\downarrow}$ be the symmetrisation of may-convergence, i.e. $s \sim_c^{\downarrow} t$ iff $s \leq_c^{\downarrow} t \wedge t \leq_c^{\downarrow} s$.

Corollary 8.28. *If $s \leq_c t$ then $s \sim_c^{\downarrow} t$.*

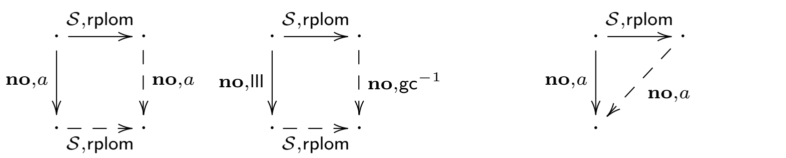
A consequence of the previous corollary is that contextual equivalence can be defined using only must-convergence.

Corollary 8.29. $s \sim_c t$ *iff* $\forall C : C[s]{\Downarrow} \iff C[t]{\Downarrow}$

The remaining two lemmas of this section show that $\leq_c$ is not an equivalence.

Lemma 8.30. *Let s be an Ω-term and t be an arbitrary term, then $s \leq_c^{\downarrow} t$.*

Proof. Let (rplom) be the transformation, that replaces an Ω-term by an arbitrary term. A complete set of forking diagrams for $(\mathcal{S}, \text{rplom})$ is:

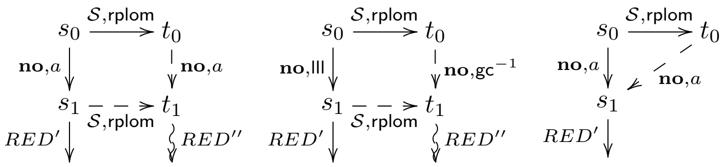


a arbitrary $a \in \{\text{case}, \text{seq}, \text{amb-l}, \text{amb-r}\}$

This follows by inspecting all cases where a normal order reduction overlaps with an (S, rplom)-transformation. Either the reduction and the transformation commute, or the environment of the Ω-term is floated out, via an (lll)-reduction, then it needs to be deleted via an (S, gc)-transformation, or the Ω-term is deleted by the normal order reduction.

Let $s_0 = S[s]$, $t_0 = S[t]$ and $s_0 \xrightarrow{S,\text{rplom}} t_0$. Further, let $s_0{\downarrow}$ and $RED_s \in \mathcal{CON}(s_0)$. We show by induction on $l = \texttt{rl}(RED_s)$, that there exists a sequence of (S, allr)-transformations that starts with t_0 and ends in a WHNF. The standardisation theorem then shows $t_0{\downarrow}$. If $l = 0$ then s_0 is a WHNF, and obviously

t_0 is also an WHNF. If $l > 0$ then we apply a forking diagram to a suffix of $\xleftarrow{RED_s} s_0 \xrightarrow{S,\mathsf{rplom}}$. With RED' being the suffix of RED_s of length $l-1$ we have the cases:



(1) (2) (3)

For cases (1) and (2) we apply the induction hypothesis to $\xleftarrow{RED'} s_1 \xrightarrow{S,\mathsf{rplom}} t_1$ and have a sequence RED'' of (S, allr)-transformations starting with t_1 that ends in a WHNF. By appending $t_0 \xrightarrow{S,\mathsf{allr}} t_1$ to RED'' we have such a sequence for t_0. Case (3) is trivial.

Finally, the context lemma for may-convergence shows that $s \leq t$. □

Lemma 8.31. *$\leq_c$ is not symmetric.*

Proof. Let $s \equiv \texttt{choice}\ \Omega\ \mathbf{I}$ and $t \equiv \mathbf{I}$. From Lemma 8.30 we have $s \leq_c^{\downarrow} t$. For all surface contexts S we can $S[s]$ transform into $S[t]$:

$$S[s] \equiv S[(\texttt{amb}\ (\lambda x.\Omega)\ (\lambda x.\mathbf{I}))\ \texttt{True}] \xrightarrow{S,\mathsf{amb}} S[(\lambda x.\mathbf{I})\ \texttt{True}]$$
$$\xrightarrow{S,\mathsf{lbeta}} S[(\texttt{letrec}\ x = \texttt{True}\ \texttt{in}\ \mathbf{I})] \xrightarrow{S,\mathsf{gc}} S[\mathbf{I}] \equiv S[t]$$

From the Standardisation Theorem follows that for all surface contexts $S[t]\uparrow \implies S[s]\uparrow$. Hence, using Corollary 4.12 we have $s \leq_c t$. Obviously $s\Uparrow$ and $t\Downarrow$. Thus, the empty context shows that $t \not\leq_c^{\Downarrow} s$. □

9 Conclusion and Further Research

We presented a call-by-need lambda-calculus with a non-deterministic `amb`-operator together with a small-step reduction semantics where the used equational theory takes fairness into account. Moreover, the Standardisation Theorem shows that our normal order reduction is a standardising reduction strategy. We have shown that all deterministic rules and additional program transformations keep contextual equivalence, where the combination of a context lemma together with complete sets of commuting and forking diagrams turned out to be successful.

With the developed proof tools, we may attempt to prove correctness of further program transformations, e.g. a rule for inlining expressions, that are used only once or that are deterministic (i.e. do not contain `amb`-expressions, also satisfying some other conditions). Also, an analysis of non-terminating terms and subterms should be subject to further investigations. By proving correctness of program transformations used in Haskell [Pey03] compilers and switching off incorrect transformations we could derive a correct compiler for Haskell extended with `amb`.

References

AFM+95. Zena Ariola, Matthias Felleisen, John Maraist, Martin Odersky, and Philip Wadler. A call-by-need lambda calculus. In *Proc. POPL '95, 22'nd Annual Symposium on Principles of Programming Languages, San Francisco, California.* ACM Press, January 1995.

Bar84. H.P. Barendregt. *The Lambda Calculus. Its Syntax and Semantics.* North-Holland, Amsterdam, New York, 1984.

BPLT02. A. Du Bois, R. Pointon, H.-W. Loidl, and P. Trinder. Implementing declarative parallel bottom-avoiding choice. In *SBAC-PAD '02: Proceedings of the 14th Symposium on Computer Architecture and High Performance Computing (SCAB-PAD'02)*, pages 82–89, Washington, DC, USA, 2002. IEEE Computer Society.

CHS05. Arnaud Carayol, Daniel Hirschkoff, and Davide Sangiorgi. On the representation of McCarthy's amb in the pi-calculus. *Theor. Comput. Sci.*, 330(3):439–473, 2005.

Gor95. Andrew Gordon. A tutorial on co-induction and functional programming. In *Functional Programming, Glasgow 1994*, pages 78–95. Springer Workshops in Computing, 1995.

HC95. Thomas Hallgren and Magnus Carlsson. Programming with fudgets. In *Advanced Functional Programming, First International Spring School on Advanced Functional Programming Techniques-Tutorial Text*, pages 137–182, London, UK, 1995. Springer-Verlag.

Hen80. Peter Henderson. *Functional Programming – Application and Implementation.* Series in Computer Science. Prentice Hall International, 1980.

Hen82. Peter Henderson. Purely Functional Operating Systems. In J. Darlington, P. Henderson, and D. A. Turner, editors, *Functional Programming and its Applications*, pages 177–192. Cambridge University Press, 1982.

HM95. John Hughes and Andrew Moran. Making choices lazily. In *FPCA '95: Proceedings of the seventh international conference on Functional programming languages and computer architecture*, pages 108–119, New York, NY, USA, 1995. ACM Press.

JH93. Mark P. Jones and Paul Hudak. Implicit and explicit parallel programming in Haskell. Technical Report CT 06520-2158, Department of Computer Science, Yale University, August 1993.

KSS98. Arne Kutzner and Manfred Schmidt-Schauß. A nondeterministic call-by-need lambda calculus. In *International Conference on Functional Programming 1998*, pages 324–335. ACM Press, 1998.

Kut00. Arne Kutzner. *Ein nichtdeterministischer call-by-need Lambda-Kalkül mit erratic choice: Operationale Semantik, Programmtransformationen und Anwendungen.* Dissertation, J.W.Goethe-Universität Frankfurt, 2000. in german.

Las98. Søren Bøgh Lassen. *Relational Reasoning about Functions and Nondeterminism.* PhD thesis, Department of Computer Science, University of Aarhus, 1998. BRICS Dissertation Series DS-98-2.

Las05. Søren Lassen. Normal Form Simulation for McCarthy's amb. In *21st Annual Conference on Mathematical Foundations of Programming Semantics, MFPS XXI*, 2005. preliminary version.

LLP05. Søren B. Lassen, Paul Blain Levy, and Prakash Panangaden. Divergence-least semantics of amb is Hoare, September 2005. Short presentation at the APPSEM II workshop, Frauenchiemsee, Germany. Available at `http://www.cs.bham.ac.uk/~pbl/papers/`.

LM99. Søren B. Lassen and Andrew Moran. Unique fixed point induction for McCarthy's amb. In *MFCS '99: Proceedings of the 24th International Symposium on Mathematical Foundations of Computer Science*, pages 198–208, London, UK, 1999. Springer-Verlag.

Man05. Matthias Mann. *A Non-Deterministic Call-by-Need Lambda Calculus: Proving Similarity a Precongruence by an Extension of Howe's Method to Sharing.* Dissertation, Johann Wolgang Goethe-Universität, Frankfurt, 2005.

McC63. John McCarthy. A Basis for a Mathematical Theory of Computation. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, pages 33–70. North-Holland, Amsterdam, 1963.

Mor98. A. K. Moran. *Call-by-name, Call-by-need, and McCarthy's Amb.* PhD thesis, Department of Computing Science, Chalmers University of Technology and University of Gothenburg, Gothenburg, Sweden, September 1998.

MS99. Andrew Moran and David Sands. Improvement in a lazy context: an operational theory for call-by-need. In *POPL '99: Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 43–56, New York, NY, USA, 1999. ACM Press.

MSS06. Matthias Mann and Manfred Schmidt-Schauß. How to prove similarity a precongruence in non-deterministic call-by-need lambda calculi. Frank report 22, Institut für Informatik. J.W.Goethe-Universität Frankfurt, January 2006.

NC95. V. Natarajan and Rance Cleaveland. Divergence and fair testing. In Zoltán Fülöp and Ferenc Gécseg, editors, *ICALP*, volume 944 of *Lecture Notes in Comput. Sci*, pages 648–659. Springer, 1995.

Pey03. Simon Peyton Jones, editor. *Haskell 98 language and libraries: the Revised Report.* Cambridge University Press, 2003. `www.haskell.org`.

PM02. Simon Peyton Jones and Simon Marlow. Secrets of the Glasgow Haskell Compiler inliner. *J. Funct. Programming*, 12(4+5):393–434, July 2002.

Sab03. David Sabel. Realising nondeterministic I/O in the Glasgow Haskell Compiler. Frank report 17, Institut für Informatik, J.W. Goethe-Universität Frankfurt am Main, December 2003.

San95. André Santos. *Compilation by Transformation in Non-Strict Functional Languages.* PhD thesis, Glasgow University, Department of Computing Science, 1995.

SS92. H. Søndergaard and P. Sestoft. Non-determinism in functional languages. *Comput. J.*, 35(5):514–523, 1992.

SS03. Manfred Schmidt-Schauß. FUNDIO: A Lambda-Calculus with a `letrec`, `case`, Constructors, and an IO-Interface: Approaching a Theory of `unsafePerformIO`. Frank report 16, Institut für Informatik, J.W. Goethe-Universität Frankfurt, September 2003.

SSSS04. Manfred Schmidt-Schauß, Marko Schütz, and David Sabel. On the safety of Nöcker's strictness analysis. Frank Report 19, Institut für Informatik, J.W. Goethe-Universität Frankfurt, October 2004.

SSSS05. Manfred Schmidt-Schauß, Marko Schütz, and David Sabel. A complete proof of the safety of Nöcker's strictness analysis. Frank report 20, Institut für Informatik. J.W.Goethe-Universität Frankfurt, April 2005.

THLP98. Philip W. Trinder, Kevin Hammond, Hans-Wolfgang Loidl, and Simon L. Peyton Jones. Algorithm + Strategy = Parallelism. *J. Funct. Programming*, 8(1):23–60, January 1998.