

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

David Sabel and Manfred Schmidt-Schauß

Research group for Artificial Intelligence and Software Technology
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
Revised Version¹

February 19, 2008

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** that is locally bottom-avoiding. We use a small-step operational semantics in form of a single-step rewriting system that defines a (non-deterministic) normal order reduction. This strategy can be made fair by adding resources for bookkeeping. As equational theory we use contextual equivalence, i.e. terms are equal if plugged into any program context their termination behaviour is the same, where we use a combination of may- as well as must-convergence, which is appropriate for non-deterministic computations. We show that we can drop the fairness condition for equational reasoning, since the valid equations w.r.t. normal order reduction are the same as for fair normal order reduction. We evolve different proof tools for proving correctness of program transformations, in particular, a context lemma for may- as well as must-convergence is proved, 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 preserve contextual equivalence. We also prove a standardisation theorem for fair normal order reduction. The structure of the ordering \leq_c is also analysed: Ω is not a least element, and \leq_c already implies contextual equivalence w.r.t. may-convergence.

¹ This document is a revised version of an earlier version originally published on the web in July 2006.

Table of Contents

A Call-by-Need Lambda-Calculus with Locally Bottom-Avoiding Choice: Context Lemma and Correctness of Transformations	1
<i>David Sabel and Manfred Schmidt-Schauß</i>	
1 Introduction	3
1.1 Motivation	3
1.2 Related Work	6
1.3 Overview	7
2 The Nondeterministic Call-by-Need Calculus λ_{amb}^{let}	8
2.1 The Syntax of the Language	8
2.2 Reduction Rules	10
2.3 Normal Order Reduction	13
2.4 Encoding of Non-deterministic and Parallel Operators	16
2.5 Convergence and Divergence	17
2.5.1 An Alternative Definition of Divergence	19
3 Fair Normal Order Reduction	20
4 Contextual Equivalence and Proof Techniques	27
4.1 Preorders for May- and Must-Convergence	27
4.2 Context Lemmas	29
4.3 Properties of the (III)-Reduction	33
4.4 Complete Sets of Commuting and Forking Diagrams	34
5 Correctness of (lbeta), (case-c), (seq-c) and the CD-Properties	36
5.1 Correctness of (lbeta), (case-c), (seq-c)	36
5.2 Properties for Correctness of Program Transformations	39
6 Additional Correct Program Transformations	40
6.1 Diagrams for (gc)	42
6.2 Diagrams for (cpx)	42
6.3 Diagrams for (xch)	43
6.4 Diagrams for (abs)	43
6.5 Diagrams for (cpcx)	44
6.6 Correctness of (opt)	44
7 Correctness of Deterministic Reduction Rules	48
7.1 Correctness of (case)	48
7.2 Correctness of (III)	48
7.2.1 Diagrams for (lapp), (lcase) and (lseq)	48
7.2.2 Diagrams for (lamb)	49
7.2.3 Diagrams for (llet)	50
7.2.4 Proving Correctness of (III)	51
7.3 Correctness of (seq)	55
7.4 Correctness of (cp)	58
8 The Standardisation Theorem and an Application	61
8.1 Properties of the Reduction (amb)	61

8.2	The Standardisation Theorem	64
8.3	On the Equivalence of Ω -terms	65
	8.3.1 Properties of B -Labelled Expressions	67
	8.3.2 Equivalence of Ω -terms	68
8.4	Proving a Bottom-Avoidance Law	68
8.5	On the Relation Between \leq_c^\downarrow and $\leq_c^{\downarrow\downarrow}$	72
9	Conclusion and Further Research	75

1 Introduction

1.1 Motivation

Higher-order lambda calculi with non-deterministic operators have been investigated in several works. In particular a non-deterministic choice operator that chooses one of its arguments as result, but converges if one of its 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], it permits to implement a `merge` operator for streams in event-driven systems like graphical user interfaces e.g. [HC95] or functional operating systems [Hen82].

Such a non-deterministic 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. The operator *amb* is bottom-avoiding: let \perp be an expression that cannot converge and s be a value. Then the expressions $(amb\ s\ \perp)$ and $(amb\ \perp\ s)$ both evaluate to s . The bottom-avoidance of *amb* is only *local*, because its evaluation is independent of the surrounding context, e.g. the expression²

$$(\text{if } (amb\ \text{True}\ \text{False}) \text{ then True else } \perp)$$

is may-divergent.

There are many investigations in the properties of higher-order calculi with `amb` using call-by-value, call-by-name, or call-by-need evaluation strategies. The strategies can be distinguished by their internal treatment of function arguments that contain `amb`-expressions. Call-by-name tends to copy arguments, even if an argument is e.g. of the form $(amb\ s\ t)$ and thus implements plural non-determinism [SS92], whereas call-by-value and call-by-need are more restrictive in copying arguments: only values, in particular abstractions, are allowed to be copied. Thus these evaluation strategies implement singular non-determinism

Since we will investigate a rather expressive calculus with several constructs and many reductions, which is close to a real-world calculus, we will provide some justification for that. A locally bottom-avoiding choice in combination with constructs for explicit sharing and for sequential evaluation enables to define many other non-deterministic operators within the language, e.g. erratic-choice, locally demonic choice (see [SS92] for an overview of different non-deterministic

² `if b then s else t` can be encoded in our calculus as `caseBool b (True → s) (False → t)`

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. A further reason for the expressiveness is that we want the results to be immediately applicable to an implementation of `amb` in a variant of Haskell [Sab03a]. This enforces that the calculus has to respect sharing and must be a call-by-need calculus that implements singular non-determinism, as already argued in [Mor98]. Another reason is that using a less expressive language to prove equalities of expressions is of limited value, since there is no guarantee that equations remain valid if the language is extended, since extensions of the language may increase the expressiveness of contexts and thus invalidate equalities. Also verification of properties of programs and of compilers for a higher-order calculus with `amb` require an exact semantics as solid foundation, and also methods to prove non-trivial properties.

For all these reasons we investigate a higher-order lambda-calculus with an operator `amb`, (weakly) typed `case`, constructors, `letrec` and `seq`. The `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 as rewriting system on expressions together with a strategy. An unwinding mechanism determines all permitted subterms for the next reduction, called normal order redexes. A normal order reduction sequence is one that only reduces normal order redexes. For `amb`-free expressions this will be deterministic, since normal order redexes are unique, whereas for expressions containing occurrences of `amb` there may be several normal order redexes. A single normal order reduction sequence is a reduction that in every step non-deterministically chooses one of the concurrently possible subexpressions for reduction. The set of all normal order reduction sequences also comprises reductions such that `amb` is not locally bottom-avoiding. We add resources in Definition 3.1 to normal order reductions to single out the fair normal order reductions that all have the property that `amb` is locally bottom-avoiding.

Based on all normal order reductions 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 taking *may-convergence* into account is too weak for calculi with an `amb`-operator (e.g. see [Mor98]). Hence our equivalence will test for *must-convergence*, too. Summarising, contextual equivalence \sim_c is the symmetrisation of the contextual preorder \leq_c , where

$$s \leq_c t \text{ iff } (\forall C : C[s] \downarrow \implies C[t] \downarrow \text{ and } \forall C : C[s] \not\downarrow \implies C[t] \not\downarrow)$$

with C denoting contexts and \downarrow and $\not\downarrow$ being the predicates for may- and must-convergence. Note that our predicate for must-convergence is the converse of may-divergence.

In Theorem 3.20 we show that the notions of may- as well as must-convergence, respectively, are the same for normal order reductions and for fair

normal order reductions, which implies that the corresponding notions of contextual equivalences are identical. This will be a great simplification in the following, since normal order reduction is sufficient for all argumentations about equivalence and correctness of program transformations. For a call-by-name calculus with `amb` the same coincidence was shown in [CHS05].

In contrast to [HM95, Mor98], we only treat divergences that are called *strong* in [CHS05] based on distinguishing between strong and weak divergence, introduced by [NC95]. A consequence is that a term that has an infinite reduction but never loses the ability to converge is not divergent.

Proving contextual equivalence directly seems to be very hard, since all program contexts have to be taken into account. Other methods like 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`, since their approach requires to eliminate let-bindings, and the elimination method does not work for recursive `lets` (cf. [Man05, Section 6.2]).

In this paper 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 that the deterministic reductions of the calculus are correct program transformations (Theorem 8.1), i.e. their application preserves contextual equivalence. This is of importance in a compiler that uses reductions as optimisations, in particular, if partial evaluation is used. We will also show the correctness of some other program transformations that are used for optimisation in compilers of functional programming languages (e.g. [San95]).

An important result is the Standardisation Theorem 8.14 which states that if there exists a sequence of transformations or reductions to a weak head normal form then there is also a fair evaluation in normal order to a weak head normal form (see Corollary 8.15). The second part of the Standardisation Theorem states that if there exists a sequence of transformations inside surface contexts (i.e. not within abstractions) resulting in a term that cannot converge, then fair normal order reduction can also reduce to such a term. A consequence of the Standardisation Theorem is that must-convergence is preserved while applying program transformations inside surface contexts (see Proposition 8.16).

Using the Standardisation Theorem we first show that all (closed) must-divergent terms are in the same equivalence class w.r.t. \sim_c . We then show a classical bottom-avoidance law of our `amb`-operator (Proposition 8.33), i.e. $\mathbf{amb} \ \Omega \ t \sim_c \ t$ and $\mathbf{amb} \ t \ \Omega \sim_c \ t$, where Ω is a term that cannot converge. As final result we show that contextual equivalence can be defined by only taking must-convergence into account (Corollary 8.39), i.e. $s \sim_c t$ iff $\forall C : C[s] \Downarrow \Leftrightarrow C[t] \Downarrow$.

Our results show that it is possible to overcome the difficulties that Moran encountered in his thesis [Mor98]: We were able to prove the full context lemma, and also to show correctness of several transformation rules for the full \sim_c -equivalence, where Moran has to confine himself mostly to may-convergence.

To our knowledge this is the first paper that proves correctness of program transformations for a call-by-need calculus with **amb** including may- and must-convergence in the definition of correctness.

An implementation of evaluation can be done by implementing normal order reduction and in addition taking care that only fair reductions are possible. Note that the unrestricted definition of normal order reduction permits unfair reduction sequences: i.e., evaluation of an expression (**amb** \perp **True**) may reduce \perp infinitely often while ignoring that the second argument of **amb** is already a value. Fair reduction strategies can be achieved by using Moran’s [Mor98] approach to implement fair evaluation by annotating **amb**-expressions with resources for both of its arguments, and decreasing the resource of an argument for every reduction step that is related to this argument. The (fair) scheduler increases the resources only if both resources are 0 and the increase is greater than 0 for both arguments.

[CHS05] define a fair operational semantics for their call-by-name calculus which is free of resource annotations. We considered to adopt this method for our investigation, but unfortunately it does not properly work for a calculus with shared bindings (see Section 3).

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, however, there are some small differences: He uses strict **let** expressions, where we use lazy **let** and a **seq**-operator for implementing sequential evaluation. We use (weakly) typed **case**-expressions, whereas [Mor98] uses an untyped **case**. Moran also uses contextual equivalence, but our equational theory differs from his, since his predicate for must-convergence captures weak divergences, and thus is not the same as our predicate (see Example 4.4).

Moran was not able to show correctness of program transformations w.r.t. to contextual equivalence that takes may- and must-convergence into account. He only provides a context lemma (based on improvement theory [MS99]) for the may-convergence part, but failed to prove a context lemma for must-convergence. As Moran already mentioned, proofs of correctness only w.r.t. may-convergence do not distinguish an **amb**-operator from erratic choice.

The technical advantage of our approach over Moran’s is that we do not need an explicit heap, and that for large parts of the reasoning we can ignore the resource annotations of **amb**. This may be the reason that we were able to prove a context lemma for may- as well as must-convergence and also the correctness of several program transformations.

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. [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.

Diagram-based techniques for proving program equivalences have already been used in [MT00] for proving meaning preservation in a call-by-value calculus with recursive definitions. [WPK03] provide an abstract framework for proving meaning-preservation using diagrams. Unfortunately the axioms of this framework are not fulfilled for several reasons in our calculus: One reason is that reduction in our calculus is non-deterministic, hence evaluation does not preserve meaning in the sense of [WPK03].

Work on call-by-value calculi extended with bottom-avoiding choice has been done in [Las98]. [FK03] investigated interaction nets extended with an *amb*-operator.

1.3 Overview

In section 2 we introduce the calculus A_{amb}^{let} , 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 rearrange **letrec**-environments and finally we introduce the notion of complete sets of commuting and forking diagrams.

In section 5 we prove correctness of those reduction rules where correctness follows easily and define general properties of a program transformation and show that their validity ensures correctness of the transformation. Equipped with this proof tool in sections 6 and 7 we prove the correctness of all defined deterministic reduction rules and of some additional program transformations. In section 8 we prove the Standardisation Theorem, the equivalence of must-divergent terms and the bottom-avoidance law. We conclude the section by proving some remarkable properties of the used contextual preorder. In particular we prove that the contextual preorder implies equivalence w.r.t. may-convergence. In the last section we conclude and give some directions for further research.

2 The Nondeterministic Call-by-Need Calculus $\Lambda_{\text{amb}}^{\text{let}}$

In this section we first introduce the syntax of the language of $\Lambda_{\text{amb}}^{\text{let}}$, 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 $\Lambda_{\text{amb}}^{\text{let}}$ is very similar to the abstract language used in [SSSS04] with the difference that $\Lambda_{\text{amb}}^{\text{let}}$ 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] $\Lambda_{\text{amb}}^{\text{let}}$ 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 $\Lambda_{\text{amb}}^{\text{let}}$ 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 $\text{ar}(c_{T,i}) \geq 0$.

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

$E ::= V$	<i>(variable)</i>
$(c_{T,i} E_1 \dots E_{\text{ar}(c_{T,i})})$	<i>(constructor application)</i>
$(\text{seq } E_1 E_2)$	<i>(seq-expression)</i>
$(\text{case}_T E Alt_1 \dots Alt_{ T })$	<i>(case-expression)</i>
$(E_1 E_2)$	<i>(application)</i>
$(\text{amb } E_1 E_2)$	<i>(amb-expression)</i>
$(\lambda V.E)$	<i>(abstraction)</i>
$(\text{letrec } V_1 = E_1, \dots, V_n = E_n \text{ in } E)$	<i>(letrec-expression)</i>
where $n \geq 1$	
$Alt ::= (Pat \rightarrow E)$	<i>(case-alternative)</i>
$Pat ::= (c_{T,i} V_1 \dots V_{\text{ar}(c_{T,i})})$	<i>(pattern)</i>

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 $V_1 \dots V_{\text{ar}(c_{T,i})}$ are pairwise disjoint.
- In a 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 $(\mathbf{letrec} \ x_1 = s_1, \dots, x_n = s_n \ \mathbf{in} \ t)$ 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:

- $(\mathbf{case}_T \ E \ \mathit{alts})$ instead of $(\mathbf{case}_T \ E \ \mathit{Alt}_1 \ \dots \ \mathit{Alt}_{|T|})$,
- $(\mathbf{letrec} \ \mathit{Env} \ \mathbf{in} \ E)$ instead of $(\mathbf{letrec} \ x_1 = E_1, \dots, x_n = E_n \ \mathbf{in} \ E)$. This will also be used freely for parts of the bindings.
- $(c_i \ \overrightarrow{s_i})$ instead of $(c_i \ s_1 \ \dots \ s_{\mathit{ar}(c_i)})$
- $\{x_{f(i)} = s_{g(i)}\}_{i=j}^n$ 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 $f, g : \mathbb{N}_0 \rightarrow \mathbb{N}_0$
- 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.

Sometimes we will use tree addresses (*positions*) as strings of positive integers in expressions with their standard meaning. Then the *depth* of a subterms t' of t at position p is the length of the string p . With $t|_p$ we denote the subexpression of t at position p .

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

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

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

$$\begin{aligned} \mathcal{C} ::= & [\cdot] \mid (\mathcal{C} \ E) \mid (E \ \mathcal{C}) \mid (\mathbf{seq} \ E \ \mathcal{C}) \mid (\mathbf{seq} \ \mathcal{C} \ E) \mid \lambda x. \mathcal{C} \mid (\mathbf{amb} \ \mathcal{C} \ E) \mid (\mathbf{amb} \ E \ \mathcal{C}) \\ & \mid (\mathbf{case}_T \ \mathcal{C} \ \mathit{alts}) \mid (\mathbf{case}_T \ E \ \mathit{Alt}_1 \ \dots \ (\mathit{Pat} \ \rightarrow \ \mathcal{C}) \ \dots \ \mathit{Alt}_n) \\ & \mid (c_{T,i} \ E_1 \ \dots \ E_{i-1} \ \mathcal{C} \ E_{i+1} \ \dots \ E_{\mathit{ar}(c)}) \\ & \mid (\mathbf{letrec} \ x_1 = E_1, \dots, x_n = E_n \ \mathbf{in} \ \mathcal{C}) \\ & \mid (\mathbf{letrec} \ x_1 = E_1, \dots, x_{i-1} = E_{i-1}, x_i = \mathcal{C}, x_{i+1} = E_{i+1}, \dots, x_n = E_n \ \mathbf{in} \ E) \end{aligned}$$

The main depth of a context \mathcal{C} is the depth of the hole of the context \mathcal{C} , i.e. the length of the position of the hole. With $\mathcal{C}_{\#i}$ we denote a context of main depth i . Let t be a term, \mathcal{C} be a context, then $\mathcal{C}[t]$ is the result of replacing the hole of \mathcal{C} with term t .

Two contexts C_1, C_2 are called disjoint iff the positions p_1, p_2 of their respective holes are not prefixes of each other, i.e. neither p_1 is a prefix of p_2 nor p_2 is a prefix of p_1 .

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

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

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 $(\text{letrec } x_1 = E_1, \dots, x_n = E_n \text{ in } R^-)$ where R^- is a maximal weak reduction context and $t_0 \not\equiv x_j$ for all $j = 1, \dots, n$, or
- of the form $(\text{letrec } x_1 = R_1^-, x_2 = R_2^-[x_1], \dots, x_j = R_j^-[x_{j-1}], \dots \text{ in } R^-[x_j])$, where $R_i^-, i = 1, \dots, j$ 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 $(\text{letrec } x_2 = \lambda x.x, x_1 = x_2 \ x_1, x_3 = (\text{amb } (x_2 \ x_1) \ y) \text{ in } (\text{amb } x_1 \ x_3))$ there exist the following maximal reduction contexts:

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

The first maximal reduction context can be calculated in 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 deterministic reduction rules of $\Lambda_{\text{amb}}^{\text{let}}$ are defined in Fig. 1. The non-deterministic reduction rules for evaluating **amb**-expressions are defined in 2. We define the following unions of some reductions:*

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

Reductions are denoted using an arrow with superscripts: e.g. $\xrightarrow{\text{llet}}$. 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 $*$, e.g. $\xrightarrow{\text{III},*}$ is reflexive transitive closure of $\xrightarrow{\text{III}}$. We use uppercase words to denote (finite) sequences of reductions, e.g. for the sequence $RED = \xrightarrow{\text{case}} \xrightarrow{\text{III}} \xrightarrow{\text{lbeta}}$, we write \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 (**III**) arranges **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 explicitly follow the bindings during the reduction, instead of removing indirections. The reason for keeping indirections is technical: the reduction diagrams are easier to close and it seems easier to find measures for the induction in the correctness proofs. Nevertheless we show that the program transformations (**abs**), (**cpx**), (**cpcx**) and (**gc**) are correct (see section 6) and thus copying of variables and constructor applications are allowed optimisations. Another reason for having more rules than [AFM⁺95] is that our

(lbeta)	$((\lambda x.s) r) \rightarrow (\text{letrec } x = r \text{ in } s)$
(cp-in)	$(\text{letrec } x_1 = (\lambda x.s), \{x_i = x_{i-1}\}_{i=2}^m, Env \text{ in } C[x_m])$ $\rightarrow (\text{letrec } x_1 = (\lambda x.s), \{x_i = x_{i-1}\}_{i=2}^m, Env \text{ in } C[(\lambda x.s)])$
(cp-e)	$(\text{letrec } x_1 = (\lambda x.s), \{x_i = x_{i-1}\}_{i=2}^m, Env, y = C[x_m] \text{ in } r)$ $\rightarrow (\text{letrec } x_1 = (\lambda x.s), \{x_i = x_{i-1}\}_{i=2}^m, Env, y = C[(\lambda x.s)] \text{ in } r)$
(llet-in)	$(\text{letrec } Env_1 \text{ in } (\text{letrec } Env_2 \text{ in } r)) \rightarrow (\text{letrec } Env_1, Env_2 \text{ in } r)$
(llet-e)	$(\text{letrec } x_1 = s_1, \dots, x_i = (\text{letrec } Env_2 \text{ in } s_i), \dots, x_n = s_n \text{ in } r)$ $\rightarrow (\text{letrec } x_1 = s_1, \dots, x_i = s_i, \dots, x_n = s_n, Env_2 \text{ in } r)$
(lapp)	$((\text{letrec } Env \text{ in } t) s) \rightarrow (\text{letrec } Env \text{ in } (t s))$
(lcase)	$(\text{case}_T (\text{letrec } Env \text{ in } t) alts) \rightarrow (\text{letrec } Env \text{ in } (\text{case}_T t alts))$
(lseq)	$(\text{seq } (\text{letrec } Env \text{ in } s) t) \rightarrow (\text{letrec } Env \text{ in } (\text{seq } s t))$
(lamb-l)	$(\text{amb } (\text{letrec } Env \text{ in } s) t) \rightarrow (\text{letrec } Env \text{ in } (\text{amb } s t))$
(lamb-r)	$(\text{amb } s (\text{letrec } Env \text{ in } t)) \rightarrow (\text{letrec } Env \text{ in } (\text{amb } s t))$
(seq-c)	$(\text{seq } v t) \rightarrow t$, if v is a value
(seq-in)	$(\text{letrec } x_1 = v, \{x_i = x_{i-1}\}_{i=2}^m, Env \text{ in } C[(\text{seq } x_m t)])$ $\rightarrow (\text{letrec } x_1 = v, \{x_i = x_{i-1}\}_{i=2}^m, Env \text{ in } C[t])$, if v is a value
(seq-e)	$(\text{letrec } x_1 = v, \{x_i = x_{i-1}\}_{i=2}^m, Env, y = C[(\text{seq } x_m t)] \text{ in } r)$ $\rightarrow (\text{letrec } x_1 = v, \{x_i = x_{i-1}\}_{i=2}^m, Env, y = C[t] \text{ in } r)$, if v is a value
(case-c)	for the case $\text{ar}(c_{T,i}) = n \geq 1$: Let y_i be fresh variables, then $(\text{case}_T (c_{T,i} \vec{t}_i) \dots ((c_{T,i} \vec{y}_i) \rightarrow t) \dots) \rightarrow (\text{letrec } \{y_i = t_i\}_{i=1}^n \text{ in } t)$
(case-c)	for the case $\text{ar}(c_{T,i}) = 0$: $(\text{case}_T c_{T,i} \dots (c_{T,i} \rightarrow t) \dots) \rightarrow t$
(case-in)	for the case $\text{ar}(c_{T,i}) = n \geq 1$: Let y_i be fresh variables, then $\text{letrec } x_1 = (c_{T,i} \vec{t}_i), \{x_i = x_{i-1}\}_{i=2}^m, Env$ $\text{in } C[\text{case}_T x_m \dots (c_{T,i} \vec{z}_i \rightarrow t) \dots]$ $\rightarrow \text{letrec } x_1 = (c_{T,i} \vec{y}_i), \{y_i = t_i\}_{i=1}^n, \{x_i = x_{i-1}\}_{i=2}^m, Env$ $\text{in } C[(\text{letrec } \{z_i = y_i\}_{i=1}^n \text{ in } t)]$
(case-in)	for the case $\text{ar}(c_{T,i}) = 0$: $\text{letrec } x_1 = c_{T,i}, \{x_i = x_{i-1}\}_{i=2}^m, Env \text{ in } C[\text{case}_T x_m \dots (c_{T,i} \rightarrow t) \dots]$ $\rightarrow \text{letrec } x_1 = c_{T,i}, \{x_i = x_{i-1}\}_{i=2}^m, Env \text{ in } C[t]$
(case-e)	for the case $\text{ar}(c_{T,i}) = n \geq 1$: Let y_i be fresh variables, then $\text{letrec } x_1 = (c_{T,i} \vec{t}_i), \{x_i = x_{i-1}\}_{i=2}^m, Env,$ $u = C[\text{case}_T x_m \dots (c_{T,i} \vec{z}_i \rightarrow r_1) \dots]$ $\text{in } r_2$ $\rightarrow \text{letrec } x_1 = (c_{T,i} \vec{y}_i), \{y_i = t_i\}_{i=1}^n, \{x_i = x_{i-1}\}_{i=2}^m, Env,$ $u = C[(\text{letrec } \{z_i = y_i\}_{i=1}^n \text{ in } r_1)]$ $\text{in } r_2$
(case-e)	for the case $\text{ar}(c_{T,i}) = 0$: $\text{letrec } x_1 = c_{T,i}, \{x_i = x_{i-1}\}_{i=2}^m, Env,$ $u = C[\text{case}_T x_m \dots (c_i \rightarrow r_1) \dots]$ $\text{in } r_2$ $\rightarrow \text{letrec } x_1 = c_{T,i}, \{x_i = x_{i-1}\}_{i=2}^m, Env, u = C[r_1] \text{ in } r_2$

Fig. 1. Deterministic Reduction Rules of the Calculus

$(\mathbf{amb-l-c}) \quad (\mathbf{amb} \ v \ s) \rightarrow v, \text{ if } v \text{ is a value}$
$(\mathbf{amb-r-c}) \quad (\mathbf{amb} \ s \ v) \rightarrow v, \text{ if } v \text{ is a value}$
$(\mathbf{amb-l-in}) \quad (\mathbf{letrec} \ x_1 = v, \{x_i = x_{i-1}\}_{i=2}^m, Env \ \mathbf{in} \ C[(\mathbf{amb} \ x_m \ s)])$ $\rightarrow (\mathbf{letrec} \ x_1 = v, \{x_i = x_{i-1}\}_{i=2}^m, Env \ \mathbf{in} \ C[x_m]),$ if v is a value
$(\mathbf{amb-r-in}) \quad (\mathbf{letrec} \ x_1 = v, \{x_i = x_{i-1}\}_{i=2}^m, Env \ \mathbf{in} \ C[(\mathbf{amb} \ s \ x_m)])$ $\rightarrow (\mathbf{letrec} \ x_1 = v, \{x_i = x_{i-1}\}_{i=2}^m, Env \ \mathbf{in} \ C[x_m]),$ if v is a value
$(\mathbf{amb-l-e}) \quad (\mathbf{letrec} \ x_1 = v, \{x_i = x_{i-1}\}_{i=2}^m, Env, y = C[(\mathbf{amb} \ x_m \ t)] \ \mathbf{in} \ r)$ $\rightarrow (\mathbf{letrec} \ x_1 = v, \{x_i = x_{i-1}\}_{i=2}^m, Env, y = C[x_m] \ \mathbf{in} \ r),$ if v is a value
$(\mathbf{amb-r-e}) \quad (\mathbf{letrec} \ x_1 = v, \{x_i = x_{i-1}\}_{i=2}^m, Env, y = C[(\mathbf{amb} \ t \ x_m)] \ \mathbf{in} \ r)$ $\rightarrow (\mathbf{letrec} \ x_1 = v, \{x_i = x_{i-1}\}_{i=2}^m, Env, y = C[x_m] \ \mathbf{in} \ r),$ if v is a value

Fig. 2. Non-Deterministic Reduction Rules of the Calculus

syntax has **case**-, **seq**- and **amb**-expressions, which are not present for the call-by-need calculus of [AFM⁺95]. The special 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 . To establishing understanding we start with describing how a position of a normal order redex can be reached by using a nondeterministic unwinding algorithm \mathcal{UN} . After that we will define the normal order reduction.

Let s be a term. If $s \equiv (\mathbf{letrec} \ Env \ \mathbf{in} \ s')$ apply **uw** to the pair $(s', (\mathbf{letrec} \ Env \ \mathbf{in} \ [\cdot]))$, otherwise apply **uw** to the pair $(s, [\cdot])$. For detecting loops the unwinding algorithm marks the bindings of a **letrec**-expression when visiting them. We label the symbol = with a dot, i.e. a binding $x \overset{\bullet}{=} s$ is marked as ‘visited’ and the algorithm stops if it hits a visited binding. We assume that

these markers are removed before the result of the algorithm is returned.

$$\begin{aligned}
\text{uw}((s \ t), R) &\rightarrow \text{uw}(s, R[(\cdot) \ t]) \\
\text{uw}((\text{seq } s \ t), R) &\rightarrow \text{uw}(s, R[(\text{seq } \cdot) \ t]) \\
\text{uw}((\text{case } s \ \text{alts}), R) &\rightarrow \text{uw}(s, R[(\text{case } \cdot) \ \text{alts}]) \\
\text{uw}((\text{amb } s \ t), R) &\rightarrow \text{uw}(s, R[(\text{amb } \cdot) \ t]) \ \text{or} \ \text{uw}(t, R[(\text{amb } s \ \cdot)]) \\
\text{uw}(x, (\text{letrec } x = s, \text{Env} \ \text{in } R^-)) &\rightarrow \text{uw}(s, (\text{letrec } x \doteq [\cdot], \text{Env} \ \text{in } R^-[x])) \\
\text{uw}(x, (\text{letrec } y \doteq R^-, x = s, \text{Env} \ \text{in } t)) &\rightarrow \text{uw}(s, (\text{letrec } y \doteq R^-[x], x \doteq [\cdot], \text{Env} \ \text{in } t)) \\
\text{uw}(s, R) &\rightarrow (s, R) \ \text{if no other rule is applicable}
\end{aligned}$$

The algorithm starting with term s returns a pair (s', R) with $R[s'] \equiv s$ and either R is a maximal reduction context for s or the algorithm stops because a cycle has been detected; in this case s' is a variable that is bound in R .

Since a cycle detection is implemented, the algorithm always terminates, e.g. for the term $(\text{letrec } x = y, y = x \ \text{in } x)$ the result is the pair $(x, (\text{letrec } x = y, y = [\cdot] \ \text{in } x))$:

$$\begin{aligned}
\text{uw}(x, (\text{letrec } x = y, y = x \ \text{in } [\cdot])) &\rightarrow \text{uw}(y, (\text{letrec } x \doteq [\cdot], y = x \ \text{in } x)) \\
&\rightarrow \text{uw}(x, (\text{letrec } x \doteq y, y \doteq [\cdot] \ \text{in } x)) \rightarrow (x, (\text{letrec } x = y, y = [\cdot] \ \text{in } x))
\end{aligned}$$

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 for the result (s, R) of the unwinding algorithm no normal order reduction is possible. E.g. that happens, if the first argument of a case -expression has the wrong type, if a free variable occurs inside the maximal reduction context, or if the unwinding algorithm stops, because it has detected a loop.

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 $\xrightarrow{\text{no}}$ is defined by one of the following cases:*

If t' is a letrec -expression $(\text{letrec } \text{Env}_1 \ \text{in } t'')$, and $R \not\equiv [\cdot]$, then there are the following cases, where R_0 is a reduction context:

1. $R \equiv R_0[(\text{seq } \cdot) \ r]$. Reduce $(\text{seq } t' \ r)$ using rule (lseq) .

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

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

8. $R \equiv R_0[\text{case}_T [\cdot] \dots]$, $t' \equiv (c_T \dots)$, i.e. the top constructor of t' belongs to type T . Then apply (case-c) to $(\text{case}_T t' \dots)$. Note that this covers two cases of the reduction (case-c).
9. $R \equiv \text{letrec } x_1 = [\cdot], \{x_i = x_{i-1}\}_{i=2}^m, Env \text{ in } R_0^-[\text{case}_T x_m (c_{T,j} \vec{y}_i \rightarrow r) \text{alts}]$ and $t' \equiv (c_{T,j} \vec{t}_i)$. Then apply (case-in) resulting in $\text{letrec } x_1 = (c_{T,j} \vec{z}_i), \{x_i = x_{i-1}\}_{i=2}^m, \{z_i = t_i\}_{i=1}^n, Env \text{ in } R_0^-[(\text{letrec } \{y_i = z_i\}_{i=1}^n \text{ in } r)]$
10. $R \equiv \text{letrec } x_1 = [\cdot], \{x_i = x_{i-1}\}_{i=2}^m, Env \text{ in } R_0^-[\text{case}_T x_m (c_{T,j} \rightarrow r) \text{alts}]$ and $t' \equiv c_{T,j}$. Apply (case-in) resulting in $\text{letrec } x_1 = c_{T,j}, \{x_i = x_{i-1}\}_{i=2}^m, Env \text{ in } R_0^-[r]$.
11. $R \equiv \text{letrec } x_1 = [\cdot], \{x_i = x_{i-1}\}_{i=2}^m, Env,$
 $y = R_0^-[\text{case}_T x_m (c_{T,j} \vec{y}_i \rightarrow r) \text{alts}]$
 $\text{in } r'$,
 and $t' \equiv (c_{T,j} \vec{t}_i)$, and y is in a reduction context. Then apply (case-e) resulting in
 $\text{letrec } x_1 = (c_{T,j} \vec{z}_i), \{x_i = x_{i-1}\}_{i=2}^m, \{z_i = t_i\}_{i=1}^n, Env,$
 $y = R_0^-[(\text{letrec } \{y_i = z_i\}_{i=1}^n \text{ in } r)]$
 $\text{in } r'$
12. $R \equiv \text{letrec } x_1 = [\cdot], \{x_i = x_{i-1}\}_{i=2}^m, Env,$
 $y = R_0^-[\text{case}_T x_m (c_{T,j} \rightarrow r) \text{alts}]$
 $\text{in } r'$,
 and $t' \equiv c_{T,j}$, and y is in a reduction context. Then apply (case-e) resulting in $\text{letrec } x_1 = c_{T,j}, \{x_i = x_{i-1}\}_{i=2}^m, Env, y = R_0^-[r] \text{ in } r'$.
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. $R \equiv (\text{letrec } x_1 = [\cdot], \{x_i = x_{i-1}\}_{i=2}^m, Env \text{ in } R_0^-[x_m])$ 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 $(\text{letrec } x_1 = t', \{x_i = x_{i-1}\}_{i=2}^m, Env \text{ in } R_0^-[t'])$.
15. $R \equiv (\text{letrec } x_1 = [\cdot], \{x_i = x_{i-1}\}_{i=2}^m, Env, y = R_0^-[x_m] \text{ in } r)$ 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 $(\text{letrec } x_1 = t', \{x_i = x_{i-1}\}_{i=2}^m, Env, y = R_0^-[t'] \text{ in } r)$.
16. $R \equiv R_0[(\text{seq} [\cdot] r)]$. Then apply (seq-c) to $(\text{seq } t' r)$ resulting in r .

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

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).

Note that there are 24 cases for normal order reduction and 25 reduction rules, however, every rule occurs exactly once. The reason for the difference of 1 is that two (case-c)-reductions are one case in the definition of normal order reduction.

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 $\mathbf{rl}(RED)$. With $\mathbf{rl}_{(\wedge_a)}(RED)$ we denote the number of non- a reductions in RED where a is a specific reduction.

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

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 argument, but has the value of its second argument (Glasgow parallel Haskell

<code>par</code>	$\equiv \lambda x.\lambda y.\mathbf{amb}(\mathbf{seq} x y)(\mathbf{seq} y y)$
<code>spar</code>	$\equiv \lambda x.\lambda y.\mathbf{amb}(\mathbf{seq} x(\mathbf{seq} y(\mathbf{Pair} x y)))(\mathbf{seq} y(\mathbf{seq} x(\mathbf{Pair} x y)))$
<code>dchoice</code>	$\equiv \lambda x.\lambda y.\mathbf{amb}(\mathbf{seq} x(\mathbf{seq} y x))(\mathbf{seq} y(\mathbf{seq} x y))$
<code>choice</code>	$\equiv \lambda x.\lambda y.(\mathbf{amb}(\lambda z_1.x)(\lambda z_2.y)) \mathbf{True}$
<code>or</code>	$\equiv \lambda x.\lambda y.(\mathbf{amb}(\mathbf{if} x \mathbf{then} \mathbf{True} \mathbf{else} y)(\mathbf{if} y \mathbf{then} \mathbf{True} \mathbf{else} x))$
<code>merge</code>	$\equiv \mathbf{letrec} m = \lambda xs.\lambda ys.\mathbf{amb}(\mathbf{case}_{List} xs (\lambda _ \rightarrow ys) (z : zs \rightarrow z : m zs ys))$ $\qquad\qquad\qquad (\mathbf{case}_{List} ys (\lambda _ \rightarrow xs) (z : zs \rightarrow z : m xs zs))$ $\qquad\qquad\qquad \mathbf{in} m$

Fig. 3. Encoding of Operators

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

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-converges* if it may evaluate to a WHNF using normal order reductions. If the opposite holds, a term *must-diverges*. 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 $A_{\mathbf{amb}}^{\mathbf{let}}$ a term *may-diverges* 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 $\Lambda_{\text{amb}}^{\text{let}}$. The advantage of reasoning with the chosen predicates is that our semantics fulfills a fairness property, without explicitly 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.*

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

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

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

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

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

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

Definition 2.14 (May- and Must-Divergence). *For a term t we write $t \uparrow$ iff there does not exist a sequence of normal order reductions starting with t that ends in a WHNF. Then we say t *must-diverges*, i.e.*

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

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

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

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

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

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

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
- $(t \uparrow \implies t \Downarrow)$.

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*

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

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

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

$$\begin{aligned} \mathbf{d}^0 &= A_{\text{amb}}^{\text{let}} \\ \mathbf{d}^i &= \mathcal{D}(\mathbf{d}^{i-1}) \end{aligned}$$

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

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

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

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

□

Corollary 2.18. *Since set-inclusion forms a complete lattice, with Tarski's fixpoint theorem (see e.g. [DP92]) follows that \mathcal{D} has a greatest fixpoint.*

Lemma 2.19. *The operator \mathcal{D} is lower continuous, i.e. for every non-empty descending countably infinite chain $S_1 \supseteq S_2 \dots$ with $S_i \subseteq A_{\text{amb}}^{\text{let}}$ the following holds: $\mathcal{D}(\bigcap_i S_i) = \bigcap_i \mathcal{D}(S_i)$.*

Proof. Obviously $\forall i : \bigcap_i S_i \subseteq S_i$. With monotonicity of \mathcal{D} follows $\forall i : \mathcal{D}(\bigcap_i S_i) \subseteq \mathcal{D}(S_i)$ and thus $\mathcal{D}(\bigcap_i S_i) \subseteq \bigcap_i \mathcal{D}(S_i)$.

For the other direction, let $t \in \bigcap_i \mathcal{D}(S_i)$, then t is not a WHNF and for all $t' : t \xrightarrow{\text{no}} t' \implies t' \in S_i$ for every i . Then also holds $\forall t' : t \xrightarrow{\text{no}} t' \implies t' \in \bigcap_i S_i$. Hence $t \in \mathcal{D}(\bigcap_i S_i)$.

Lemma 2.20. *$s \in \mathcal{BOT}$ iff $\forall j : s \in \mathbf{d}^j$.*

Proof. Since \mathcal{D} is (lower) continuous Kleene's fixpoint theorem is applicable, i.e. the greatest fixed point can be represented as $\text{gfp}(\mathcal{D}) = \bigcap_i \mathbf{d}^i$. □

Lemma 2.21. *Let s, t be terms with $s \xrightarrow{\text{no}} t$. If $s \in \mathcal{BOT}$ then $t \in \mathcal{BOT}$.*

Proof. We use Lemma 2.20. From s in \mathcal{BOT} we have, for all j : $s \in \mathbf{d}^j$. Since $s \xrightarrow{\text{no}} t$, we have $\forall j > 0 : t \in \mathbf{d}^j$. It remains to prove $t \in \mathbf{d}^0$, but that holds by definition. □

Corollary 2.22. *If $s \xrightarrow{\text{no},*} t$ with $s \in \mathcal{BOT}$. Then $t \in \mathcal{BOT}$*

Lemma 2.23. $\mathcal{BOT} = \mathcal{MD}$

Proof. $\mathcal{BOT} \subseteq \mathcal{MD}$: By definition of \mathcal{BOT} we have that \mathcal{BOT} does not contain terms in WHNF. Then Corollary 2.22 shows the claim.

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

- s has no normal order reduction. Since s cannot be in WHNF $s \in \mathcal{D}(\mathcal{MD})$
- s has at least one normal order reduction. For every t with $s \xrightarrow{\text{no}} t$ 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 $\forall t : s \xrightarrow{\text{no}} t \implies t \in \mathcal{MD}$ and since s is not in WHNF we have $s \in \mathcal{D}(\mathcal{MD})$.

□

3 Fair Normal Order Reduction

Our normal order reduction does not take fairness into account, i.e. the set of all sequences of normal order reductions for a given term t may include infinite sequences where a normal order redex is infinitely often not chosen for reduction.

For example we consider the term $t \equiv (\text{letrec } x = (\lambda y.y \ y) \text{ in amb } (x \ x) \ \text{True})$. Then there exists the infinite sequence of normal order reductions that repeats the three reductions $\xrightarrow{\text{no,cp}} \xrightarrow{\text{no,lbeta}} \xrightarrow{\text{no,lamb-l}}$ infinitely often, i.e. this sequence begins as follows

$$\begin{aligned}
& (\text{letrec } x = (\lambda y.y \ y) \text{ in amb } (x \ x) \ \text{True}) \\
\frac{\text{no,cp}}{\longrightarrow} & (\text{letrec } x = (\lambda y.y \ y) \text{ in amb } ((\lambda x_1.x_1 \ x_1) \ x) \ \text{True}) \\
\frac{\text{no,lbeta}}{\longrightarrow} & (\text{letrec } x = (\lambda y.y \ y) \text{ in amb } ((\text{letrec } x_1 = x \text{ in } (x_1 \ x_1))) \ \text{True}) \\
\frac{\text{no,lamb-l}}{\longrightarrow} & (\text{letrec } x = (\lambda y.y \ y), x_1 = x \text{ in amb } (x_1 \ x_1) \ \text{True}) \\
\frac{\text{no,cp}}{\longrightarrow} & (\text{letrec } x = (\lambda y.y \ y), x_1 = x \text{ in amb } ((\lambda x_2.x_2 \ x_2) \ x_1) \ \text{True})
\end{aligned}$$

This sequence is not fair, since the (amb-r-c)-redex is always avoided by normal order reduction. Nevertheless, t is must-convergent in our calculus. Hence, our notion of convergence already introduces a kind of fairness on the semantic level. A similar observation has already been made in [CHS05] for a call-by-name calculus with **amb**.

This example also shows that without fair evaluation our operator **amb** is not bottom-avoiding, since evaluation not eventually chooses the terminating argument of **amb**. Already [Mor98] remarked that fairness of the evaluator is *needed* to implement bottom-avoiding choice. Hence, in this section we will define *fair normal order* reduction, such that those unfair sequences of reductions are forbidden.

The result of this section is that the notions of convergence and divergence using fair evaluation are the same as for our normal order reduction. This is of

great value, since we can use the normal order reduction for reasoning, and all results are transferable to fair evaluation.

For implementing fair evaluation we considered two different approaches: The approach of [Mor98] by using resource annotations for every **amb**-expression and the more elegant approach of [CHS05] by defining a small-step semantics that allows to reduce both arguments of an **amb**-expression in parallel. Unfortunately the second approach does not properly work for calculi with shared bindings, since it is difficult to define a parallel reduction step if both redexes modify the same environment, e.g. for the expression (**letrec** $x = c_T s_1 \dots s_{ar(c_T)}$ **in amb** (**case** _{T} $x \dots$) (**case** _{T} $x \dots$)). Thus, we chose to use the approach of [Mor98] and annotate the **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 $\mathbf{amb}_{\langle m, n \rangle}$. The set of all annotated variants of a term s is denoted with $\mathbf{ann}(s)$. With $\mathbf{ann}_0(s)$ 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 $(\mathbf{amb}_{\langle m, n \rangle} s t)$ 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 $da(C)$ is a context. An annotated variant s of a term is a WHNF iff $da(s)$ is a WHNF.*

We now give a description of a non-deterministic unwinding algorithm \mathcal{UWF} that leads to fair evaluation. The algorithm performs three tasks: It finds a position where a normal order reduction may be applied, it decreases the annotations for the path that leads to this position and if necessary it performs scheduling by increasing the annotations. Let s be an annotated variant of a term. If $s \equiv (\mathbf{letrec} \textit{Env} \mathbf{in} s')$ then apply \mathbf{uwf} to the pair $(s', (\mathbf{letrec} \textit{Env} \mathbf{in} [\cdot]))$, otherwise apply \mathbf{uwf} to the pair $(s, [\cdot])$. For detecting loops the algorithm marks the visited **letrec**-bindings (i.e. with $\overset{\bullet}{=}$). We assume that these markers are

removed before returning the result (s, R) .

$$\begin{aligned}
& \mathbf{uwf}((s \ t), R) \rightarrow \mathbf{uwf}(s, R[(\cdot) \ t]) \\
& \mathbf{uwf}((\mathbf{seq} \ s \ t), R) \rightarrow \mathbf{uwf}(s, R[(\mathbf{seq} \ [\cdot] \ t)]) \\
& \mathbf{uwf}((\mathbf{case} \ s \ \mathit{alts}), R) \rightarrow \mathbf{uwf}(s, R[(\mathbf{case} \ [\cdot] \ \mathit{alts})]) \\
& \mathbf{uwf}((\mathbf{amb}_{\langle m+1, n \rangle} \ s \ t), R) \rightarrow \mathbf{uwf}(s, R[(\mathbf{amb}_{\langle m, n \rangle} \ [\cdot] \ t)]) \\
& \mathbf{uwf}((\mathbf{amb}_{\langle m, n+1 \rangle} \ s \ t), R) \rightarrow \mathbf{uwf}(t, R[(\mathbf{amb}_{\langle m, n \rangle} \ s \ [\cdot])]) \\
& \mathbf{uwf}((\mathbf{amb}_{\langle 0, 0 \rangle} \ s \ t), R) \rightarrow \mathbf{uwf}((\mathbf{amb}_{\langle m, n \rangle} \ s \ t), R), \text{ where } m, n > 0 \\
& \mathbf{uwf}(x, (\mathbf{letrec} \ x = s, \mathit{Env} \ \mathbf{in} \ R^-)) \rightarrow \mathbf{uwf}(s, (\mathbf{letrec} \ x \doteq [\cdot], \mathit{Env} \ \mathbf{in} \ R^-[x])) \\
& \mathbf{uwf}(x, (\mathbf{letrec} \ y \doteq R^-, x = s, \mathit{Env} \ \mathbf{in} \ t)) \\
& \quad \rightarrow \mathbf{uwf}(s, (\mathbf{letrec} \ y \doteq R^-[x], x \doteq [\cdot], \mathit{Env} \ \mathbf{in} \ t)) \\
& \mathbf{uwf}(s, R) \rightarrow (s, R) \text{ if no other rule is applicable}
\end{aligned}$$

The unwinding algorithm \mathcal{UWF} always terminates with a result (s, R) .

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

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

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 $\mathbf{amb}_{\langle m, n \rangle}$ constructs with \mathbf{amb} , we can perform every step with the algorithm \mathcal{UW} , too. \square

Fair normal order reduction $\xrightarrow{\mathbf{fno}}$ on annotated variants consists of one or more executions of \mathcal{UWF} followed by a normal order reduction:

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 perform the following steps*

1. *Execute \mathcal{UWF} starting with s . Let the result be (s'', R) .*
2. *If no normal order reduction is applicable to $R[s'']$ with maximal reduction context R , then go to step 1 using the variant $R[s'']$ instead of s .*
3. *If a normal order reduction is applicable to $R[s'']$ with maximal reduction context R , then apply this reduction where annotations are inherited like labels in labelled reduction (see [Bar84]). Let the result be t .*

Then $s \xrightarrow{\mathbf{fno}} t$.

Note that for the case that step 2 matches, the new search of \mathcal{UWF} starts with the result of the previous execution of \mathcal{UWF} . This is necessary to decrease the annotations for a subterm that cannot be reduced, since it has a typing error (e.g. $\text{case}_{List} \text{True} \dots$) or it is a term with a black hole, e.g. $(\text{letrec } x = x \text{ in } x)$. [Mor98] uses an additional reduction rule for these cases. We decrease the annotation by executing the unwinding algorithm again with another variant of the same term, where annotations are decreased.

A fair normal order reduction sequence (denoted with F as subscript) is a sequence consisting of fair normal order reductions.

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

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

A term t fair may-converges (denoted with $t \downarrow_F$) iff $\text{ann}_0(t) \downarrow_F$, a term t fair must-converges (denoted with $t \downarrow_F$) iff $\text{ann}_0(t) \downarrow_F$.

We use \uparrow_F and \uparrow_F for the logical counterparts of both convergence predicates.

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

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

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

Lemma 3.9. *If $s \xrightarrow{\text{no},*} t$, then there exists $t' \in \text{ann}(t)$ with $\text{ann}_0(s) \xrightarrow{\text{fno},*} t'$.*

Proof. Let $s \xrightarrow{RED} t$, where RED is a sequence of normal order reductions. If RED is empty then the claim follows with $t' = \text{ann}_0(t)$. Otherwise, let $RED = \text{red}_1 \dots \text{red}_k$. 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 $R''_i[s''_i] \in \text{ann}(s_i)$. Now, let m be the sum of all steps $\text{uw}(\text{amb } t_1 t_2, R) \rightarrow \dots$ 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 $\text{uwf}(\text{amb}_{(0,0)}, R')$ then insert the step $\text{uwf}(\text{amb}_{(0,0)}, R') \rightarrow \text{uwf}(\text{amb}_{\langle m, m \rangle}, R')$. 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. \square

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. \square

Lemma 3.12. *Let s be term, $s' \in \text{ann}(s)$ 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. \square

Lemma 3.13. *Let s be a term with $s \uparrow$ then $\text{ann}_0(s) \uparrow_F$.*

Proof. Let $RED \in \mathcal{DTV}(s)$, then $s \xrightarrow{RED} t$ and $t \uparrow$. Lemma 3.9 shows that $\text{ann}_0(s) \xrightarrow{\text{fno},*} t'$, with $t' \in \text{ann}(t)$. From Lemma 3.12 we have $t' \uparrow_F$ and thus $\text{ann}_0(s) \uparrow_F$. \square

Lemma 3.14. *Let s be a term with $s \downarrow$, $s \xrightarrow{\text{no}} s'$ then $s' \downarrow$.*

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

For the remaining part, i.e. to show that $t \downarrow \implies \text{ann}_0(t) \downarrow_F$, we introduce *well-annotated variants*.

Definition 3.15. *An annotated variant is well-annotated if **amb**-expressions inside the body of abstractions are annotated with both resources zero.*

Fair normal order reduction preserves the property of an annotated variant being well annotated, and for all expressions s , $\text{ann}_0(s)$ is well-annotated.

Lemma 3.16. *Let s be a well-annotated variant, then $s \uparrow_F$ implies $\text{da}(s) \uparrow$.*

Proof. We reason by contradiction and assume that there exists a well-annotated variant s with $s \uparrow_F$ such that $s \downarrow$. Let $P(s) = \{p_1, \dots, p_n\}$ be the set of surface positions in s with $s|_{p_i} = \mathbf{amb}_{\langle m, n \rangle} s_m s_n$ such that $(m, n) \neq (0, 0)$. For every true successor t of s w.r.t fair normal order reduction, i.e. $s \xrightarrow{\text{fno},*} s' \xrightarrow{\text{fno}} t$, we iteratively calculate a set $P(t)$ of positions using the set $P(s')$ of the direct predecessor of t as follows:

Let $P(s')$ be given, where $s'_{|p'_i}$ is an **amb**-expression for all $p'_i \in P(s')$. We search the successors of the **amb**-expressions of s' at positions $P(s')$ in t (using labelled reduction). For every $p'_i \in P(s')$ there are two possibilities:

1. The **amb**-expression at position p'_i in s' is removed by the fair normal order reduction. This may happen if the fair normal order reduction is an (**amb**)-, a (**case**)-, or a (**seq**)-reduction.
2. There exists a position p''_i of t such that $t|_{p''_i}$ is the successor of the **amb**-expression in s' at position p'_i .

The set $P(t)$ is defined according to the following rules for every $p'_i \in P(s')$:

- If case 1 applies, then for $p'_i \in P(s')$ no position is inserted into $P(t)$.
- Otherwise, if (case 2) applies, there exists the according position p''_i in t . Now, p''_i is inserted into $P(t)$ except for the following two cases:
 - If the resources of the **amb** expression at position p''_i of t are both zero, or
 - if the fair unwinding algorithm schedules the resources of the **amb**-expression at position p'_i of s during the reduction $s \xrightarrow{\text{fno}} t$.

The measure $Q(r)$ is defined as the sum of all resources in $P(r)$.

Now we choose a successor t_{min} of s w.r.t. $\xrightarrow{\text{fno},*}$, that is minimal w.r.t. the measure $\mu(t_i) = (Q(t_i), k_i)$ where k_i is the number of fair normal order reductions from s to t_i , i.e. $s \xrightarrow{\text{fno},k_i} t_i$. Note, that there may be (countably) infinitely many of those minimal successors. Then we select one of them arbitrarily. Note also that for t_{min} the number k_{min} is defined, since μ is well-founded.

We inspect the reduction sequence $s \xrightarrow{\text{fno},k_{min}} t_{min}$. Let P' be the set of surface positions of **amb**-expressions in t_{min} that are not included in $P(t_{min})$. The respective **amb**-expressions either have only (0,0)-resources, or they have been scheduled during the reductions from s to t_{min} . We only take into account those **amb**-expressions with position in P' where not both resources are 0.

If an **amb**-expression is scheduled more than once, then it is obvious that we can do all these schedulings by performing the first scheduling with adjusted resources. This modification neither changes the measure μ nor the expression t_{min} . Hence, we assume that every of these **amb**-expressions is scheduled at most once. Let such a scheduling step be $\text{amb}_{\langle 0,0 \rangle} s_1 s_2 \rightarrow \text{amb}_{\langle m,n \rangle} s_1 s_2$. If we replace this scheduling step by $\text{amb}_{\langle 0,0 \rangle} s_1 s_2 \rightarrow \text{amb}_{\langle m',n' \rangle} s_1 s_2$ where $m' \geq m$ and $n' \geq n$, then there exists a reduction sequence $s \xrightarrow{\text{fno},k_{min}} t'_{min}$, such that t_{min} and t'_{min} can only differ insofar as one **amb**-subexpression of t'_{min} has higher resources than t_{min} . If we replace some scheduling steps of different **amb**-expressions, the same holds, i.e. there exists t''_{min} with $s \xrightarrow{\text{fno},k_{min}} t''_{min}$, and $da(t''_{min}) = da(t_{min})$ where t''_{min} may have higher resources for **amb**-subexpressions than t_{min} and the positions of these **amb**-expressions are disjoint from $P(t_{min}) = P(t'_{min})$. That implies that t''_{min} is also minimal w.r.t. μ .

The goal is now to show that $t''_{min} \downarrow_F$. From $da(s) \downarrow$, Lemma 3.7, and Lemma 3.14 it follows that $da(t_{min}) \downarrow$. Let t_r be a WHNF such that $da(t_{min}) \xrightarrow{\text{no},k_r} t_r$ and k_r is minimal. We show that there exists a t'_{min} derived from t_{min} as explained above, such that $t'_{min} \xrightarrow{\text{fno},k_r} t'_r$ with $t'_r \in \text{ann}(t_r)$.

Therefore we try to perform the sequence $da(t_{min}) \xrightarrow{\text{no}, k_r} t_r$ as a fair normal order reduction sequence for t_{min} , and thereby derive the expression t''_{min} for which this fair normal order reduction is possible, as we show. If one reduction step cannot be performed, then the only reason may be that there is some **amb**-expression where one resource is 0 and thus fair unwinding must choose the wrong (and other) argument of this **amb**-expression. This **amb**-expression cannot be an **amb**-expression with a position in $P(t_{min})$:

If the second resource is also 0 then, this is obvious, Otherwise let the other resource be $n > 0$, then the fair normal order reduction can decrease this resource using unwindings (and may be reductions), but then t_{min} is not a minimal example w.r.t. Q .

Thus we assume that the **amb**-expression does not belong to $P(t_{min})$. But then we can add more resources to this **amb**-expression during the construction of t''_{min} , say at least a number that is greater than k_r , and thus fair normal order reduction can proceed.

Since t_r is a WHNF it follows that t'_r is a WHNF, and thus $s \xrightarrow{\text{fno}, *}$ $t''_{min} \xrightarrow{\text{fno}, *}$ t'_r what contradicts $s \uparrow_F$. \square

Proposition 3.17. *Let s be a well-annotated variant and $da(s) \Downarrow$, then $s \Downarrow_F$.*

Proof. We show that $s \uparrow_F$ implies $da(s) \uparrow$ for every well-annotated variant s . Let $s \xrightarrow{\text{fno}, k} s'$ such that $s' \uparrow_F$. By Lemma 3.7 we have $da(s) \xrightarrow{\text{fno}}$ $da(s')$ and by Lemma 3.16 we have $da(s') \uparrow$, and thus $da(s) \uparrow$. \square

Corollary 3.18. *Let s be a term with $s \Downarrow$, then $s \Downarrow_F$.*

Remark 3.19. Proposition 3.17 is more general than needed for Corollary 3.18, since it shows that must-convergence implies fair must convergence for every well-annotated variant.

Note, that the inverse is not true, since annotations may introduce must-convergence, e.g. the well-annotated variant $s := \text{if } (\text{amb}_{\langle 0,1 \rangle} \text{ True False}) \text{ then } \Omega \text{ else True}$ is fair must-convergent since it must reduce to **True**, while $da(s) \uparrow$.

Note also, that Proposition 3.17 does not hold for all annotated variants: Consider the annotated variant s (which is not well-annotated):

$$\begin{aligned} s := \text{letrec } f &= \lambda x. (\text{amb}_{\langle 1,0 \rangle} \text{ True False}) \\ &g = \lambda y. \text{if}(f \text{ True}) \text{ then } g \text{ True else True} \\ \text{in } (g \text{ True}) \end{aligned}$$

Then $da(s) \Downarrow$, since every successor w.r.t. normal order reduction remains may-convergent, but $s \uparrow_F$, since due to the resource annotations, the subexpression $(\text{amb}_{\langle 1,0 \rangle} \text{ True False})$ always reduces to **True**.

Theorem 3.20. *For all terms t : $(t \Downarrow \text{ iff } t \Downarrow_F)$ and $(t \Downarrow \text{ iff } t \Downarrow_F)$.*

Proof. Follows from Proposition 3.11, Lemma 3.13 and Corollary 3.18. \square

4 Contextual Equivalence and Proof Techniques

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:*

$$\begin{aligned} s \leq_c^\downarrow t &\text{ iff } (\forall C \in \mathcal{C} : C[s] \downarrow \Rightarrow C[t] \downarrow) \\ s \leq_c^\Downarrow t &\text{ iff } (\forall C \in \mathcal{C} : C[s] \Downarrow \Rightarrow C[t] \Downarrow) \\ s \leq_c t &\text{ iff } s \leq_c^\downarrow t \wedge s \leq_c^\Downarrow t \end{aligned}$$

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 $C[s], C[t]$ may be open terms.

Remark 4.2. It makes no difference if we replace the convergence predicates based on normal order reduction with predicates based on fair normal order reduction in the definitions of \leq_c^\downarrow and \leq_c^\Downarrow , since Theorem 3.20 holds, i.e.

$$s \leq_c t \text{ iff } (\forall C \in \mathcal{C} : C[s] \downarrow_F \Rightarrow C[t] \downarrow_F) \wedge (\forall C \in \mathcal{C} : C[s] \Downarrow_F \Rightarrow C[t] \Downarrow_F)$$

We choose to work with the predicates for normal order reduction, since that makes reasoning considerably easier, since we do not need to take care of resource annotations.

Remark 4.3. In Section 8 (Corollary 8.39) we show that contextual equivalence can be defined using only must-convergence, i.e. $s \sim_c t$ iff $\forall C : C[s] \Downarrow \Leftrightarrow C[t] \Downarrow$.

Even if we could prove the statement of the previous remark at this time, it would not simplify our proofs, since for proving $s \leq_c^\Downarrow t$ we always require as precondition that $t \leq_c^\downarrow s$ holds.

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 holds for 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 4.4. 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

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

then

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

Now, we consider a contextual equivalence \sim_M that is the same as \sim_c with the only difference that a term t must-converges 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

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

Weak divergences are typical for some kinds of reactive systems, i.e. those systems that run until they are successfully terminated by some user input (e.g. for operating systems a shut-down command). Those programs should not be equivalent to programs that loop forever without any chance to terminate, i.e. programs that may *strongly* diverge.

Example 4.5. Assuming that integers and an operator $+$ for addition of integers are implemented using Peano numbers, then by using the operator **choice** we can define an expression t that generates a natural number as follows

$$t \equiv (\mathbf{letrec} \ gen = \lambda x.\mathbf{choice} \ x \ (gen \ (x + 1)) \ \mathbf{in} \ gen \ 1)$$

The expression t is must-convergent in our calculus, but may-divergent using Moran's semantics, since if every **choice** chooses the second argument, the evaluation of t may not terminate (i.e. t is weakly divergent). Using Moran's semantics the equivalence $t \sim_M t'$ holds, where t' is defined as

$$t' \equiv (\mathbf{letrec} \ gen = \lambda x.\mathbf{choice} \ x \ (gen \ (x + 1)) \ \mathbf{in} \ \mathbf{choice} \ \Omega \ (gen \ 1)).$$

I.e. if the equivalence of t and t' is used as a program transformation, a possibility for a strong divergence is introduced.

The operator **choice** can model the input of a user in such a way that if the user says "Yes" then it chooses the second argument of **choice** otherwise it chooses the first argument. Using this view t is a small reactive system that implements a stopwatch. Obviously, t' does no longer implement a stopwatch.

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

Example 4.6. Ω is not least w.r.t. \leq_c : Let $C \equiv (\mathbf{amb} \ (\lambda x.\lambda y.x) \ [\cdot] \ \Omega)$, then the term $C[\mathbf{I}]$ may-diverges whereas $C[\Omega]$ must-converges, 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 4.7. \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 $C_2[C_1[s]] \not\Downarrow$ we can reason in the same way. \square

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 4.8. *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 $\forall C \in \mathcal{C} : C[t] \Uparrow \implies C[s] \Uparrow$.

A main contribution of our work is showing all deterministic reduction rules being correct program transformations:

Definition 4.9 (Correct Program Transformation). *A binary relation P on terms is a correct program transformation iff \forall terms $s, t : s P t \implies s \sim_c t$.*

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 set \mathcal{C} . [Mor98] also provides a context lemma for his call-by-need calculus, but only for may-convergence. We obtained improved results by providing a context lemma for may- as well as must-convergence.

The structure of this section is as follows: We first 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 pre-congruence.

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 $C[\cdot_1, \dots, \cdot_n]$, and if the terms s_i for $i = 1, \dots, n$ are placed into the holes \cdot_i , then we denote the resulting term as $C[s_1, \dots, s_n]$.

Lemma 4.10. *Let C be a multicontext with n holes then the following holds:*

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

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

Note that the numbers i and j in the previous lemma are not necessarily identical, e.g. for the multicontext $(\mathbf{letrec} \ x = [\cdot_1], y = [\cdot_2] \ \mathbf{in} \ y)$ and the term $s_2 \equiv x$ the context $(\mathbf{letrec} \ x = [\cdot_1], y = s_2 \ \mathbf{in} \ y)$ is a reduction context. If we replace s_2 by another term t_2 e.g. $t_2 \equiv y$ then $(\mathbf{letrec} \ x = [\cdot_1], y = t_2 \ \mathbf{in} \ y)$ is not a reduction context, but $(\mathbf{letrec} \ x = t_1, y = [\cdot] \ \mathbf{in} \ y)$ is a reduction context for any term t_1 .

Lemma 4.11. *Let s, t be expressions, σ be a permutation on variables, then*

$$\begin{aligned} & - (\forall R \in \mathcal{R} : R[s] \downarrow \implies R[t] \downarrow) \implies (\forall R \in \mathcal{R} : R[\sigma(s)] \downarrow \implies R[\sigma(t)] \downarrow), \text{ and} \\ & - (\forall R \in \mathcal{R} : R[s] \uparrow \implies R[t] \uparrow) \implies (\forall R \in \mathcal{R} : R[\sigma(s)] \uparrow \implies R[\sigma(t)] \uparrow). \end{aligned}$$

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. \square

We now prove a lemma using multicontexts which is more general than needed, since the context lemma for may-convergence (Lemma 4.13) is a specialisation of the claim.

Lemma 4.12. *For all $n \geq 0$ and for all multicontexts C with n holes and for all expressions s_1, \dots, s_n and t_1, \dots, t_n :*

If for all $i = 1, \dots, n$: $\forall R \in \mathcal{R} : (R[s_i] \downarrow \implies R[t_i] \downarrow)$, then $C[s_1, \dots, s_n] \downarrow \implies C[t_1, \dots, t_n] \downarrow$.

Proof. The proof is by induction, where n, C, s_i, t_i for $i = 1, \dots, n$ are given. The induction is on the measure (l, n) , where

- l is the length of the shortest reduction sequence in $\mathcal{CON}(C[s_1, \dots, s_n])$.
- n is the number of holes in C .

We assume that the pairs are ordered lexicographically, thus this measure is well-founded. The claim holds for $n = 0$, i.e., 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 that the claim holds for all $n', C', s'_i, t'_i, i = 1, \dots, n'$ with $(l', n') < (l, n)$. Let us assume that the precondition holds, i.e., that $\forall i : \forall R \in \mathcal{R} : (R[s_i] \downarrow \implies R[t_i] \downarrow)$. Let RED be a shortest reduction sequence in $\mathcal{CON}(C[s_1, \dots, s_n])$ with $\mathbf{rl}(RED) = l$. We distinguish two cases:

- There is some index j , such that $C[s_1, \dots, s_{j-1}, \cdot_j, s_{j+1}, \dots, s_n]$ is a reduction context. Lemma 4.10 implies that there is a hole \cdot_i such that $R_1 \equiv C[s_1, \dots, s_{i-1}, \cdot_i, s_{i+1}, \dots, s_n]$ and $R_2 \equiv C[t_1, \dots, t_{i-1}, \cdot_i, t_{i+1}, \dots, t_n]$ are both 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. $C[t_1, \dots, t_n] \downarrow$.
- There is no index j , such that $C[s_1, \dots, s_{j-1}, \cdot_j, s_{j+1}, \dots, s_n]$ is 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 $C[t_1, \dots, t_n]$, i.e., the position of the redex and the inner redex are the same. 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 . We now argue that the elimination, duplication or variable permutation for every s_i can also be applied to t_i . More formally, we will show if $C[s_1, \dots, s_n] \xrightarrow{\text{no}, a} C'[s'_1, \dots, s'_m]$, then there exists a variable permutation ρ such that for every $i = 1, \dots, m$, there is some j such that $(s'_i, t'_i) \equiv (\rho(s_j), \rho(t_j))$; Moreover, $C[t_1, \dots, t_n] \xrightarrow{\text{no}, a} C'[t'_1, \dots, t'_m]$.

We go through the cases of Definition 2.8 and figure out how the terms s_i and t_i are modified by the reduction step, where we only mention the interesting cases.

- If \cdot_i is in an alternative of **case**, which 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 item, then s_i and t_i can only change their respective 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 4.11 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 $C'[s'_1, \dots, s'_m]$ has a terminating sequence of normal order reductions of length $l - 1$ we also have $C'[t'_1, \dots, t'_m] \downarrow$. With $C[t_1, \dots, t_n] \xrightarrow{\text{no}, a} C'[t'_1, \dots, t'_m]$ we have $C[t_1, \dots, t_n] \downarrow$. \square

Lemma 4.13 (Context Lemma for May-Convergence). *Let s, t be terms. If for all reduction contexts R : $(R[s] \downarrow \implies R[t] \downarrow)$, then $\forall C : (C[s] \downarrow \implies C[t] \downarrow)$; i.e. $s \leq_c^\downarrow t$.*

Proof. The Lemma is a specialisation of Lemma 4.12. \square

Lemma 4.14 (Context Lemma for Must-Convergence). *Let s, t be terms, then*

$$\begin{aligned} & \left((\forall R \in \mathcal{R} : R[s] \Downarrow \implies R[t] \Downarrow) \wedge (\forall R \in \mathcal{R} : R[s] \Downarrow \implies R[t] \Downarrow) \right) \\ & \implies \\ & (\forall C : C[s] \Downarrow \implies C[t] \Downarrow) \end{aligned}$$

Proof. We prove a more general claim using multicontexts and the contrapositive of the first of the inner implications: For all $n \geq 0$ and for all multicontexts C with n holes and for all expressions s_1, \dots, s_n and t_1, \dots, t_n holds: If

$$\left((\forall R \in \mathcal{R} : R[t_i] \Uparrow \implies R[s_i] \Uparrow) \wedge (\forall R \in \mathcal{R} : R[s_i] \Downarrow \implies R[t_i] \Downarrow) \right)$$

then $C[t_1, \dots, t_n] \Uparrow \implies C[s_1, \dots, s_n] \Uparrow$.

The proof is by induction where n, C, s_i, t_i for $i = 1, \dots, n$ are given. The induction is on the measure (l, n) , where

- l is the length of a shortest reduction sequence in $\mathcal{D}\mathcal{I}\mathcal{V}(C[t_1, \dots, t_n])$.
- n is 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 $C[t_1, \dots, t_m] \xrightarrow{\text{no}, l} t'$ 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 $R_1 \equiv C[t_1, \dots, t_{j-1}, \cdot, t_{j+1}, \dots, t_m]$ and $R_2 \equiv C[s_1, \dots, s_{j-1}, \cdot, s_{j+1}, \dots, s_m]$ being reduction contexts. Let $C' = C[\cdot_1, \dots, \cdot_{j-1}, t_j, \cdot_{j+1}, \dots, \cdot_m]$. Since $C'[t_1, \dots, t_{j-1}, t_{j+1}, \dots, t_m] \xrightarrow{\text{no}, l} t'$ and C' has $m - 1$ holes, we can use the induction hypothesis and have $C'[s_1, \dots, s_{j-1}, s_{j+1}, \dots, s_m] \Uparrow$ and hence $R_2[t_i] \Uparrow$. Using the precondition we have $R_2[s_i] \Uparrow$, thus $C[s_1, \dots, s_m] \Uparrow$.
- 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 $C[t_1, \dots, t_m]$ also for $C[s_1, \dots, s_m]$. With the same reasoning as in the proof of Lemma 4.13, we have that the results of the reduction are $C'[t'_1, \dots, t'_{m'}]$ and $C'[s'_1, \dots, s'_{m'}]$, where $(t'_i, s'_i) = (\rho(t'_j), \rho(s'_j))$ for a variable permutation ρ . With Lemma 4.11 we have that the precondition also holds for s'_i, t'_i . Since $C'[t'_1, \dots, t'_{m'}] \xrightarrow{\text{no}, l-1} t'$ we can use the induction hypothesis and have $C'[s'_1, \dots, s'_{m'}] \Uparrow$. Since $C[s_1, \dots, s_m] \xrightarrow{\text{no}} C[s'_1, \dots, s'_m]$, we have $C[s_1, \dots, s_m] \Uparrow$.
 - $l = 0$: Then $C[t_1, \dots, t_m] \Uparrow$. We assume that $\neg(C[s_1, \dots, s_m] \Uparrow)$, i.e. $C[s_1, \dots, s_m] \Downarrow$. Then we also have $C[s_1, \dots, s_m] \Downarrow$. Using the precondition and Lemma 4.12, we have $C[t_1, \dots, t_m] \Downarrow$ which is a contradiction. \square

Corollary 4.15. *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 4.14 we have $s \leq_c^\downarrow t$. \square

By combining the context lemma for may-convergence (Lemma 4.13) and the context lemma for must-convergence (Lemma 4.14) we derive the context lemma:

Lemma 4.16 (Context Lemma). *Let s, t be terms, then*

$$\left((\forall R \in \mathcal{R} : R[s] \downarrow \implies R[t] \downarrow) \wedge (\forall R \in \mathcal{R} : R[s] \downarrow \implies R[t] \downarrow) \right) \implies s \leq_c t$$

Proof. Follows from Lemma 4.13 and Lemma 4.14. \square

4.3 Properties of the (III)-Reduction

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

Lemma 4.17. *The following holds:*

1. *For all terms of the form $(\mathbf{letrec} \text{ Env}_1 \text{ in } R_1^-[(\mathbf{letrec} \text{ Env}_2 \text{ in } t)])$ where R_1^- is a weak reduction context, there exists a sequence of normal order (III)-reductions with*

$$\begin{aligned} & (\mathbf{letrec} \text{ Env}_1 \text{ in } R_1^-[(\mathbf{letrec} \text{ Env}_2 \text{ in } t)]) \\ & \xrightarrow{\text{no,III,+}} (\mathbf{letrec} \text{ Env}_1, \text{Env}_2 \text{ in } R_1^-[t]). \end{aligned}$$

2. *For all terms of the form*

$$\begin{aligned} & \mathbf{letrec} \text{ Env}_{1,x_1} = R_1^-[(\mathbf{letrec} \text{ Env}_2 \text{ in } t)], \{x_i = R_i^-[x_{i-1}]\}_{i=2}^m \\ & \text{in } R_{m+1}^-[x_m] \end{aligned}$$

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

$$\begin{aligned} & \mathbf{letrec} \text{ Env}_{1,x_1} = R_1^-[(\mathbf{letrec} \text{ Env}_2 \text{ in } t)], \{x_i = R_i^-[x_{i-1}]\}_{i=2}^m \\ & \text{in } R_{m+1}^-[x_m] \\ & \xrightarrow{\text{no,III,+}} (\mathbf{letrec} \text{ Env}_1, \text{Env}_2, x_1 = R_1^-[t], \{x_i = R_i^-[x_{i-1}]\}_{i=2}^m \text{ in } R_{m+1}^-[x_m]) \end{aligned}$$

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

$$R_1^-[(\mathbf{letrec} \text{ Env} \text{ in } t)] \xrightarrow{\text{no,III,*}} (\mathbf{letrec} \text{ Env} \text{ in } R_1^-[t])$$

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

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

Definition 4.18. For a given term t , the measure $\mu_{\text{III}}(t)$ 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 $\text{lrdepth}(s, t)$ of all **letrec**-subexpressions s of t , where lrdepth is defined as follows:

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

Example 4.19. Let $s \equiv (\text{letrec } x = ((\lambda y.y) (\text{letrec } z = \text{True in } z)) \text{ in } x)$ then $\mu_{\text{III}}(s) = (2, 1)$ where

$$\begin{aligned} &\text{lrdepth}(s, (\text{letrec } z = \text{True in } z)) \\ &= \text{lrdepth}(((\lambda y.y) (\text{letrec } z = \text{True in } z)), (\text{letrec } z = \text{True in } z)) \\ &= 1 + \text{lrdepth}((\text{letrec } z = \text{True in } z), (\text{letrec } z = \text{True in } z)) = 1 + 0 = 1 \end{aligned}$$

Proposition 4.20. The reduction (III) is terminating, I.e. there are no infinite reductions sequences consisting only of (C, III)-reductions.

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

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, Sab03b, SSSS04, SSSS05, Man05] as a proof tool for proving contextual equivalence of program transformations.

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

Definition 4.21. Let s, t be terms, red be a reduction of Definition 2.5, and \mathcal{X} be a set of contexts. A reduction $s \xrightarrow{X, \text{red}} t$ 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 $s \xrightarrow{i\mathcal{X}, \text{red}} t$.

A *reduction sequence* is of the form $t_1 \rightarrow \dots \rightarrow t_n$, where t_i are terms and $t_i \rightarrow t_{i+1}$ is a reduction or some other program transformation. In the following we introduce transformations on reduction sequences, by using the notation

$$\xrightarrow{X, \text{red}} \cdot \xrightarrow{\text{no}, a_1} \dots \xrightarrow{\text{no}, a_k} \rightsquigarrow \xrightarrow{\text{no}, b_1} \dots \xrightarrow{\text{no}, b_m} \cdot \xrightarrow{X, \text{red}_1} \dots \xrightarrow{X, \text{red}_h},$$

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

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

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

The transformation consists in replacing the prefix of RED with the result, i.e. the new reduction sequence starts with $s \xrightarrow{\text{no},b_1} r_1 \dots \xrightarrow{\text{no},b_m} r_m \xrightarrow{X,red_1} r_{m+1} \dots \xrightarrow{X,red_h} t$.

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 4.22 (Complete Sets of Commuting Diagrams / 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

$$\xrightarrow{X,red} \dots \xrightarrow{\text{no},a_1} \dots \xrightarrow{\text{no},a_k} \rightsquigarrow \xrightarrow{\text{no},b_1} \dots \xrightarrow{\text{no},b_m} \dots \xrightarrow{X,red_1} \dots \xrightarrow{X,red_{k'}} \rightsquigarrow$$

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 $t_0 \xrightarrow{X,red} t_1 \xrightarrow{\text{no}} \dots \xrightarrow{\text{no}} t_h$, 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

$$\xleftarrow{\text{no},a_1} \dots \xleftarrow{\text{no},a_k} \dots \xrightarrow{X,red} \rightsquigarrow \xrightarrow{X,red_1} \dots \xrightarrow{X,red_{k'}} \dots \xleftarrow{\text{no},b_1} \dots \xleftarrow{\text{no},b_m} \rightsquigarrow$$

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 $t_h \xleftarrow{\text{no}} \dots t_2 \xleftarrow{\text{no}} t_1 \xrightarrow{X,red} t_0$, where t_1 is not a WHNF and $t_0 \neq t_2$ at least one of the transformation rules from the set is applicable to a suffix of the sequence.

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 labelled with $*$ or $+$ exist.

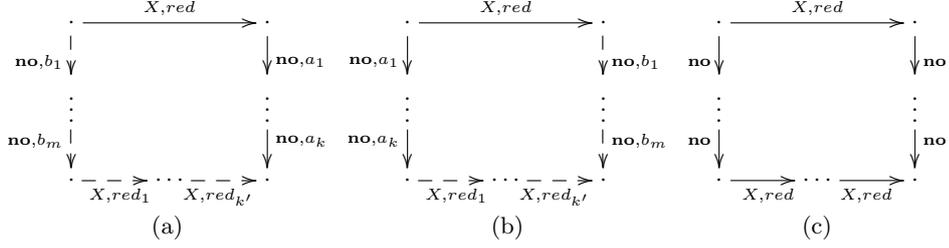


Fig. 4. Commuting (a) and Forking Diagrams (b) and their Common Representation (c)

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) and the \mathbb{CD} -Properties

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. The correctness proofs of the other reduction rules are more complex, hence as second part of this section we will provide general properties of a program transformation that ensure correctness of the transformation.

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

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

Proof. Let $red \in \{\text{lbeta}, \text{case-c}, \text{seq-c}, \text{amb-c}, \text{lapp}, \text{lcase}, \text{lseq}, \text{lamb}\}$, $s \xrightarrow{red} t$ 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 $R[s] \xrightarrow{no,a} R[t]$. By appending RED to $R[s] \xrightarrow{R,red} R[t]$ we have $R[s] \downarrow$. Hence, $\forall R \in \mathcal{R} : R[t] \downarrow \implies R[s] \downarrow$. The context lemma for may-convergence shows the claim. \square

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 $(\mathbf{no}, \mathbf{amb})$ -reduction, we require a non-normal order

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 $\overleftarrow{RED} s \xrightarrow{S, red} t$. Let RED' be the suffix of RED of length $l - 1$, then we have the following two possibilities:

$$\begin{array}{ccc}
 s \xrightarrow{S, red} t & & s \xrightarrow{S, red} t \\
 \text{no}, a \downarrow & & \text{no}, a \downarrow \\
 s' \xrightarrow{S, red} t' & & s' \xrightarrow{S, red} t' \\
 RED' \downarrow & & RED' \downarrow \\
 (1) & & (2)
 \end{array}$$

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

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

Proof. We use Corollary 4.15. 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 a reduction context, $R[s] \xrightarrow{R, red} R[t]$ and $R[t] \uparrow$. Since every reduction $red \in \{\text{beta}, \text{case-c}, \text{seq-c}\}$ in a reduction context is also a normal order reduction, we have $R[s] \uparrow$. \square

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

Proof. We use Corollary 4.15. 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 $RED \in \mathcal{D}\mathcal{I}\mathcal{V}(s_0)$. If $k = 0$, i.e. $s_0 \uparrow$, then the claim follows from Lemma 5.1 using Lemma 4.8. Now, let $k > 0$. Since s_0 cannot be a WHNF, we can apply a forking diagram to a suffix of $\overleftarrow{RED} s_0 \xrightarrow{S, red} t_0$. Let RED' be the suffix of RED of length $k - 1$. Then we have two cases:

$$\begin{array}{ccc}
 s_0 \xrightarrow{S, red} t_0 & & s_0 \xrightarrow{S, red} t_0 \\
 \text{no}, a \downarrow & & \text{no}, a \downarrow \\
 s'_0 \xrightarrow{S, red} t'_0 & & s'_0 \xrightarrow{S, red} t'_0 \\
 RED' \downarrow & & RED' \downarrow \\
 (1) & & (2)
 \end{array}$$

- (1) We can apply the induction hypothesis to $\overleftarrow{RED'} s'_0 \xrightarrow{S, red} t'_0$ and hence have $t'_0 \uparrow$. With $t_0 \xrightarrow{\text{no}, a} t'_0$, we have $t_0 \uparrow$.
- (2) For this case we have $t_0 \uparrow$, since $t_0 \xrightarrow{\text{no}, a} s'_0$. \square

Proposition 5.8. *The reductions (lbeta), (case-c) and (seq-c) are correct program transformations.*

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

5.2 Properties for Correctness of Program Transformations

The proofs of correctness of the remaining program transformations are more difficult. We postpone these proofs and now provide general properties of program transformations and show that their validity guarantees correctness of the transformation.

Definition 5.9. *Let P be a program transformation. Then P fulfills the \mathbb{CD} -properties if all of the following properties hold for all expressions s, t with $s \xrightarrow{P} t$:*

- $\mathbb{C}_{\Rightarrow}(P)$: $\forall S \in \mathcal{S}$: for all $RED_s \in \mathcal{CON}(S[s])$ there exists $RED_t \in \mathcal{CON}(S[t])$.
- $\mathbb{C}_{\Leftarrow}(P)$: $\forall S \in \mathcal{S}$: for all $RED_t \in \mathcal{CON}(S[t])$ there exists $RED_s \in \mathcal{CON}(S[s])$.
- $\mathbb{D}_{\Rightarrow}(P)$: $\forall S \in \mathcal{S}$: for all $RED_s \in \mathcal{DTV}(S[s])$ there exists $RED_t \in \mathcal{DTV}(S[t])$.
- $\mathbb{D}_{\Leftarrow}(P)$: $\forall S \in \mathcal{S}$: for all $RED_t \in \mathcal{DTV}(S[t])$ there exists $RED_s \in \mathcal{DTV}(S[s])$.

Theorem 5.10. *If a program transformation P fulfills the \mathbb{CD} -properties then P is correct.*

Proof. Let s, t be expressions with $s \xrightarrow{P} t$. Then $\mathbb{C}_{\Rightarrow}(P)$ implies that $\forall S \in \mathcal{S} : S[s] \downarrow \implies S[t] \downarrow$. Using Lemma 2.15 it follows from $\mathbb{D}_{\Leftarrow}(P)$ that $\forall S \in \mathcal{S} : S[s] \Downarrow \implies S[t] \Downarrow$. Hence, the context lemma (Lemma 4.16) shows $s \leq_c t$. The relation $t \leq_c s$ follows analogously from $\mathbb{C}_{\Leftarrow}(P)$ and $\mathbb{D}_{\Rightarrow}(P)$.

The next lemma is useful when proving properties $\mathbb{D}_{\Leftarrow}(P)$ and $\mathbb{D}_{\Rightarrow}(P)$, since their base cases (i.e. RED_s (RED_t , respectively) has length 0) directly follow from the validity of $\mathbb{C}_{\Leftarrow}(P)$ and $\mathbb{C}_{\Rightarrow}(P)$.

Lemma 5.11. *Let s, t be expression with $s \xrightarrow{P} t$ where P is a program transformation that fulfills the properties $\mathbb{C}_{\Rightarrow}(P)$ and $\mathbb{C}_{\Leftarrow}(P)$. Then $\forall C \in \mathcal{C} : C[s] \uparrow$ iff $C[t] \uparrow$.*

Proof. Using the context lemma for may-convergence from $\mathbb{C}_{\Rightarrow}(P)$ and $\mathbb{C}_{\Leftarrow}(P)$ follows $s \leq_c^\downarrow t$ and $t \leq_c^\downarrow s$. With Lemma 4.8 the claim follows.

The following lemma shows that for proving the properties $\mathbb{C}_{\Leftarrow}(P)$ and $\mathbb{D}_{\Leftarrow}(P)$ it is sufficient to consider only transformations inside surface contexts that are internal, i.e. not a normal order reduction.

Lemma 5.12. *Let P be a program transformation. Then the following properties hold:*

1. *If for all expressions s, t with $s \xrightarrow{iS, P} t$ and for all $RED_t \in \mathcal{CON}(t)$ there exists $RED_s \in \mathcal{CON}(s)$, then $\mathbb{C}_{\Leftarrow}(P)$.*

2. If for all expressions s, t with $s \xrightarrow{iS, P} t$ and for all $RED_t \in \mathcal{DTV}(t)$ there exists $RED_s \in \mathcal{DTV}(s)$, then $\mathbb{D}_{\Leftarrow}(P)$.

Proof. Let S be a surface context, s, t be expressions with $s \xrightarrow{S, P} t$. It is sufficient to consider the case where $s \xrightarrow{S, P} t$ is a normal order reduction. For every reduction sequence $RED_t \in \mathcal{CON}(t)$ ($RED_t \in \mathcal{DTV}(t)$, resp.) a reduction sequence $RED_s \in \mathcal{CON}(s)$ ($RED_s \in \mathcal{DTV}(s)$, resp.) can be constructed by appending RED_t to $s \xrightarrow{S, P} t$.

Note that for program transformations that are never used as normal order reduction (e.g. the transformations in Section 6), every application inside a surface context is internal.

In the following sections we show that the reduction rules and further program transformations fulfill the \mathbb{CD} -properties and thus are correct.

We describe in general the method we will use for proving the properties: Let P be a program transformation. Roughly speaking, the proof of properties $\mathbb{C}_{\Rightarrow}(P)$ ($\mathbb{D}_{\Rightarrow}(P)$, resp.) is by induction on the length of the reduction sequence RED_s where the induction step uses the complete set of forking diagrams for P . Some inductions do not work using only the length of RED_s (this depends on the diagrams), then stronger claims with different measures are proven. The proofs of $\mathbb{C}_{\Leftarrow}(P)$ ($\mathbb{D}_{\Leftarrow}(P)$, resp.) are by induction on the length of RED_t with specialised claims, if this measure does not work. The induction step uses a complete set of commuting diagrams for P .

For a transformation P we will tag a lemma corresponding to the property $\mathbb{C}_{\Rightarrow}(P)$, $\mathbb{C}_{\Leftarrow}(P)$, $\mathbb{D}_{\Leftarrow}(P)$ or $\mathbb{D}_{\Rightarrow}(P)$ if the validity of this property is a direct consequence of the lemma, i.e. if we prove a stronger claim or the validity of the property follows from Lemma 5.12.

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 $A_{\text{amb}}^{\text{let}}$ 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:

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

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

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

Fig. 5. Additional Transformation Rules

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)**.

6.1 Diagrams for **(gc)**

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

$$\begin{array}{ccc}
 \begin{array}{ccc} \cdot & \xrightarrow{S, \mathbf{gc}} & \cdot \\ \text{no}, a \downarrow & & \downarrow \text{no}, a \\ \cdot & \xrightarrow{S, \mathbf{gc}} & \cdot \end{array} &
 \begin{array}{ccc} \cdot & \xrightarrow{S, \mathbf{gc}} & \cdot \\ \text{no}, a \downarrow & \searrow & \cdot \\ & & \cdot \end{array} &
 \begin{array}{ccc} \cdot & \xrightarrow{S, \mathbf{gc}} & \cdot \\ \text{no}, \text{lll} \downarrow & \searrow & \cdot \\ & & \cdot \end{array} \\
 a \text{ arbitrary} & \text{for } a \in \{\text{case}, \text{seq}, \text{amb}\} & a \text{ arbitrary}
 \end{array}$$

Proof. This follows by inspecting all cases where an (S, \mathbf{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, \mathbf{gc}) -transformation is discarded by an **(no, case)**-, **(no, seq)**- or **(no, amb)**-reduction then the second diagram is applicable. The last diagram covers the cases where an **(no, lll)** redex is eliminated by an (S, \mathbf{gc}) -transformation, e.g.

$$\begin{array}{ccc}
 \text{letrec } x = (\lambda y. y) \text{ in } ((\text{letrec } z = (\lambda z'. z') \text{ in } (z z))) & \xrightarrow{S, \mathbf{gc}} & (\text{letrec } z = (\lambda z'. z') \text{ in } (z z)) \\
 \downarrow \text{no, llet-in} & \nearrow S, \mathbf{gc} & \\
 (\text{letrec } x = (\lambda y. y), z = (\lambda z'. z') \text{ in } (z z)) & &
 \end{array}$$

□

6.2 Diagrams for **(cpx)**

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

$$\begin{array}{ccc}
 \begin{array}{ccc} \cdot & \xrightarrow{S, \mathbf{cpx}} & \cdot \\ \text{no}, a \downarrow & & \downarrow \text{no}, a \\ \cdot & \xrightarrow{S, \mathbf{cpx}} & \cdot \end{array} &
 \begin{array}{ccc} \cdot & \xrightarrow{S, \mathbf{cpx}} & \cdot \\ \text{no}, a \downarrow & \searrow & \cdot \\ & & \cdot \end{array} &
 \begin{array}{ccc} \cdot & \xrightarrow{S, \mathbf{cpx}} & \cdot \\ \text{no}, \text{cp} \downarrow & & \downarrow \text{no}, \text{cp} \\ \cdot & \xrightarrow{S, \mathbf{cpx}} & \cdot \\ \cdot & \xrightarrow{S, \mathbf{cpx}} & \cdot \end{array} \\
 a \text{ arbitrary} & \text{for } a \in \{\text{case}, \text{seq}, \text{amb}, \text{cp}\} &
 \end{array}$$

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

- The first diagram describes the cases, where the reductions can be computed.
- The second diagram is applicable, if the redex or inner redex of the (S, cpx) -transformation is discarded by an (no, case) -, (no, seq) - or (no, amb) -reduction or if the target variable of (S, cpx) and an (no, 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 (no, cp) -reduction. \square

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:*

$$\begin{array}{ccc}
 \begin{array}{ccc} \cdot & \xrightarrow{S, \text{xch}} & \cdot \\ \text{no}, a \downarrow & & \downarrow \text{no}, a \\ \cdot & \xrightarrow{S, \text{xch}} & \cdot \end{array} & & \begin{array}{ccc} \cdot & \xrightarrow{S, \text{xch}} & \cdot \\ \text{no}, a \downarrow & \searrow \text{no}, a & \cdot \\ \cdot & & \cdot \end{array} \\
 a \text{ arbitrary} & & a \in \{\text{case}, \text{seq}, \text{amb}\}
 \end{array}$$

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

6.4 Diagrams for (abs)

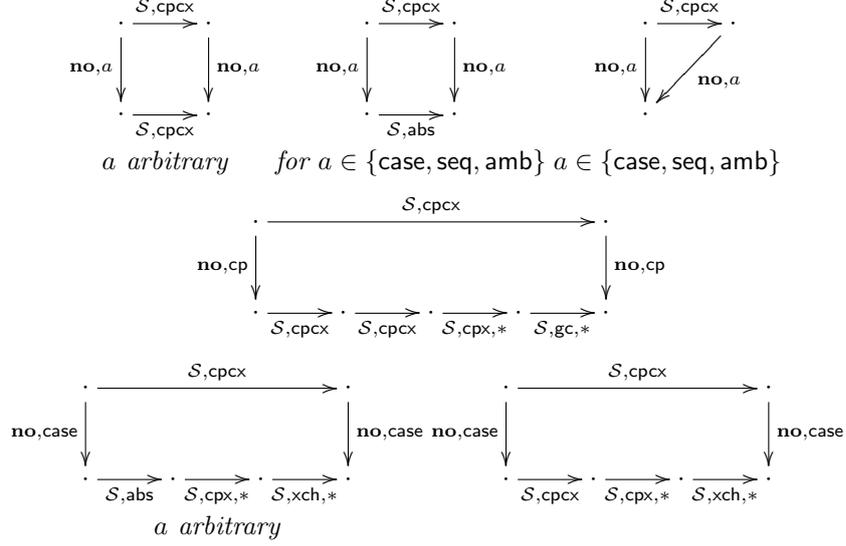
Lemma 6.5. *A complete set of forking and a complete set of commuting diagrams for (S, abs) can be read off the following diagrams:*

$$\begin{array}{ccc}
 \begin{array}{ccc} \cdot & \xrightarrow{S, \text{abs}} & \cdot \\ \text{no}, a \downarrow & & \downarrow \text{no}, a \\ \cdot & \xrightarrow{S, \text{abs}} & \cdot \end{array} & & \begin{array}{ccc} \cdot & \xrightarrow{S, \text{abs}} & \cdot \\ \text{no}, a \downarrow & \searrow \text{no}, a & \cdot \\ \cdot & & \cdot \end{array} & & \begin{array}{ccc} \cdot & \xrightarrow{S, \text{abs}} & \cdot \\ \text{no}, \text{case} \downarrow & & \downarrow \text{no}, \text{case} \\ \cdot & \xrightarrow{S, \text{abs}} \cdot \xrightarrow{S, \text{cpx}, *} \cdot \xrightarrow{S, \text{xch}, *} & \cdot \end{array} \\
 a \text{ arbitrary} & & \text{for } a \in \{\text{case}, \text{seq}, \text{amb}\}
 \end{array}$$

Proof. By inspecting the overlappings between an (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. \square

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 (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 (no, case) -, (no, seq) - or (no, 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 (no, a) -reduction or where the (S, cpcx) -transformation copies a constant into the first argument of a **case**-expression.

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

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

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:*

$$\begin{array}{ccc}
 \begin{array}{ccc} \cdot & \xrightarrow{S, \text{opt}} & \cdot \\ \text{no}, a \downarrow & & \downarrow \text{no}, a \\ \cdot & \xrightarrow{S, \text{opt}} & \cdot \end{array} & \begin{array}{ccc} \cdot & \xrightarrow{S, \text{opt}} & \cdot \\ \text{no}, a \downarrow & \searrow & \cdot \\ \cdot & & \cdot \end{array} & \begin{array}{ccc} \cdot & \xrightarrow{S, \text{opt}} & \cdot \\ \text{no}, \text{lll} \downarrow & \searrow & \cdot \\ \cdot & & \cdot \end{array} & \begin{array}{ccc} \cdot & \xrightarrow{S, \text{opt}} & \cdot \\ \text{no}, a \downarrow & & \downarrow \text{no}, a \\ \cdot & \xrightarrow{S, \text{opt}, +} & \cdot \end{array} \\
 \text{a arbitrary} & \text{for } a \in \{\text{case}, \text{seq}, \text{amb-l}, \text{amb-r}, \text{cp}\} & & \text{for } a \in \{\text{cp}, \text{case}\}
 \end{array}$$

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

Lemma 6.8. *Let $s \xrightarrow{S, \text{opt}} t$ 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 $s \xrightarrow{\text{no}, \text{lllet}} s'$ where s' is a WHNF.

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

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

In all cases an $(\text{no}, \text{lllet})$ -reduction for s leads to a WHNF. \square

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 ($\mathbb{C}_{\Rightarrow}(\text{opt})$). *If $s \xrightarrow{S, \text{opt}} t$, then for all $RED_s \in \mathcal{CON}(s)$ there exists $RED_t \in \mathcal{CON}(t)$ with $\mathbf{rl}(RED_t) \leq \mathbf{rl}(RED_s)$.*

Proof. Let $s \xrightarrow{S, \text{opt}} t$ 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 $\mathbf{rl}(RED_t) \leq l$. 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 $\xleftarrow{RED_s} s \xrightarrow{S, \text{opt}} t$. With RED' being the suffix of RED_s of length $l - 1$, we have the cases:

$$\begin{array}{cccc}
 \begin{array}{ccc} s & \xrightarrow{S, \text{opt}} & t \\ \text{no}, a \downarrow & & \downarrow \text{no}, a \\ s' & \xrightarrow{S, \text{opt}} & t' \\ RED' \downarrow & & \downarrow RED'_t \end{array} & \begin{array}{ccc} s & \xrightarrow{S, \text{opt}} & t \\ \text{no}, a \downarrow & \searrow & \cdot \\ s' & & \cdot \\ RED' \downarrow & & \downarrow \end{array} & \begin{array}{ccc} s & \xrightarrow{S, \text{opt}} & t \\ \text{no}, \text{lll} \downarrow & \searrow & \cdot \\ s' & & \cdot \\ RED' \downarrow & & \downarrow \end{array} & \begin{array}{ccc} s & \xrightarrow{S, \text{opt}} & t \\ \text{no}, a \downarrow & & \downarrow \text{no}, a \\ s' & \xrightarrow{S, \text{opt}, +} & t' \\ RED' \downarrow & & \downarrow RED'_t \end{array} \\
 (1) & (2) & (3) & (4)
 \end{array}$$

- (1) We can apply the induction hypothesis to RED' and hence have $RED'_t \in \mathcal{CON}(t')$ with $\mathbf{rl}(RED'_t) \leq \mathbf{rl}(RED')$. By appending RED'_t to $t \xrightarrow{\mathbf{no},a} t'$ we have $RED_t \in \mathcal{CON}(t)$ with $\mathbf{rl}(RED_t) \leq \mathbf{rl}(RED_s)$.
- (2) There exists $RED_t = \xrightarrow{\mathbf{no},a} \cdot \xrightarrow{RED'}$ and $\mathbf{rl}(RED_t) = l$.
- (3) By the induction hypothesis we have $RED_t \in \mathcal{CON}(t)$ with $\mathbf{rl}(RED_t) \leq \mathbf{rl}(RED')$. With $\mathbf{rl}(RED') = \mathbf{rl}(RED_s) + 1$ 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 $\mathbf{rl}(RED'_t) \leq \mathbf{rl}(RED')$. By appending RED'_t to $t \xrightarrow{\mathbf{no},a} t'$ we have $RED_t \in \mathcal{CON}(t)$ with $\mathbf{rl}(RED_t) \leq l$. \square

Corollary 6.10. *Let $T = \{\text{beta}, \text{lll}, \text{seq}, \text{case}, \text{amb}\}$ then for all $T' \subseteq T$, $RED_s \in \mathcal{CON}(s)$ there exists $RED_t \in \mathcal{CON}(t)$ with $\mathbf{rl}_{T'}(RED_t) \leq \mathbf{rl}_{T'}(RED_s)$.*

Proof. This follows, since through the construction of RED_t the forking diagrams do not modify the kind of a reduction, nor introduce new reductions.

Lemma 6.11 ($\mathcal{C}_{\Leftarrow}(\text{opt})$). *If $s \xrightarrow{S,\text{opt}} t$ then for all $RED_t \in \mathcal{CON}(t)$ there exists $RED_s \in \mathcal{CON}(s)$ with $\mathbf{rl}_{(\setminus \text{lll})}(RED_s) \leq \mathbf{rl}_{(\setminus \text{lll})}(RED_t)$.*

Proof. We use induction on the following measure μ on reduction sequences $s \xrightarrow{S,\text{opt}} t \xrightarrow{RED}$ with $\mu(s \xrightarrow{S,\text{opt}} t \xrightarrow{RED}) = (\mathbf{rl}_{(\setminus \text{lll})}(RED), \mu_{\text{lll}}(s))$ (the measure $\mu_{\text{lll}}(\cdot)$ was introduced in Definition 4.18). We assume the measure to be ordered lexicographically. If $\mu(s \xrightarrow{S,\text{opt}} t \xrightarrow{RED_t}) = (0, (0, 0))$, then from $\mu_{\text{lll}}(s) = (0, 0)$ follows $\mu_{\text{lll}}(t) = (0, 0)$ 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 $s \xrightarrow{\mathbf{no}, \text{lll}} s'$ where s' is a WHNF. In both cases we have a (possibly empty) $RED_s \in \mathcal{CON}(S)$ with $\mathbf{rl}_{(\setminus \text{lll})}(RED_s) \leq \mathbf{rl}_{(\setminus \text{lll})}(RED_t)$.

Now, let $\mu(s \xrightarrow{S,\text{opt}} t \xrightarrow{RED_t}) = (l, m) > (0, (0, 0))$. 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:

$$\begin{array}{cccc}
 \begin{array}{ccc} s & \xrightarrow{S,\text{opt}} & t \\ \text{no},a \downarrow & & \downarrow \text{no},a \\ s' & \xrightarrow{S,\text{opt}} & t' \\ \text{RED}'_s \downarrow & & \downarrow \text{RED}' \end{array} &
 \begin{array}{ccc} s & \xrightarrow{S,\text{opt}} & t \\ \text{no},a \searrow & & \downarrow \text{no},a \\ & & t' \\ \text{RED}' \downarrow & & \end{array} &
 \begin{array}{ccc} s & \xrightarrow{S,\text{opt}} & t \\ \text{no}, \text{lll} \downarrow & & \downarrow \text{RED}_t \\ s' & \xrightarrow{S,\text{opt}} & t' \\ \text{RED}'_s \downarrow & & \downarrow \end{array} &
 \begin{array}{ccc} s & \xrightarrow{S,\text{opt}} & t \\ \text{no},a \downarrow & & \downarrow \text{no},a \\ s' & \xrightarrow{S,\text{opt},+} & t' \\ \text{RED}'_s \downarrow & & \downarrow \text{RED}' \end{array} \\
 (1) & (2) & (3) & (4)
 \end{array}$$

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

- If the (\mathbf{no}, a) -reduction is not an $(\mathbf{no}, \text{lll})$ -reduction, then $\mathbf{rl}_{(\setminus \text{lll})}(RED') < \mathbf{rl}_{(\setminus \text{lll})}(RED_t)$ and we can apply the induction hypothesis to $s' \xrightarrow{S, \text{opt}} t' \xrightarrow{RED'} \text{and have } RED'_s \in \mathcal{CON}(s')$ with $\mathbf{rl}_{(\setminus \text{lll})}(RED'_s) \leq \mathbf{rl}_{(\setminus \text{lll})}(RED')$. Again, we append RED'_s to $s \xrightarrow{\mathbf{no}, a} s'$ and have a terminating normal order reduction RED_s for s with $\mathbf{rl}_{(\setminus \text{lll})}(RED_s) \leq \mathbf{rl}_{(\setminus \text{lll})}(RED_t)$.
- (2) We have $RED_s \xrightarrow{\mathbf{no}, a, RED'} \text{and } \mathbf{rl}_{(\setminus \text{lll})}(RED_s) = \mathbf{rl}_{(\setminus \text{lll})}(RED_t)$.
- (3) Since $\mu_{\text{lll}}(s') < \mu_{\text{lll}}(s)$, we can apply the induction hypothesis to $s' \xrightarrow{S, \text{opt}} t \xrightarrow{RED_t}$ and have $RED'_s \in \mathcal{CON}(s')$ with $\mathbf{rl}_{(\setminus \text{lll})}(RED'_s) \leq \mathbf{rl}_{(\setminus \text{lll})}(RED_t)$.
By appending RED'_s to $s \xrightarrow{\mathbf{no}, \text{lll}} s'$ the claim follows.
- (4) Since $\mathbf{rl}_{(\setminus \text{lll})}(RED') < \mathbf{rl}_{(\setminus \text{lll})}(RED_t)$, we can apply the induction hypothesis multiple times for every (S, opt) -transformation leading to $RED'_s \in \mathcal{CON}(s')$ with $\mathbf{rl}_{(\setminus \text{lll})}(RED'_s) \leq l - 1$. By appending RED'_s to $s \xrightarrow{\mathbf{no}, a} s'$ we have $RED_s \in \mathcal{CON}(s)$ with $\mathbf{rl}_{(\setminus \text{lll})}(RED_s) \leq l$. \square

Since we have shown $\mathbb{C}_{\Leftarrow}(\text{opt})$ and $\mathbb{C}_{\Rightarrow}(\text{opt})$, Lemma 5.11 can be applied:

Corollary 6.12. *If $s \xrightarrow{S, \text{opt}} t$ then $s \uparrow$ iff $t \uparrow$.*

Lemma 6.13 ($\mathbb{D}_{\Rightarrow}(\text{opt})$). *If $s \xrightarrow{S, \text{opt}} t$, then for all $RED_s \in \mathcal{DIV}(s)$ there exists $RED_t \in \mathcal{DIV}(t)$ with $\mathbf{rl}(RED_t) \leq \mathbf{rl}(RED_s)$*

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

Lemma 6.14 ($\mathbb{D}_{\Leftarrow}(\text{opt})$). *If $s \xrightarrow{S, \text{opt}} t$ then for all $RED_t \in \mathcal{DIV}(t)$ there exists $RED_s \in \mathcal{DIV}(s)$ with $\mathbf{rl}_{(\setminus \text{lll})}(RED_s) \leq \mathbf{rl}_{(\setminus \text{lll})}(RED_t)$.*

Proof. The claim follows by induction on the measure μ on reduction sequences $s \xrightarrow{S, \text{opt}} t \xrightarrow{RED}$ with $\mu(s \xrightarrow{S, \text{opt}} t \xrightarrow{RED}) = (\mathbf{rl}_{(\setminus \text{lll})}(RED), \mu_{\text{lll}}(s))$. Let the measure be ordered lexicographically. The base case is covered by Corollary 6.12. I.e., let $s = S[s_0], t = S[t_0]$ and $s_0 \xrightarrow{\text{opt}} t_0$, then we have $\mu(s \xrightarrow{S, \text{opt}} t \xrightarrow{RED_t}) = (0, (0, 0))$. Since $\mu_{\text{lll}}(s) = (0, 0)$ we have that $\mu_{\text{lll}}(t) = (0, 0)$ since an (S, opt) transformation does not introduce new **letrec**-expressions. Thus $\mathbf{rl}(RED_t) = 0$, i.e. $t \uparrow$ and Corollary 6.12 shows the claim. The induction step uses the same arguments as the proof of Lemma 6.11. \square

The previous lemmas show that (opt) fulfills the \mathbb{CD} -properties, hence with Theorem 5.10 the following proposition follows:

Proposition 6.15. *(opt) is a correct program transformation.*

7 Correctness of Deterministic Reduction Rules

In this section we prove the correctness of the remaining reduction rules of $\Lambda_{\text{amb}}^{\text{let}}$.

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 $s \xrightarrow{\text{case}} t$ then $s \sim_c t$.

Proof. From Propositions 6.15 and 5.8 we have that (cpx), (cpcx) and (case-c) preserve contextual equivalence. Let $\{x_i = x_{i-1}\}_{i=1}^m$ be the chain which is used by a $(\mathcal{C}, \text{case-in})$ - or $(\mathcal{C}, \text{case-e})$ -reduction, then every (case-in)-reduction can be replaced by the sequence $\xrightarrow{\mathcal{C}, \text{cpx}, m-1} \xrightarrow{\mathcal{C}, \text{cpcx}} \xrightarrow{\mathcal{C}, \text{case-c}}$:

$$\begin{array}{l} \text{letrec } x_1 = c \vec{t}_i, \{x_i = x_{i-1}\}_{i=2}^m, Env \text{ in } C[\text{case}_T x_m \dots (c \vec{z}_i \rightarrow r) \dots] \\ \xrightarrow{\text{cpx}, m-1} \text{letrec } x_1 = c \vec{t}_i, \{x_i = x_{i-1}\}_{i=2}^m, Env \text{ in } C[\text{case}_T x_1 \dots (c \vec{z}_i \rightarrow r) \dots] \\ \xrightarrow{\text{cpcx}} \text{letrec } x_1 = c \vec{y}_i, \{y_i = t_i\}_{i=1}^{\text{ar}(c)}, \{x_i = x_{i-1}\}_{i=2}^m, Env \text{ in } C[\text{case}_T (c \vec{y}_i) \dots] \\ \xrightarrow{\text{case-c}} \text{letrec } x_1 = c \vec{y}_i, \{y_i = t_i\}_{i=1}^{\text{ar}(c)}, \{x_i = x_{i-1}\}_{i=2}^m, Env \\ \text{in } C[\text{letrec } \{z_i = y_i\}_{i=1}^{\text{ar}(c)} \text{ in } r] \end{array}$$

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

7.2 Correctness of (III)

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 (III) and finally prove the correctness of (III)

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

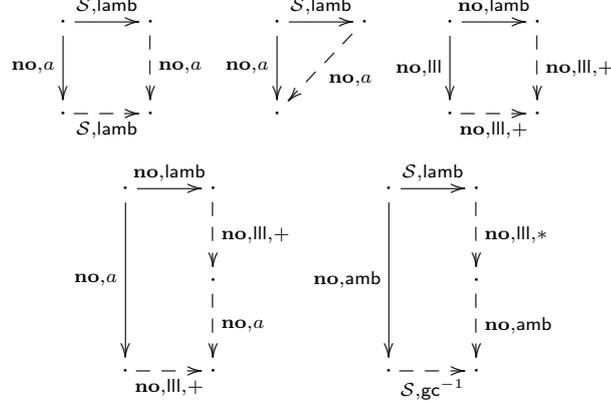
Lemma 7.2. Let $\text{red} \in \{\text{lapp}, \text{lcase}, \text{lseq}\}$, a complete set of forking diagrams for $\xrightarrow{S, \text{red}}$ is

$$\begin{array}{ccc} \begin{array}{ccc} \cdot & \xrightarrow{S, \text{red}} & \cdot \\ \text{no}, a \downarrow & & \downarrow \text{no}, a \\ \cdot & \xrightarrow{S, \text{red}} & \cdot \end{array} & & \begin{array}{ccc} \cdot & \xrightarrow{S, \text{red}} & \cdot \\ \text{no}, a \downarrow & & \downarrow \text{no}, a \\ \cdot & \xrightarrow{S, \text{red}} & \cdot \end{array} \\ a \text{ arbitrary} & & \text{for } a \in \{\text{case}, \text{seq}, \text{amb}\} \end{array}$$

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

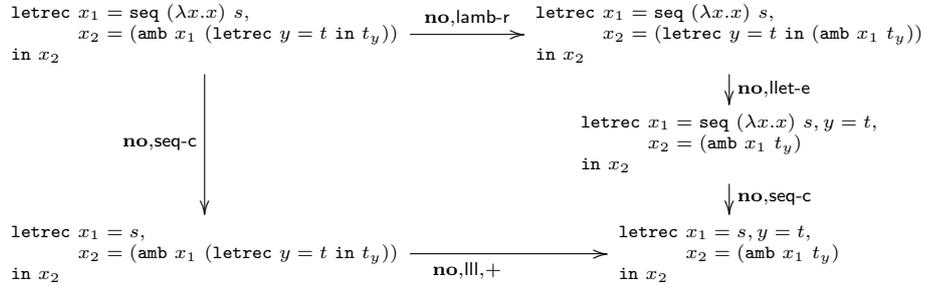
7.2.2 Diagrams for (lamb)

Lemma 7.3. *A complete set of forking diagrams for (S, lamb) is*

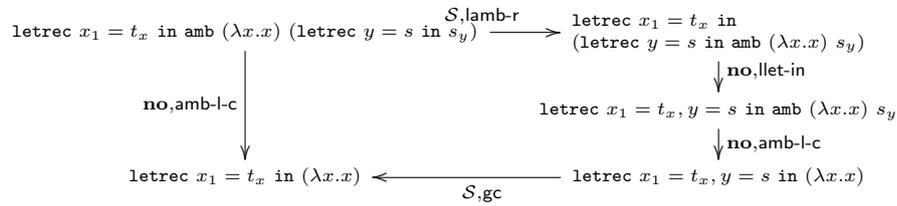


where for the first diagram a is arbitrary, for second diagram $a \in \{\text{case}, \text{seq}, \text{amb}\}$ and for the 4th diagram $a \in \{\text{case}, \text{lbeta}, \text{cp}, \text{seq}, \text{amb}\}$.

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 (no, amb) -reduction are identical, e.g.



□

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

$$\begin{array}{ccc}
 \begin{array}{ccc} \cdot & \xrightarrow{iS, red} & \cdot \\ \text{no}, a \downarrow & & \downarrow \text{no}, a \\ \cdot & \xrightarrow{iS, red} & \cdot \end{array} & & \begin{array}{ccc} \cdot & \xrightarrow{iS, red} & \cdot \\ & \searrow \text{no}, a & \downarrow \text{no}, a \\ & & \cdot \end{array} \\
 a \text{ arbitrary} & \text{for } a \in \{\text{case}, \text{seq}, \text{amb}\} &
 \end{array}$$

Proof. This follows by checking all cases where (iS, a) with $a \in \{\text{lapp}, \text{lseq}, \text{lcase}, \text{lamb}\}$ is followed by a normal order reduction. The first diagram covers the cases where the reductions commute. If the normal order reduction is a (case)–, (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 (S, llet) is*

$$\begin{array}{ccc}
 \begin{array}{ccc} \cdot & \xrightarrow{S, \text{llet}} & \cdot \\ \text{no}, a \downarrow & & \downarrow \text{no}, a \\ \cdot & \xrightarrow{S, \text{llet}} & \cdot \end{array} & & \begin{array}{ccc} \cdot & \xrightarrow{S, \text{llet}} & \cdot \\ & \searrow \text{no}, a & \downarrow \text{no}, a \\ & & \cdot \end{array} & & \begin{array}{ccc} \cdot & \xrightarrow{S, \text{llet}} & \cdot \\ \text{no}, a \downarrow & & \downarrow \text{no}, \text{lll}, + \\ \cdot & \xrightarrow{S, \text{llet}} & \cdot \end{array} \\
 a \text{ arbitrary} & \text{for } a \in \{\text{case}, \text{seq}, \text{amb}\} & \text{for } a \in \{\text{lapp}, \text{lcase}, \text{lseq}, \text{lamb}\} & &
 \end{array}$$

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 (**no, case**)–, (**no, seq**)– or (**no, 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 (**no, a**) with $a \in \{\text{lapp}, \text{lcase}, \text{lseq}, \text{lamb}\}$ is no longer in a reduction context. An example for this case is:

$$\begin{array}{ccc}
 \text{letrec } E_1 \text{ in} & & \\
 ((\text{letrec } E_2 \text{ in } (\text{letrec } E_3 \text{ in } r)) s) & \xrightarrow{S, \text{llet-in}} & \text{letrec } E_1 \text{ in } ((\text{letrec } E_2, E_3 \text{ in } r) s) \\
 \text{no}, \text{lapp} \downarrow & & \downarrow \text{no}, \text{lll}, + \\
 \text{letrec } E_1 \text{ in} & & \\
 (\text{letrec } E_2 \text{ in } ((\text{letrec } E_3 \text{ in } r) s)) & \xrightarrow{\text{no}, \text{lll}, +} & \text{letrec } E_1, E_2, E_3 \text{ in } (r s)
 \end{array}$$

□

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

$$\begin{array}{cccc}
 \begin{array}{ccc} \cdot & \xrightarrow{i\mathcal{S}, \text{llet}} & \cdot \\ \text{no}, a \downarrow & & \downarrow \text{no}, a \\ \cdot & \xrightarrow{i\mathcal{S}, \text{llet}} & \cdot \end{array} &
 \begin{array}{ccc} \cdot & \xrightarrow{i\mathcal{S}, \text{llet}} & \cdot \\ \text{no}, a \searrow & & \downarrow \text{no}, a \\ \cdot & & \cdot \end{array} &
 \begin{array}{ccc} \cdot & \xrightarrow{i\mathcal{S}, \text{llet}} & \cdot \\ \text{no}, \text{lll}, + \searrow & & \downarrow \text{no}, \text{lll} \\ \cdot & & \cdot \end{array} &
 \begin{array}{ccc} \cdot & \xrightarrow{i\mathcal{S}, \text{llet}} & \cdot \\ \text{no}, \text{lll} \downarrow & & \downarrow \text{no}, \text{lll} \\ \cdot & \xrightarrow{i\mathcal{S}, \text{lll}, +} & \cdot \end{array} \\
 a \text{ arbitrary} & \text{for } a \in \{\text{case}, \text{seq}, \text{amb}\} & &
 \end{array}$$

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 (no, case)-, (no, seq)- or (no, amb)-reduction, then the second diagram is applicable. The 4th diagram covers the cases where an (no, lll)-reduction overlaps with the ($i\mathcal{S}, \text{llet}$)-reduction and the letrec -environment needs to be adjusted.

$$\begin{array}{ccc}
 R_0[R'_{\#1}[(\text{letrec } Env \text{ in } (\text{letrec } Env' \text{ in } r'))]] & \xrightarrow{\mathcal{S}, \text{llet-in}} & R_0[R'_{\#1}[(\text{letrec } Env, Env' \text{ in } r')]] \\
 \text{no}, a \downarrow & & \downarrow \text{no}, a \\
 R_0[(\text{letrec } Env \text{ in } R'_{\#1}[(\text{letrec } Env' \text{ in } r'))]] & \xrightarrow{i\mathcal{S}, \text{lll}, +} & R_0[(\text{letrec } Env, Env' \text{ in } R'_{\#1}[r'])]
 \end{array}$$

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

$$\begin{array}{ccc}
 R'_{\#1}[(\text{letrec } Env \text{ in } (\text{letrec } Env' \text{ in } r'))]] & \xrightarrow{\mathcal{S}, \text{llet-in}} & R'_{\#1}[(\text{letrec } Env, Env' \text{ in } r')] \\
 \text{no}, a \downarrow & & \swarrow \text{no}, a \\
 (\text{letrec } Env \text{ in } R'_{\#1}[(\text{letrec } Env' \text{ in } r'))]] & & \\
 \text{no}, \text{lll}, + \downarrow & & \swarrow \text{no}, a \\
 (\text{letrec } Env, Env' \text{ in } R'_{\#1}[r']) & &
 \end{array}$$

□

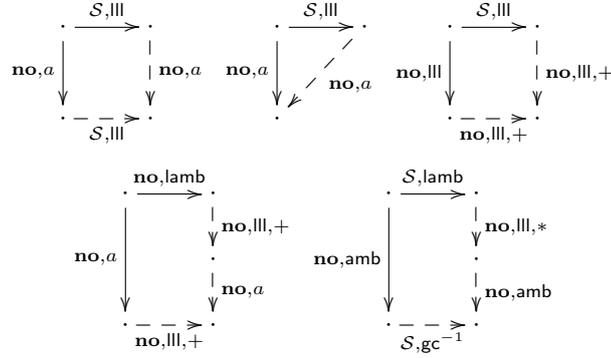
7.2.4 Proving Correctness of (lll) Now we can combine the diagrams to derive complete sets of forking and commuting diagrams for (lll).

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

$$\begin{array}{cccc}
 \begin{array}{ccc} \cdot & \xrightarrow{i\mathcal{S}, \text{lll}} & \cdot \\ \text{no}, a \downarrow & & \downarrow \text{no}, a \\ \cdot & \xrightarrow{i\mathcal{S}, \text{lll}} & \cdot \end{array} &
 \begin{array}{ccc} \cdot & \xrightarrow{i\mathcal{S}, \text{lll}} & \cdot \\ \text{no}, a \searrow & & \downarrow \text{no}, a \\ \cdot & & \cdot \end{array} &
 \begin{array}{ccc} \cdot & \xrightarrow{i\mathcal{S}, \text{llet}} & \cdot \\ \text{no}, \text{lll}, + \searrow & & \downarrow \text{no}, \text{lll} \\ \cdot & & \cdot \end{array} &
 \begin{array}{ccc} \cdot & \xrightarrow{i\mathcal{S}, \text{llet}} & \cdot \\ \text{no}, \text{lll} \downarrow & & \downarrow \text{no}, \text{lll} \\ \cdot & \xrightarrow{i\mathcal{S}, \text{lll}, +} & \cdot \end{array} \\
 a \text{ arbitrary} & \text{for } a \in \{\text{case}, \text{seq}, \text{amb-l}, \text{amb-r}\} & &
 \end{array}$$

Proof. Follows from Lemma 7.4 and Lemma 7.6. □

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



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

Proof. Follows from Lemma 7.2, Lemma 7.3 and Lemma 7.5. \square

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

Lemma 7.10 ($\mathbb{C}_{\neq}(\text{III})$). *If $s \xrightarrow{iS, \text{III}} t$ then for all $RED_t \in CON(t)$ there exists $RED_s \in CON(s)$ with $\text{rl}_{(\setminus \text{III})}(RED_s) \leq \text{rl}_{(\setminus \text{III})}(RED_t)$.*

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

$$\begin{array}{cccc}
 \begin{array}{ccc}
 s \xrightarrow{iS, \text{III}} t & & \\
 \text{no}, a \downarrow & & \downarrow \text{no}, a \\
 s' \xrightarrow{iS, \text{III}} t' & & \\
 \text{RED}'' \downarrow & & \downarrow \text{RED}'
 \end{array} &
 \begin{array}{ccc}
 s \xrightarrow{iS, \text{III}} t & & \\
 \text{no}, a \searrow & & \downarrow \text{no}, a \\
 t' & & \\
 \downarrow \text{RED}' & &
 \end{array} &
 \begin{array}{ccc}
 s \xrightarrow{iS, \text{IIlet}} t & & \\
 \text{no}, \text{III}, + \searrow & & \downarrow \text{no}, \text{III} \\
 t' & & \\
 \downarrow \text{RED}' & &
 \end{array} &
 \begin{array}{ccc}
 s \xrightarrow{iS, \text{IIlet}} t & & \\
 \text{no}, \text{III} \downarrow & & \downarrow \text{no}, \text{III} \\
 s' \xrightarrow{iS, \text{III}, +} t' & & \\
 \text{RED}'' \downarrow & & \downarrow \text{RED}'
 \end{array} \\
 (1) & (2) & (3) & (4)
 \end{array}$$

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

- (1) If the (no, a) -reduction is an (no, III) -reduction then we can apply the induction hypothesis to the sequence $s' \xrightarrow{iS, \text{III}} t' \xrightarrow{RED'} r$ since from $s \xrightarrow{\text{no}, \text{III}} s'$

follows that $\mu_{ll}(s') < \mu_{ll}(s)$ and $\mathbf{rl}_{(\setminus \text{ll})}(RED') = \mathbf{rl}_{(\setminus \text{ll})}(RED_t)$. Hence, we have $RED'' \in \mathcal{CON}(s')$ with $\mathbf{rl}_{(\setminus \text{ll})}(RED'') \leq \mathbf{rl}_{(\setminus \text{ll})}(RED')$. By appending RED'' to $s \xrightarrow{\mathbf{no}, \text{ll}} s'$ we have $RED_s \in \mathcal{CON}(s)$ with $\mathbf{rl}_{(\setminus \text{ll})}(RED_s) = \mathbf{rl}_{(\setminus \text{ll})}(RED'')$.

If the (\mathbf{no}, a) -reduction is not an (\mathbf{no}, ll) -reduction, then we also can apply the induction hypothesis to the sequence $s' \xrightarrow{iS, \text{ll}} t' \xrightarrow{RED'}$ since $\mathbf{rl}_{(\setminus \text{ll})}(RED) > \mathbf{rl}_{(\setminus \text{ll})}(RED')$. Hence we have $RED'' \in \mathcal{CON}(s')$ with $\mathbf{rl}_{(\setminus \text{ll})}(RED'') \leq \mathbf{rl}_{(\setminus \text{ll})}(RED')$. By appending RED'' to $s \xrightarrow{\mathbf{no}, a} s'$ we have $RED_s \in \mathcal{CON}(s)$ with $\mathbf{rl}_{(\setminus \text{ll})}(RED_s) \leq \mathbf{rl}_{(\setminus \text{ll})}(RED_t)$.

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

$$\begin{array}{c}
 s \xrightarrow{iS, \text{ll}et} t \\
 \begin{array}{ccc}
 \mathbf{no}, \text{ll} \downarrow & & \downarrow \mathbf{no}, \text{ll} \\
 s' \xrightarrow{iS, \text{ll}} s'_1 \xrightarrow{iS, \text{ll}} \dots \xrightarrow{iS, \text{ll}} s'_{k-1} \xrightarrow{iS, \text{ll}} t' \\
 \left. \begin{array}{c} \text{RED}'' \\ \downarrow \end{array} \right\} & \left. \begin{array}{c} \text{RED}'_1 \\ \downarrow \end{array} \right\} & \left. \begin{array}{c} \text{RED}''_{k-1} \\ \downarrow \end{array} \right\} \text{RED}' \\
 \end{array}
 \end{array}$$

It holds that $\mu_{ll}(s) > \mu_{ll}(s') > \mu_{ll}(s'_1) > \dots > \mu_{ll}(s'_{k-1}) > \mu_{ll}(t')$. Since $\mathbf{rl}_{(\setminus \text{ll})}(RED_t) = \mathbf{rl}_{(\setminus \text{ll})}(RED')$, we can apply the induction hypothesis to the sequence $s'_{k-1} \xrightarrow{iS, \text{ll}} t' \xrightarrow{RED'}$ and have $RED''_{k-1} \in \mathcal{CON}(s'_{k-1})$ with $\mathbf{rl}_{(\setminus \text{ll})}(RED''_{k-1}) \leq \mathbf{rl}_{(\setminus \text{ll})}(RED')$. 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 $\mathbf{rl}_{(\setminus \text{ll})}(RED'') \leq \mathbf{rl}_{(\setminus \text{ll})}(RED')$. By appending RED'' to $s \xrightarrow{\mathbf{no}, \text{ll}} s'$, we have $RED_s \in \mathcal{CON}(s)$ with $\mathbf{rl}_{(\setminus \text{ll})}(RED_s) \leq \mathbf{rl}_{(\setminus \text{ll})}(RED)$. \square

Lemma 7.11 ($\mathbb{C}_{\Rightarrow}(\text{ll})$). *If $s \xrightarrow{S, \text{ll}} t$ then for all $RED_s \in \mathcal{CON}(s)$ there exists $RED_t \in \mathcal{CON}(t)$ with $\mathbf{rl}_{(\setminus \text{ll})}(RED_t) \leq \mathbf{rl}_{(\setminus \text{ll})}(RED_s)$.*

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

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

For the induction step, we use the complete set of forking diagrams for (S, ll) 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, III) -reduction. By dropping the first reduction from RED_s we have $RED_t \in \mathcal{CON}(t)$ with $\mathbf{rl}_{(\setminus \text{III})}(RED_t) = l$.

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

$$\begin{array}{ccc}
 \begin{array}{c} s \xrightarrow{S, \text{III}} t \\ \text{no}, a \downarrow \quad \downarrow \text{no}, a \\ s' \xrightarrow{S, \text{III}} t' \\ RED' \downarrow \quad \downarrow \\ RED' \downarrow \quad \downarrow \\ (1) \end{array} &
 \begin{array}{c} s \xrightarrow{S, \text{III}} t \\ \text{no}, a \downarrow \quad \swarrow \text{no}, a \\ s' \xrightarrow{S, \text{III}} t' \\ RED' \downarrow \quad \downarrow \\ RED' \downarrow \quad \downarrow \\ (2) \end{array} &
 \begin{array}{c} s \xrightarrow{S, \text{III}} t \\ \text{no}, \text{III} \downarrow \quad \downarrow \text{no}, \text{III}, + \\ s' \xrightarrow{\text{no}, \text{III}, +} t' \\ RED' \downarrow \quad \downarrow \\ RED' \downarrow \quad \downarrow \\ (3) \end{array} \\
 \\
 \begin{array}{c} \text{no}, \text{lamb} \\ s \xrightarrow{\quad} t \\ \text{no}, a \downarrow \quad \downarrow \text{no}, \text{III}, + \\ s' \xrightarrow{\quad} t' \\ RED' \downarrow \quad \downarrow \\ RED' \downarrow \quad \downarrow \\ (4) \end{array} &
 \begin{array}{c} S, \text{lamb} \\ s \xrightarrow{\quad} t \\ \text{no}, \text{amb} \downarrow \quad \downarrow \text{no}, \text{III}, * \\ s' \xrightarrow{S, \text{gc}} t' \\ RED' \downarrow \quad \downarrow \\ RED' \downarrow \quad \downarrow \\ (5) \end{array}
 \end{array}$$

- For the first diagram we split into two cases:
 - The reduction $s \xrightarrow{\text{no}, a} s'$ is not an (no, III) -reduction. Since $\mathbf{rl}_{(\setminus \text{III})}(RED') = 1 + \mathbf{rl}_{(\setminus \text{III})}(RED_s)$, we can apply the induction hypothesis, i.e. there exists $RED'' \in \mathcal{CON}(t')$ with $\mathbf{rl}_{(\setminus \text{III})}(RED'') \leq \mathbf{rl}_{(\setminus \text{III})}(RED')$. By appending RED'' to $t \xrightarrow{\text{no}, a} t'$ we have $RED_t \in \mathcal{CON}(t)$ with $\mathbf{rl}_{(\setminus \text{III})}(RED_t) \leq \mathbf{rl}_{(\setminus \text{III})}(RED_s)$.
 - The reduction $s \xrightarrow{\text{no}, a} s'$ is an (no, III) -reduction. Since $\mathbf{rl}_{(\setminus \text{III})}(RED_s) = \mathbf{rl}_{(\setminus \text{III})}(RED')$ and $\mu_{\text{III}}(s) > \mu_{\text{III}}(s')$ we can apply the induction hypothesis and have a reduction sequence $RED'' \in \mathcal{CON}(t')$ with $\mathbf{rl}_{(\setminus \text{III})}(RED'') \leq \mathbf{rl}_{(\setminus \text{III})}(RED')$. By appending RED'' to $t \xrightarrow{\text{no}, a} t'$ we have $RED_t \in \mathcal{CON}(t)$ with $\mathbf{rl}_{(\setminus \text{III})}(RED_t) \leq \mathbf{rl}_{(\setminus \text{III})}(RED_s)$.
- For the second diagram the existence of $RED_t \in \mathcal{CON}(t)$ with $\mathbf{rl}_{(\setminus \text{III})}(RED_t) \leq \mathbf{rl}_{(\setminus \text{III})}(RED_s)$ is obvious.
- Since $\mathbf{rl}_{(\setminus \text{III})}(RED) = \mathbf{rl}_{(\setminus \text{III})}(RED')$ but $\mu_{\text{III}}(s') < \mu_{\text{III}}(s)$ and (III) decreases the measure μ_{III} strictly, we can apply the induction hypothesis multiple times for every (no, III) -reduction in the sequence $s' \xrightarrow{\text{no}, \text{III}, +} t'$. Thus, we have $RED'' \in \mathcal{CON}(t')$ with $\mathbf{rl}_{(\setminus \text{III})}(RED'') \leq \mathbf{rl}_{(\setminus \text{III})}(RED')$. By appending RED'' to $t \xrightarrow{\text{no}, \text{III}, +} t'$ we have $RED_t \in \mathcal{CON}(t)$ with $\mathbf{rl}_{(\setminus \text{III})}(RED_t) \leq l$.
- Since the first reduction of RED_s is not an (III) -reduction, $\mathbf{rl}_{(\setminus \text{III})}(RED') = l - 1$. Hence, we can use the induction hypothesis multiple times for every (III) -reduction in $s' \xrightarrow{\text{no}, \text{III}, +} t'$ and derive $RED'' \in \mathcal{CON}(t')$ with $\mathbf{rl}_{(\setminus \text{III})}(RED'') \leq l - 1$. By appending RED'' to $t \xrightarrow{\text{no}, \text{III}, +} t' \xrightarrow{\text{no}, a} t''$ we have $RED_t \in \mathcal{CON}(t)$ with $\mathbf{rl}_{(\setminus \text{III})}(RED_t) \leq l$.

- (5) We can construct the reduction sequence $t \xrightarrow{\mathbf{no}, \text{III}, *}$ $t' \xrightarrow{\mathbf{no}, \text{amb}}$ $t'' \xrightarrow{S, \text{gc}}$ $s' \xrightarrow{RED'}$ where $RED' \in \mathcal{CON}(s')$. From Lemma 6.11 we have that there exists $RED'' \in \mathcal{CON}(t'')$ with $\mathbf{rl}_{(\setminus \text{III})}(RED'') \leq \mathbf{rl}_{(\setminus \text{III})}(RED')$. Hence, we have the sequence $RED_t = t \xrightarrow{\mathbf{no}, \text{III}, *}$ $t' \xrightarrow{\mathbf{no}, \text{amb}}$ $t'' \xrightarrow{RED''}$ with $\mathbf{rl}_{(\setminus \text{III})}(RED_t) \leq \mathbf{rl}_{(\setminus \text{III})}(RED_s)$. \square

Corollary 7.12. *If $s \xrightarrow{S, \text{III}} t$ then $s \uparrow$ iff $t \uparrow$.*

Lemma 7.13. *If $s \xrightarrow{iS, \text{III}} t$ then for all $RED_t \in \mathcal{DIV}(t)$ there exists $RED_s \in \mathcal{DIV}(s)$ with $\mathbf{rl}_{(\setminus \text{III})}(RED_s) \leq \mathbf{rl}_{(\setminus \text{III})}(RED_t)$.*

Proof. Let μ be a measure on reduction sequences $s \xrightarrow{iS, \text{III}} t \xrightarrow{RED} r$ with $\mu(s \xrightarrow{iS, \text{III}} t \xrightarrow{RED} r) = (\mathbf{rl}_{(\setminus \text{III})}(RED), \mu_{III}(s))$. We use induction on the measure μ , ordered lexicographically. $\mu_{III}(s)$ cannot be smaller than $(1, 1)$, since otherwise no (iS, III) -reduction would be possible. If $\mu(s \xrightarrow{iS, \text{III}} t \xrightarrow{RED} r) = (0, (1, 1))$ then $\mu_{III}(t) < (1, 1)$, hence the RED cannot contain $(\mathbf{no}, \text{III})$ -reductions and thus RED is empty, i.e. $t \uparrow$. Then Corollary 7.12 shows the claim. The induction step is analogous as in the proof of Lemma 7.10. \square

Lemma 7.14. *If $s \xrightarrow{S, \text{III}} t$ then for all $RED_s \in \mathcal{DIV}(s)$ there exists $RED_t \in \mathcal{DIV}(t)$ with $\mathbf{rl}_{(\setminus \text{III})}(RED_t) \leq \mathbf{rl}_{(\setminus \text{III})}(RED_s)$.*

Proof. This follows by induction on a lexicographically ordered measure μ defined as $\mu(\xleftarrow{RED_s} s \xrightarrow{S, \text{III}} t) = (\mathbf{rl}_{(\setminus \text{III})}(RED_s), \mu_{III}(s))$. The base case follows from Corollary 7.12, and the induction uses the forking diagrams for (S, III) . \square

Since we have shown the laws $\mathbb{C} \Rightarrow (\text{III})$, $\mathbb{C} \Leftarrow (\text{III})$, $\mathbb{D} \Rightarrow (\text{III})$ and $\mathbb{D} \Leftarrow (\text{III})$ the transformation (III) fulfills the \mathbb{CD} -properties. Thus the following proposition holds.

Proposition 7.15. *(III) is a correct program transformation.*

7.3 Correctness of (seq)

Lemma 7.16. *A complete set of forking diagrams for (S, seq) is*

$$\begin{array}{ccc}
 \begin{array}{ccc} \cdot & \xrightarrow{S, \text{seq}} & \cdot \\ \text{no}, a \downarrow & & \downarrow \text{no}, a \\ \cdot & \xrightarrow{\quad} & \cdot \\ & S, \text{seq} & \end{array} &
 \begin{array}{ccc} \cdot & \xrightarrow{S, \text{seq}} & \cdot \\ \text{no}, a \downarrow & \swarrow & \downarrow \text{no}, a \\ \cdot & & \cdot \\ & \text{no}, a & \end{array} &
 \begin{array}{ccc} \cdot & \xrightarrow{S, \text{seq}} & \cdot \\ \text{no}, \text{cp} \downarrow & & \downarrow \text{no}, \text{cp} \\ \cdot & \xrightarrow{\quad} & \cdot \\ & S, \text{seq} \quad S, \text{seq} & \end{array}
 \end{array}$$

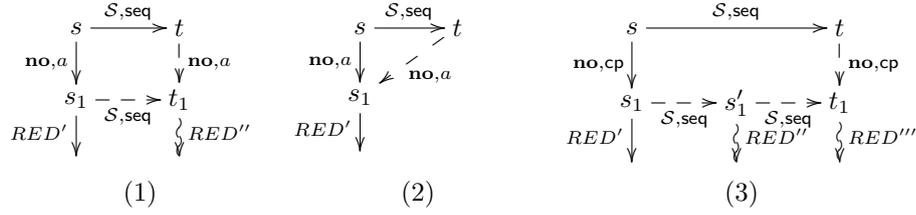
a arbitrary for $a \in \{\text{case}, \text{seq}, \text{amb-l}, \text{amb-r}\}$

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}, cp) -reduction, then two (S, seq) -reductions are necessary. \square

Lemma 7.17. *If $s \xrightarrow{iS, \text{seq}} t$ then s is a WHNF iff t is a WHNF.*

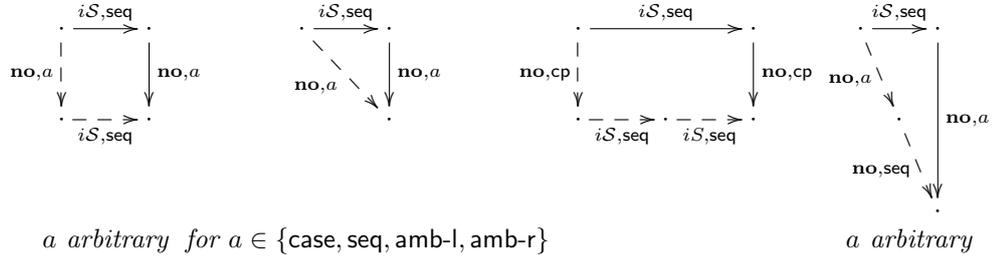
Lemma 7.18 ($\mathbb{C}_{\Rightarrow}(\text{seq})$). *If $s \xrightarrow{S, \text{seq}} t$ then for all $RED_s \in \mathcal{CON}(s)$ there exists $RED_t \in \mathcal{CON}(t)$ with $\text{rl}(RED_t) \leq \text{rl}(RED_s)$.*

Proof. Let $s \xrightarrow{S, \text{seq}} t$ and $RED_s \in \mathcal{CON}(s)$ with $\text{rl}(RED_s) = l$, we use induction on l . If $l = 0$ then s is a WHNF and the claim follows from Lemma 7.17. 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 $\xleftarrow{RED_s} s \xrightarrow{S, \text{seq}} t$ and have the cases:



For case (1), we can apply the induction hypothesis to $\xleftarrow{RED'} s_1 \xrightarrow{S, \text{seq}} t_1$. Case (2) is trivial. For case (3) we apply the induction hypothesis twice, i.e. firstly to $\xleftarrow{RED'} s_1 \xrightarrow{S, \text{seq}} s'_1$ and secondly to $\xleftarrow{RED''} s'_1 \xrightarrow{S, \text{seq}} t_1$. \square

Lemma 7.19. *A complete set of commuting diagrams for (iS, seq) is*



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.

$$\begin{array}{ccc}
 (\text{letrec } Env \text{ in } (\text{letrec } Env' \text{ in seq } v \ t)) \xrightarrow{iS, \text{seq}} (\text{letrec } Env \text{ in } (\text{letrec } Env' \text{ in } t)) \\
 \downarrow \text{no}, \text{lllet} & & \downarrow \text{no}, \text{lllet} \\
 (\text{letrec } Env, Env' \text{ in seq } v \ t) \xrightarrow{\text{no}, \text{seq}} (\text{letrec } Env, Env' \text{ in } t)
 \end{array}$$

\square

Lemma 7.20 ($\mathbb{C}_{\Leftarrow}(\text{seq})$). *If $s \xrightarrow{iS, \text{seq}} t$ then for every $RED_t \in \mathcal{CON}(t)$ there exists $RED_s \in \mathcal{CON}(s)$ with $\text{rl}_{(\setminus \text{seq})}(RED_s) \leq \text{rl}_{(\setminus \text{seq})}(RED_t)$.*

Proof. Let $s \xrightarrow{iS, \text{seq}} t$ and $RED_t \in \mathcal{CON}(t)$, we use induction on the measure μ ordered lexicographically with $\mu(RED) = (\mathbf{rl}_{(\setminus \text{seq})}(RED), \mathbf{rl}(RED))$. If $\mathbf{rl}(RED_t) = 0$ then the claim follows from Lemma 7.17. 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 7.19 to a prefix of $s \xrightarrow{iS, \text{seq}} t \xrightarrow{RED_t}$. With RED' being the suffix of RED_t of length $(m - 1)$ we have the cases:

$$\begin{array}{cccc}
 \begin{array}{ccc} s & \xrightarrow{iS, \text{seq}} & t \\ \text{no}, a \downarrow & & \downarrow \text{no}, a \\ s' & \xrightarrow{iS, \text{seq}} & t' \\ \text{RED}' \downarrow & & \downarrow \text{RED}' \end{array} &
 \begin{array}{ccc} s & \xrightarrow{iS, \text{seq}} & t \\ \text{no}, a \searrow & & \downarrow \text{no}, a \\ & & t' \\ & & \downarrow \text{RED}' \end{array} &
 \begin{array}{ccc} s & \xrightarrow{iS, \text{seq}} & t \\ \text{no}, \text{cp} \downarrow & & \downarrow \text{no}, \text{cp} \\ s' & \xrightarrow{iS, \text{seq}} & s'_1 \\ \text{RED}'' \downarrow & & \downarrow \text{RED}' \end{array} &
 \begin{array}{ccc} s & \xrightarrow{iS, \text{seq}} & t \\ \text{no}, a \downarrow & & \downarrow \text{no}, a \\ s' & \xrightarrow{iS, \text{seq}} & t' \\ \text{no}, \text{seq} \downarrow & & \downarrow \text{RED}' \end{array} \\
 (1) & (2) & (3) & (4)
 \end{array}$$

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

Since $\mathbb{C}_{\Rightarrow}(\text{seq})$ and $\mathbb{C}_{\Leftarrow}(\text{seq})$ hold the following corollary is true:

Corollary 7.21. *If $s \xrightarrow{S, \text{seq}} t$ then $s \uparrow$ iff $t \uparrow$.*

Lemma 7.22 ($\mathbb{D}_{\Rightarrow}(\text{seq})$). *If $s \xrightarrow{S, \text{seq}} t$ then for all $RED_s \in \mathcal{DIV}(s)$ there exists $RED_t \in \mathcal{DIV}(t)$ with $\mathbf{rl}(RED_t) \leq \mathbf{rl}(RED_s)$.*

Proof. Let $s = S[s_0], t = S[t_0]$, $s_0 \xrightarrow{\text{seq}} t_0$ and $RED_s \in \mathcal{DIV}(s)$ with $\mathbf{rl}(RED_s) = l$. The claim follows by induction on l , where the base case is covered by Corollary 7.21 and the induction step is analogous to the proof of Lemma 7.18. \square

Lemma 7.23 ($\mathbb{D}_{\Leftarrow}(\text{seq})$). *If $s \xrightarrow{iS, \text{seq}} t$ then for every $RED_t \in \mathcal{DIV}(t)$ there exists $RED_s \in \mathcal{DIV}(s)$ with $\mathbf{rl}_{(\setminus \text{seq})}(RED_s) \leq \mathbf{rl}_{(\setminus \text{seq})}(RED_t)$*

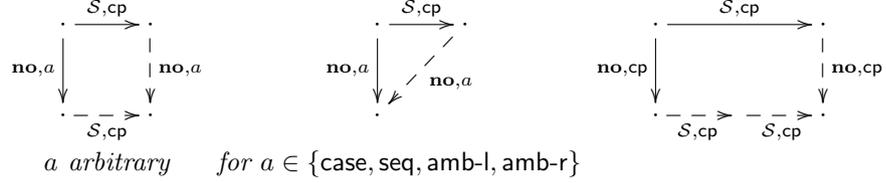
Proof. Let $s = S[s_0], t = S[t_0]$, $s_0 \xrightarrow{\text{seq}} t_0$ and $RED_t \in \mathcal{DIV}(t)$, we use induction on the measure $\mu(RED_t)$ ordered lexicographically, where $\mu(RED) = (\mathbf{rl}_{(\setminus \text{seq})}(RED), \mathbf{rl}(RED))$. If $\mathbf{rl}(RED_t) = 0$, then the claim follows from Corollary 7.21. The induction step is analogous to the proof of Lemma 7.20. \square

We have shown that (seq) fulfills the \mathbb{CD} -properties. Hence, we have:

Proposition 7.24. *(seq) is a correct program transformation.*

7.4 Correctness of (cp)

Lemma 7.25. *A complete set of forking diagrams for (S, cp) is*



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 (no, 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 (no, case) -, (no, seq) - or (no, 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. \square

Lemma 7.26. *If $s \xrightarrow{iS, \text{cp}} t$, then s is a WHNF iff t is a WHNF.*

Lemma 7.27 ($\mathbb{C} \Rightarrow (\text{cp})$).

If $s \xrightarrow{S, \text{cp}} t$, then for all $RED_s \in \text{CON}(s)$ there exists $RED_t \in \text{CON}(t)$ with $\text{r1}(RED_t) \leq \text{r1}(RED_s)$.

Proof. The proof is a copy of the proof of Lemma 7.18 using the complete set of forking diagrams for (S, cp) from Lemma 7.25 and using Lemma 7.26. \square

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

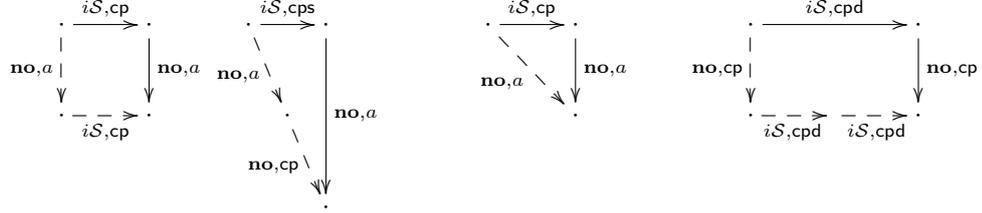
(cps) := the inner redex of the (cp) is of the form $S[x]$, i.e. the target is inside a surface context.

(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 7.28. *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 7.29. *Every (S, cps) - or (no, cp) -reduction strictly reduces the measure μ_{Sx} . Every (S, cpd) -reduction does not change the measure μ_{Sx} .*

Lemma 7.30. *A complete set of commuting diagrams for $(i\mathcal{S}, \text{cp})$ is*



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

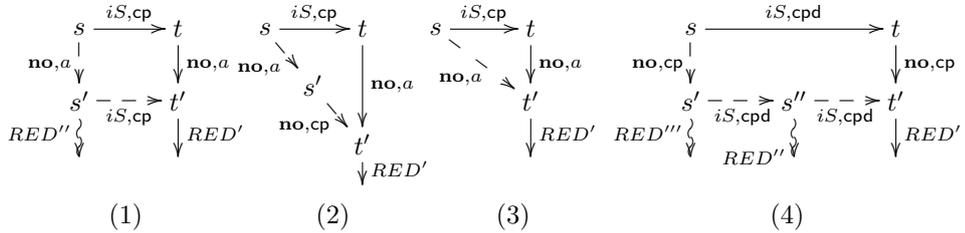
Proof. Follows by inspecting all cases where a $(i\mathcal{S}, \text{cp})$ -reduction is followed by a normal order reduction. The first diagram describe the case where the reductions commute. If the \mathcal{S} -internal (cp) -reduction becomes normal order, then the second diagram is applicable. The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

$$\begin{array}{l}
 \text{letrec } y = (\lambda x.s) \text{ in } (\text{seq } (\lambda z.z) y) \\
 \xrightarrow{i\mathcal{S}, \text{cps}} \text{letrec } y = (\lambda x.s) \text{ in } (\text{seq } (\lambda z.z) (\lambda x'.s[x'/x])) \\
 \xrightarrow{\text{no}, \text{seq}} \text{letrec } y = (\lambda x.s) \text{ in } (\lambda x'.s[x'/x]) \\
 \hline
 \text{letrec } y = (\lambda x.s) \text{ in } (\text{seq } (\lambda z.z) y) \\
 \xrightarrow{\text{no}, \text{seq}} \text{letrec } y = (\lambda x.s) \text{ in } y \\
 \xrightarrow{\text{no}, \text{cp}} \text{letrec } y = (\lambda x.s) \text{ in } (\lambda x'.s[x'/x])
 \end{array}$$

□

Lemma 7.31 ($\mathbb{C}_{\Leftarrow}(\text{cp})$). *If $s \xrightarrow{i\mathcal{S}, \text{cp}} t$ then for all $RED_t \in \mathcal{CON}(t)$ there exists $RED_s \in \mathcal{CON}(s)$ with $\text{rl}_{(\setminus \text{cp})}(RED_s) \leq \text{rl}_{(\setminus \text{cp})}(RED_t)$*

Proof. Let $s \xrightarrow{i\mathcal{S}, \text{cp}} t$ and $RED_t \in \mathcal{CON}(t)$. The claim follows by induction on the measure μ on reduction sequences with $\mu(s \xrightarrow{i\mathcal{S}, \text{cp}} t \xrightarrow{RED_t}) = (\text{rl}_{(\setminus \text{cp})}(RED_t), \mu_{Sx}(t))$. If $\mu(s \xrightarrow{i\mathcal{S}, \text{cp}} t \xrightarrow{RED_t}) = (0, 0)$, then from Lemma 7.29 we have that RED_t must be empty. Thus Lemma 7.26 shows the claim. Now, let $\mu(s \xrightarrow{i\mathcal{S}, \text{cp}} t \xrightarrow{RED_t}) = (l, m) > (0, 0)$. We apply a commuting diagram to a prefix of $s \xrightarrow{i\mathcal{S}, \text{cp}} t \xrightarrow{RED_t}$. 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 $\mathbf{rl}_{(\setminus \text{cp})}(RED') = l$ if the (\mathbf{no}, a) -reduction is an (\mathbf{no}, cp) -reduction, or
 - $\mathbf{rl}_{(\setminus \text{cp})}(RED') < l$, if the (\mathbf{no}, a) -reduction is not an (\mathbf{no}, 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 $\mathbf{rl}_{(\setminus \text{cp})}(RED'') \leq \mathbf{rl}_{(\setminus \text{cp})}(RED')$. By appending RED'' to $s \xrightarrow{\mathbf{no},a} s'$ we have $RED_s \in \mathcal{CON}(s)$ with $\mathbf{rl}_{(\setminus \text{cp})}(RED_s) \leq l$.

- (2) The sequence $RED_s = \xrightarrow{\mathbf{no},a} \xrightarrow{\mathbf{no},\text{cp}} \xrightarrow{RED'}$ is in $\mathcal{CON}(s)$ and $\mathbf{rl}(RED_s) \leq l$.
- (3) We can construct the sequence $RED_s = \xrightarrow{\mathbf{no},a} \xrightarrow{RED'}$ with $RED_s \in \mathcal{CON}(s)$ and $\mathbf{rl}_{(\setminus \text{cp})}(RED_s) \leq l$.
- (4) Since $t \xrightarrow{\mathbf{no},\text{cp}} t'$ we have with Lemma 7.29 that $\mu_{Sx}(t') < m$. From $s' \xrightarrow{iS,\text{cpd}} s''$ and $\xrightarrow{iS,\text{cpd}} t'$ we also have with Lemma 7.29 that $\mu_{Sx}(s'') = \mu_{Sx}(s') < m$. Since $\mathbf{rl}_{(\setminus \text{cp})}(RED') = l$ we can apply the induction hypothesis to $s'' \xrightarrow{iS,\text{cpd}} t' \xrightarrow{RED'}$ and have $RED'' \in \mathcal{CON}(s'')$ with $\mathbf{rl}_{(\setminus \text{cp})}(RED'') \leq l$. Since $\mu(s'') < m$ we can apply the induction hypothesis for a second time to $s' \xrightarrow{iS,\text{cpd}} s'' \xrightarrow{RED''}$ and have $RED''' \in \mathcal{CON}(s')$ with $\mathbf{rl}_{(\setminus \text{cp})}(RED''') \leq l$. Finally we append RED''' to $s \xrightarrow{\mathbf{no},\text{cp}} s'$ and have $RED_s \in \mathcal{CON}(s)$ with $\mathbf{rl}_{(\setminus \text{cp})}(RED_s) \leq l$. □

With Lemma 5.11, the following corollary is true.

Corollary 7.32. *If $s \xrightarrow{S,\text{cp}} t$ then $s \uparrow \text{iff } t \uparrow$.*

Lemma 7.33 ($\mathbb{D} \Rightarrow (\text{cp})$). *If $s \xrightarrow{S,\text{cp}} t$, then for all $RED_s \in \mathcal{DTV}(s)$ there exists $RED_t \in \mathcal{DTV}(t)$ with $\mathbf{rl}(RED_t) \leq \mathbf{rl}(RED_s)$*

Proof. The proof is a copy of the proof of Lemma 7.22 using the complete set of forking diagrams for (S, cp) from Lemma 7.25 and for the base case using Corollary 7.32. □

Lemma 7.34 ($\mathbb{D} \Leftarrow (\text{cp})$). *If $s \xrightarrow{iS,\text{cp}} t$ then for all $RED_t \in \mathcal{DTV}(t)$ there exists $RED_s \in \mathcal{DTV}(s)$ with $\mathbf{rl}_{(\setminus \text{cp})}(RED_s) \leq \mathbf{rl}_{(\setminus \text{cp})}(RED_t)$*

Proof. The proof is analogous to the proof of Lemma 7.31 using Corollary 7.32. □

We have shown that (cp) fulfills the \mathbb{CD} -properties. Hence, we have:

Proposition 7.35. *(cp) is a correct program transformation.*

8 The Standardisation Theorem and an Application

We summarise the results of the previous sections.

Theorem 8.1. *All deterministic reductions of the calculus $\Lambda_{\text{amb}}^{\text{let}}$ preserve contextual equivalence, i.e. if $s \xrightarrow{a} t$ with $a \in \{\text{beta}, \text{lll}, \text{case}, \text{seq}, \text{cp}\}$ then $s \sim_c t$.*

Proof. Follows from the Propositions 5.8, 7.15, 7.1, 7.24 and 7.35. \square

We will now develop properties of the reduction (amb), that will be necessary for the proof of the Standardisation Theorem (Theorem 8.14).

8.1 Properties of the Reduction (amb)

Lemma 8.2. *If $s \xrightarrow{iS, \text{amb}} t$ then s is a WHNF iff t is a WHNF.*

Proof. Follows by definition of WHNFs. \square

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

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

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

$$\begin{array}{l}
 \text{letrec } x_1 = c \vec{t}_i, \{x_i = x_{i-1}\}_{i=2}^m, \text{Env in } C[\text{amb } x_m \ s] \\
 \xrightarrow{\text{cpx}, m-1} \text{letrec } x_1 = c \vec{t}_i, \{x_i = x_{i-1}\}_{i=2}^m, \text{Env in } C[\text{amb } x_1 \ s] \\
 \xrightarrow{\text{cpcx}} \text{letrec } x_1 = c \vec{y}_i, \{y_i = t_i\}_{i=1}^{\text{ar}(c)}, \{x_i = x_{i-1}\}_{i=2}^m, \text{Env in } C[\text{amb } (c \vec{y}_i) \ s] \\
 \xrightarrow{\text{amb-c}} \text{letrec } x_1 = c \vec{y}_i, \{y_i = t_i\}_{i=1}^{\text{ar}(c)}, \{x_i = x_{i-1}\}_{i=2}^m, \text{Env in } C[(c \vec{y}_i)] \\
 \xrightarrow{\text{cpcx}} \text{letrec } x_1 = c \vec{t}_i, \{x_i = x_{i-1}\}_{i=2}^m, \text{Env in } C[x_1] \\
 \xrightarrow{\text{cpx}, m-1} \text{letrec } x_1 = c \vec{t}_i, \{x_i = x_{i-1}\}_{i=2}^m, \text{Env in } C[x_m]
 \end{array}$$

In case of an abstraction:

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

\square

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

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 6.15 and Lemma 5.1 show the claim. \square

Remark 8.5. An (amb)-reduction may introduce must-convergence, e.g. consider the terms $s \equiv \text{case}_{\text{Bool}} (\text{amb True False}) (\text{True} \rightarrow \Omega) (\text{False} \rightarrow \text{False})$ and $t \equiv \text{case}_{\text{Bool}} \text{False} (\text{True} \rightarrow \Omega) (\text{False} \rightarrow \text{False})$. 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 $(i\mathcal{S}, \text{amb-c})$ can be read off the following diagrams*

$$\begin{array}{ccc}
 \begin{array}{ccc} \cdot & \xrightarrow{i\mathcal{S}, \text{amb-c}} & \cdot \\ \text{no}, a \downarrow & & \downarrow \text{no}, a \\ \cdot & \xrightarrow{i\mathcal{S}, \text{amb-c}} & \cdot \end{array} &
 \begin{array}{ccc} \cdot & \xrightarrow{i\mathcal{S}, \text{amb-c}} & \cdot \\ \text{no}, a \searrow & & \downarrow \text{no}, a \\ \cdot & & \cdot \end{array} &
 \begin{array}{ccc} \cdot & \xrightarrow{i\mathcal{S}, \text{amb-c}} & \cdot \\ \text{no}, a \downarrow & & \downarrow \text{no}, a \\ \cdot & \xrightarrow{\text{no}, \text{amb-c}} & \cdot \end{array} \\
 a \text{ arbitrary} & a \in \{\text{case}, \text{seq}, \text{amb-l}, \text{amb-r}\} & a \text{ arbitrary}
 \end{array}$$

Proof. Follows by case analysis. The reductions either commute or the redex of the $(\mathcal{S}, \text{amb-c})$ is discarded, or the internal (amb-c) reduction becomes normal order. \square

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

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

Proof. Let $s = S[s_0], t = S[t_0]$ with $s_0 \xrightarrow{\text{amb-c}} t_0$ and $s \Downarrow$. We use induction on $l = \text{rl}(RED_s)$ with $RED_s \in \mathcal{CON}(s)$. If the $(\mathcal{S}, \text{amb-c})$ -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 $\xleftarrow{RED_s} s \xrightarrow{i\mathcal{S}, \text{amb}} t$. Let RED' be the suffix of RED_s of length $l - 1$, then we have the cases:

$$\begin{array}{ccc}
 \begin{array}{ccc} s & \xrightarrow{i\mathcal{S}, \text{amb-c}} & t \\ \text{no}, a \downarrow & & \downarrow \text{no}, a \\ s' & \xrightarrow{i\mathcal{S}, \text{amb-c}} & t' \\ RED' \downarrow & & \downarrow RED'' \end{array} &
 \begin{array}{ccc} s & \xrightarrow{i\mathcal{S}, \text{amb-c}} & t \\ \text{no}, a \downarrow & \nearrow \text{no}, a & \downarrow \text{no}, a \\ s' & & \cdot \\ RED' \downarrow & & \downarrow \end{array} &
 \begin{array}{ccc} s & \xrightarrow{i\mathcal{S}, \text{amb-c}} & t \\ \text{no}, a \downarrow & & \downarrow \text{no}, a \\ s' & \xrightarrow{\text{no}, \text{amb-c}} & s'' \\ RED' \downarrow & & \downarrow \end{array} \\
 (1) & (2) & (3)
 \end{array}$$

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

\square

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

Proof. Let $s \xrightarrow{S, \text{amb-c}} t$, 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 $(i\mathcal{S}, \text{amb-c})$. Let $s = S[s_0], t = S[t_0]$ with $s_0 \xrightarrow{\text{amb-c}} t_0$ and $t \uparrow$. We use induction $l = \mathbf{rl}(RED_t)$ 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:

$$\begin{array}{ccc}
 \begin{array}{ccc}
 s \xrightarrow{i\mathcal{S}, \text{amb-c}} t & & \\
 \text{no}, a \downarrow & & \downarrow \text{no}, a \\
 s' \xrightarrow{i\mathcal{S}, \text{amb-c}} t' & & \\
 \text{RED}'' \downarrow & & \downarrow \text{RED}'
 \end{array} &
 \begin{array}{ccc}
 s \xrightarrow{i\mathcal{S}, \text{amb-c}} t & & \\
 \text{no}, a \searrow & & \downarrow \text{no}, a \\
 s' & & \\
 \text{RED}' \downarrow & &
 \end{array} &
 \begin{array}{ccc}
 s \xrightarrow{i\mathcal{S}, \text{amb-c}} t & & \\
 \text{no}, a \downarrow & & \downarrow \text{no}, a \\
 s' \xrightarrow{\text{no}, \text{amb-c}} s'' & & \\
 \text{RED}' \downarrow & & \downarrow \text{RED}'
 \end{array} \\
 (1) & (2) & (3)
 \end{array}$$

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

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

Proposition 8.10. *If $s \xrightarrow{S, \text{amb}} t$ then $t \uparrow \implies s \uparrow$*

Proof. Follows from Lemma 8.9 and 8.3 using Theorem 8.1 and Proposition 6.15. \square

Corollary 8.11. *The property $\mathbb{D}_{\Leftarrow}(\text{amb})$ holds.*

Remark 8.12. We are not able to prove Proposition 8.10 for all contexts. The context lemma for must-convergence does not help, since the $s \leq_c^{\perp} t$ does not hold. If we used diagrams for $(i\mathcal{C}, \text{amb-c})$ instead of $(i\mathcal{S}, \text{amb-c})$, then we would have the additional diagram

$$\begin{array}{ccc}
 \cdot & \xrightarrow{i\mathcal{C}, \text{amb-c}} & \cdot \\
 \text{no}, \text{cp} \downarrow & & \downarrow \text{no}, \text{cp} \\
 \cdot & \xrightarrow{i\mathcal{C}, \text{amb-c}} & \cdot \\
 & \xrightarrow{i\mathcal{C}, \text{amb-c}} &
 \end{array}$$

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

Moreover, there is no chance for proving Proposition 8.10 for all contexts, since then the claim becomes wrong as the following example shows:

Example 8.13. There is an unexpected counterexample showing that $s \xrightarrow{\mathcal{C}, \text{amb-c}} t$ and $t \uparrow$ do not imply $s \uparrow$.

$$\begin{aligned} s &= \text{letrec } && y = \lambda z. \text{amb True False} \\ & && m = \lambda x. \text{if } (y \ 0) \text{ then } m \ x \ \text{else True} \\ & && \text{in } m \ \text{True} \\ \\ t &= \text{letrec } && y = \lambda z. \text{True} \\ & && m = \lambda x. \text{if } (y \ 0) \text{ then } m \ x \ \text{else True} \\ & && \text{in } m \ \text{True} \end{aligned}$$

The definition ensures that $s \xrightarrow{\mathcal{C}, \text{amb-c}} t$. Inspecting the normal order reduction, we see that $s \downarrow$ but $t \uparrow$. Consequences of this example are that Proposition 8.10 cannot be generalised from surface contexts to arbitrary contexts, and that the second part of the standardisation theorem 8.14 cannot be generalised to arbitrary contexts.

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) :

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

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 8.14 (Standardisation).

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

Proof. 1. Let $t \equiv t_0 \xrightarrow{\mathcal{C}, \text{red}_1} t_1 \xrightarrow{\mathcal{C}, \text{red}_2} \dots \xrightarrow{\mathcal{C}, \text{red}_{k-1}} t_k \equiv t'$ where t' is a WHNF. Using Theorem 8.1, Proposition 6.15 and Lemma 8.4 we have for every $t_i \xrightarrow{\mathcal{C}, \text{red}_{i+1}} t_{i+1}$ that if $t_{i+1} \downarrow$ then $t_i \downarrow$. Using induction on k we can show $t_0 \downarrow$.

2. Let $t \equiv t_0 \xrightarrow{\mathcal{S}, \text{red}_1} t_1 \xrightarrow{\mathcal{S}, \text{red}_2} \dots \xrightarrow{\mathcal{S}, \text{red}_{k-1}} t_k \equiv t'$ where $t' \uparrow$. With Theorem 8.1, Proposition 6.15 and Proposition 8.10 we have for every $t_i \xrightarrow{\mathcal{S}, \text{red}_{i+1}} t_{i+1}$ that if $t_{i+1} \uparrow$ then $t_i \uparrow$. Using induction on k we can show $t_0 \uparrow$. \square

Since fair normal order reduction induces the same notions of convergence and divergence as normal order reduction, we can transfer the Standardisation Theorem to fair evaluation:

Corollary 8.15.

- Let t be a term with $t \xrightarrow{\mathcal{C}, \text{allr}, *}$ t' where t' is a WHNF, then $t \Downarrow_F$.
- Let t be a term with $t \xrightarrow{\mathcal{S}, \text{allr}, *}$ t' where $t' \Uparrow$, then $t \Uparrow_F$.

Note that Corollary 8.15 cannot be generalised to arbitrary contexts due to the counterexample 8.13.

Using the second part of the standardisation theorem, we can prove the following proposition:

Proposition 8.16. *Let s, t be expressions with $s \Downarrow$ ($s \Downarrow_F$, respectively) and $s \xrightarrow{\mathcal{S}, \text{allr}, *}$ t . Then $t \Downarrow$ ($t \Downarrow_F$, respectively).*

Proof. Let $s \xrightarrow{\mathcal{S}, \text{allr}, *}$ t , we show that if $t \Uparrow$ then $s \Uparrow$. Since $t \Uparrow$, there exists t' with $t \xrightarrow{\text{no}, *}$ t' and $t' \Uparrow$. The sequence $\xrightarrow{\mathcal{S}, \text{allr}, *} \xrightarrow{\text{no}, *}$ is also a sequence of $(\mathcal{S}, \text{allr})$ -transformations. Hence, we can apply Theorem 8.14 and have $s \Uparrow$. The claim for fair evaluation follows from Theorem 3.20.

8.3 On the Equivalence of Ω -terms

Definition 8.17. *Let s be an expression. If for all environments Env , $(\text{letrec } Env \text{ in } s) \Uparrow$, then s is called an Ω -term.*

An example for such an expression is $(\text{letrec } y = \lambda z.z, x = x \text{ in } x)$. The purpose of this section is to show that all Ω -terms are equal w.r.t. \sim_c . In calculi with erratic choice (e.g. [SSSS04]) this result is easy to prove: In those calculi for every reduction context R the term $R[s]$ with s being an Ω -term $R[s]$ is must-divergent. This does not hold for our calculi, since the normal order redexes are not unique and since **amb** is bottom-avoiding, e.g. for the reduction context $R \equiv (\text{amb } [\cdot] \text{ True})$ the expression $R[s]$ may-converges for every expression s . Thus we will analyse the reduction behaviour of expressions of the form $S[s]$, where S is a surface context and s is an Ω -term. Then we will apply the context lemma.

In the following let S be a surface context and s be an Ω -term. Let RED be a normal order reduction of $S[s]$, i.e. $RED = S[s] = t_0 \xrightarrow{\text{no}} t_1 \xrightarrow{\text{no}} \dots \xrightarrow{\text{no}} t_n$. We label subterms and bindings of the successor reducts t_i of $S[s]$ with the labels BM, BL, respectively, where we write B , if we mean BL or BM. A subterm is a B -term, if it is within a term t of a BL-labeled binding $x = t$, or a subterm of the BM-labelled term.

Definition 8.18. *A labelling is admissible, if the following properties hold:*

- The label BL only occurs at bindings that are in a surface context, and the label BM labels at most one subterm in a surface context.
- The bound variables in bindings labelled BL only occur in B-labelled sub-expressions.

We can reconstruct an S -part, a B -part and an E -part of every t_i as follows:

- The S -part of t_i can be obtained by replacing the BM-labelled subterm with the hole, and by removing all the BL-labelled bindings. Every **letrec** with empty binding is eliminated using (**gc**). The resulting expression is denoted as $\text{Rec}_S(t_i)$. It may be a surface context, or an expression.
- The B -part is obtained as $(\text{letrec } Env \text{ in } r)$, where Env consists exactly of all BL-labelled bindings, and r is the subexpression that is labelled BM. If there is no such expression, then the B -part is undefined. The resulting expression is denoted as $\text{Rec}_B(t_i)$.
- The E -part is the set of bindings $x = t$ in a surface context of t_i that are not labelled BL. The resulting environment is denoted as $\text{Rec}_E(t_i)$.
- The EB-part $\text{Rec}_{EB}(t_i)$ is defined as $(\text{letrec } \text{Rec}_E(t_i) \text{ in } \text{Rec}_B(t_i))$.

Note that the S and the EB-part have the E -part in common. Note also that $\text{Rec}_S(t_0) = S$ and that $\text{Rec}_B(t_0) = s$.

Now we define how the labelling is initially done and how it is propagated in the reduction RED .

Definition 8.19. *Initially, the labelling is $S[s^{\text{BM}}]$. The labelling BM, initially and after every reduction step, is propagated according to the rule:*

$$(\text{letrec } x_1 = s_1, \dots, x_n = s_n \text{ in } r)^{\text{BM}} \rightarrow (\text{letrec } x_1 =^{\text{BL}} s_1, \dots, x_n =^{\text{BL}} s_n \text{ in } r^{\text{BM}})$$

For the normal order reductions the labelling is inherited in general using the rules of labelled reduction, where the exceptions are as follows:

- (**lbeta**): $(s_1^{\text{BM}} s_2)$ is never a normal order redex, see Lemma 8.21.
- (**case**): $(\text{case } s_1^{\text{BM}} \text{ alts})$ is never a normal order redex, see Lemma 8.21. The same for $(\text{case } x \text{ alts})$, where x is bound (perhaps over a variable-chain) to the BM-labelled expression.
- (**seq**), (**amb**): no exception.
- (**ll**): Only (**llet-e**) has to be specified:
 $(\text{letrec } x =^{\text{BL}} (\text{letrec } y_1 = r_1, \dots, y_m = r_m \text{ in } r_0), Env \text{ in } u) \rightarrow$
 $(\text{letrec } x =^{\text{BL}} r_0, y_1 =^{\text{BL}} r_1, \dots, y_m =^{\text{BL}} r_m, Env \text{ in } u)$
- (**cp**): $(\text{letrec } x = s, \dots C[x] \dots) \rightarrow (\text{letrec } x = s, \dots C[s])$. The cases $(\text{letrec } x = \lambda y.s, \dots C[x^{\text{BM}}] \dots)$ and $(\text{letrec } x = (\lambda y.s)^{\text{BM}}, \dots C[x] \dots)$ are not possible, see Lemma 8.21.

Definition 8.20. *Given a sequence of normal order reductions RED of $S[s]$, i.e. $RED = S[s] = t_0 \xrightarrow{\text{no}} t_1 \xrightarrow{\text{no}} \dots \xrightarrow{\text{no}} t_n$, the following projected reduction sequences are defined:*

- If $\text{Rec}_S(t_i) \neq \text{Rec}_S(t_{i+1})$, then $\text{Rec}_S(t_i) \rightarrow \text{Rec}_S(t_{i+1})$. We will show in Lemma 8.22 that these may be reductions (cp), (seq), (case), (amb), (lbeta),(lll) or (abs) in a surface context.
- If $\text{Rec}_{\text{EB}}(t_i) \neq \text{Rec}_{\text{EB}}(t_{i+1})$, then $\text{Rec}_{\text{EB}}(t_i) \rightarrow \text{Rec}_{\text{EB}}(t_{i+1})$. We will show that these are normal order reductions.

The derived sequences of reductions are denoted as RED_S and RED_{EB} , respectively.

8.3.1 Properties of B-Labelled Expressions The construction of the induction proof on the length of the reduction sequence RED has to be done using several claims at once, since only a mutual induction of all the properties will be successful. So we prove three claims within one lemma.

Lemma 8.21. *Let s be an Ω -term, S be a surface context and $RED = t_0 \xrightarrow{\text{no}} t_1 \xrightarrow{\text{no}} \dots \xrightarrow{\text{no}} t_n$ be a sequence of normal order reductions of $S[s]$. Then the following holds:*

1. The derived sequence of reductions RED_{EB} is a sequence of normal order reductions of $\text{Rec}_{\text{EB}}(t_0)$. Moreover, the BM-labelled subexpression is never a value nor bound to a value, and RED_{EB} never ends with a WHNF.
2. The labelling remains admissible after every normal order reduction step.
3. The impossible cases of reductions in Definition 8.19 do not occur.

Proof. The base cases hold obviously. Now assume the lemma holds for the sequence of reductions until t_i . The cases where the normal order reduction $t_i \xrightarrow{\text{no}} t_{i+1}$ only modifies the S -part, but not the EB -part, or where the BM-labelled part is modified are trivial. In the other case the unwinding algorithm first walks through the S -part, then perhaps visits the BM-labelled subterm, and then remains within the EB -part.

Part 1: The same unwinding performance is valid for the term $\text{Rec}_{\text{EB}}(t_i)$, after visiting the BM-labelled subterm. We see that the reduction on $\text{Rec}_{\text{EB}}(t_i)$ either changes nothing, i.e. $\text{Rec}_{\text{EB}}(t_i) = \text{Rec}_{\text{EB}}(t_{i+1})$ or is a normal order reduction step of $\text{Rec}_{\text{EB}}(t_i)$. The BM-labelled term in t_{i+1} is not a value nor bound to a value: Since RED_{EB} is a sequence of normal order reductions from 1 to $i + 1$, we would otherwise obtain a WHNF $\text{Rec}_{\text{EB}}(t_{i+1})$ after the reduction step, which is impossible, since s is an Ω -term and $\text{Rec}_S(t_0) = S$.

Part 2: Since the BM-labelled expression is never a value nor bound to a value, it is not possible to duplicate it, nor to extract the direct subexpressions of the BM-labelled subexpression using (abs) or (case). The bound variables from BL-labelled bindings can only occur in B -subterms, also after a reduction step.

Part 3: The impossible cases cannot happen, since they correspond exactly to the case where $\text{Rec}_{\text{EB}}(t_{i+1})$ is a WHNF. \square

8.3.2 Equivalence of Ω -terms

Lemma 8.22. *The derived sequence of reductions RED_S consists of some reductions (cp), (seq), (case), (amb), (lbeta), (lll) or (abs) in a surface context. If t_n is a WHNF, then $Rec_S(t_n)$ is a WHNF. If $Rec_S(t_n)$ is a WHNF, then there is an (lll,*)-reduction of t_n to a WHNF.*

Proof. The reductions may be completely in non- B -subexpressions. In this case the claim holds. The same holds if there are B -subexpressions that do not interfere with the reduction. The exception is that a (case)-reduction is performed in the B -part, and the effect in the E -part is like an (abs). Note that in general the reductions on $Rec_S(t_i)$ are not forced to be normal order reductions.

If t_n is a WHNF, then the value subterm is not labelled BM, hence it is also in $Rec_S(t_n)$, and thus it is also a WHNF. If $Rec_S(t_n)$ is a WHNF, then the only difference to a WHNF of t_n may be that some (llet-in)-reductions are missing to reduce it to a WHNF.

Theorem 8.23. *Let s, s' be two Ω -terms. Then $s \sim_c s'$*

Proof. We show that for every surface context S , $S[s]\downarrow \Rightarrow S[s']\downarrow$ and that $S[s]\uparrow \Rightarrow S[s']\uparrow$. Since the other case are symmetric, the context lemma implies that $s \sim_c s'$.

If $S[s]\downarrow$, then let RED be some sequence of normal order reductions of $S[s]$ ending in a WHNF. By Lemma 8.22, the surface reduction sequence RED_S yields a WHNF of $S = Rec_S(t_0)$, hence the same reduction yields a WHNF of $S[s']$. Using the standardisation theorem yields $S[s']\downarrow$.

Now assume that $S[s]\uparrow$, but $S[s']\downarrow$. Then let RED_1 be the sequence of normal order reductions of $S[s]$ to a term s_0 with $s_0\uparrow$. The sequence of reductions $RED_{1,S}$ reduces $Rec_S(t_0)$ to a context S_0 that is not a WHNF. Note that $S_0 = Rec_S(s_0)$. Let $r_0 := Rec_B(s_0)$. Then $S_0[r_0] \xrightarrow{(lll),*} s_0$, hence $S_0[r_0] \sim_c s_0$ by Proposition 7.15 and so $S_0[r_0]\uparrow$. Now we apply $RED_{1,S}$ to $S[s']$. If $RED_{1,S}$ eliminates the position of the hole of S , then obviously the result is a must-divergent expression, since $s_0\uparrow$, and we have reached a contradiction. Otherwise we obtain a sequence of transformations $S[s'] \xrightarrow{S,allr,*} S_0[s']$ from $RED_{1,S}$. Using Proposition 8.16 we have $S_0[s']\downarrow$. Now we use a fresh B -labelling for $S_0[s']$. There exists a term s'' with $S_0[s'] \xrightarrow{no,*} s''$ and s'' is a WHNF. Again using Lemma 8.22 we can construct RED_{2,S_0} that starts with $Rec_S(S_0[s'])$ and ends in a surface context S' that is a WHNF. Now we apply RED_{2,S_0} to $S_0[r_0]$ leading to a WHNF which contradicts $s_0\uparrow$. Thus the assumption was wrong and $S[s']\uparrow$ holds.

8.4 Proving a Bottom-Avoidance Law

For reasoning we use specific Ω -terms. At the end we will extend our results to all Ω -terms.

Definition 8.24 (Simple Ω -term). A term t is called a simple Ω -term in a context S , if either $t \equiv \Omega$ (see Example 4.4), or t is a variable x and S contains a **letrec**-binding $x = x$.

We define the bottom-avoidance law as a program transformation:

$$\begin{aligned} (\text{amb-l-o}) \quad & (\text{amb } s \ t) \rightarrow s, \text{ if } t \text{ is a simple } \Omega\text{-term.} \\ (\text{amb-r-o}) \quad & (\text{amb } s \ t) \rightarrow t, \text{ if } s \text{ is a simple } \Omega\text{-term.} \end{aligned}$$

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 $(S, \text{amb-o})$ 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 (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 8.14 shows that for these cases also a normal order reduction exists.

Lemma 8.25. A complete set of forking diagrams for $(S, \text{amb-o})$ is:

$$\begin{array}{cccc} \begin{array}{c} \cdot \xrightarrow{S, \text{amb-o}} \cdot \\ \text{no}, a \downarrow \quad \downarrow \text{no}, a \\ \cdot \xrightarrow{S, \text{amb-o}} \cdot \end{array} & \begin{array}{c} \cdot \xrightarrow{S, \text{amb-o}} \cdot \\ \text{no}, a \downarrow \quad \swarrow \text{no}, a \\ \cdot \end{array} & \begin{array}{c} \cdot \xrightarrow{S, \text{amb-o}} \cdot \\ \text{no}, a \downarrow \quad \nearrow S, \text{amb-o} \\ \cdot \end{array} & \begin{array}{c} \cdot \xrightarrow{S, \text{amb-o}} \cdot \\ \text{no}, a \downarrow \quad \downarrow S, \text{gc}^{-1} \\ \cdot \xrightarrow{S, \text{amb-o}} \cdot \end{array} \\ a \text{ arbitrary} & \text{for } a \in \{\text{case}, \text{seq}, \text{amb-l}, \text{amb-r}\} & a \text{ arbitrary} & a \text{ arbitrary} \end{array}$$

Proof. Follows by inspecting all cases where an $(S, \text{amb-o})$ -transformation and a normal order reduction overlap. The first diagram covers the cases where the reductions are performed independently. If the redex of the $(S, \text{amb-o})$ -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 **(no, 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. \square

Lemma 8.26. Let s, t be terms with $s \xrightarrow{iS, \text{amb-o}} t$ then s is a WHNF iff t is a WHNF.

Lemma 8.27 ($(\mathbb{C} \Rightarrow (\text{amb-o}))$). Let s, t be terms with $s \xrightarrow{S, \text{amb-o}} t$ 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 (S, allr) -transformations starting with t that leads to a WHNF. Then Theorem 8.14 part 1 shows that $t \downarrow$. The base case follows from Lemma 8.26. If $l > 0$ and the first reduction of RED_s is same

reduction as the $(S, \text{amb-o})$ -transformation, the claim holds. Otherwise, we apply a forking diagram from Lemma 8.25 to $\overleftarrow{RED_s} s \xrightarrow{S, \text{amb-o}}$. Let RED' be the suffix of RED_s of length $l - 1$, we have the following cases:

$$\begin{array}{cccc}
 \begin{array}{ccc} s & \xrightarrow{S, \text{amb-o}} & t \\ \text{no}, a \downarrow & & \downarrow \text{no}, a \\ s' & \xrightarrow{S, \text{amb-o}} & t' \\ RED' \downarrow & & \downarrow RED' \end{array} &
 \begin{array}{ccc} s & \xrightarrow{S, \text{amb-o}} & t \\ \text{no}, a \downarrow & \swarrow \text{no}, a & \downarrow \text{no}, a \\ s' & & t' \\ RED' \downarrow & & \downarrow RED' \end{array} &
 \begin{array}{ccc} s & \xrightarrow{S, \text{amb-o}} & t \\ \text{no}, a \downarrow & \swarrow S, \text{amb-o} & \downarrow \text{no}, a \\ s' & & t' \\ RED' \downarrow & & \downarrow RED' \end{array} &
 \begin{array}{ccc} s & \xrightarrow{S, \text{amb-o}} & t \\ \text{no}, a \downarrow & & \downarrow S, \text{gc} \\ s' & \xrightarrow{S, \text{amb-o}} & t' \\ RED' \downarrow & & \downarrow RED' \end{array} \\
 (1) & (2) & (3) & (4)
 \end{array}$$

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

Lemma 8.28. *A complete set of commuting diagrams for $(iS, \text{amb-o})$ is:*

$$\begin{array}{cccc}
 \begin{array}{ccc} \cdot & \xrightarrow{S, \text{amb-o}} & \cdot \\ \text{no}, a \downarrow & & \downarrow \text{no}, a \\ \cdot & \xrightarrow{S, \text{amb-o}} & \cdot \\ \cdot & \xrightarrow{S, \text{amb-o}} & \cdot \end{array} &
 \begin{array}{ccc} \cdot & \xrightarrow{S, \text{amb-o}} & \cdot \\ S, a \downarrow & & \downarrow \text{no}, a \\ \cdot & \xrightarrow{S, \text{amb-o}} & \cdot \\ \cdot & \xrightarrow{S, \text{amb-o}} & \cdot \end{array} &
 \begin{array}{ccc} \cdot & \xrightarrow{S, \text{amb-o}} & \cdot \\ \text{no}, a \swarrow & & \downarrow \text{no}, a \\ \cdot & & \cdot \\ \cdot & \xrightarrow{S, \text{amb-o}} & \cdot \end{array} &
 \begin{array}{ccc} \cdot & \xrightarrow{S, \text{amb-o}} & \cdot \\ \text{no}, \text{lamb} \downarrow & & \downarrow S, \text{amb-o} \\ \cdot & & \cdot \\ \cdot & \xrightarrow{S, \text{amb-o}} & \cdot \end{array}
 \end{array}$$

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 result of the (amb-o) -transformation is eliminated by a normal order reduction. The last diagram covers the case where an (no, llet) is applicable to the result of the $(S, \text{amb-o})$ -transformation but only because the **amb**-expression has been eliminated, e.g.

$$\begin{array}{l}
 \frac{\text{letrec } Env_1 \text{ in amb } \Omega ((\text{letrec } Env_2 \text{ in } s))}{\xrightarrow{S, \text{amb-o}} \text{letrec } Env_1 \text{ in } ((\text{letrec } Env_2 \text{ in } s))} \\
 \frac{\text{letrec } Env_1 \text{ in } ((\text{letrec } Env_2 \text{ in } s))}{\text{letrec } Env_1 \text{ in amb } \Omega ((\text{letrec } Env_2 \text{ in } s))} \\
 \frac{\text{no}, \text{lamb}}{\xrightarrow{\text{no}, \text{lamb}} \text{letrec } Env_1 \text{ in } (\text{letrec } Env_2 \text{ in amb } \Omega s)} \\
 \frac{\text{letrec } Env_1 \text{ in } (\text{letrec } Env_2 \text{ in amb } \Omega s)}{\xrightarrow{S, \text{amb-o}} \text{letrec } Env_1 \text{ in } (\text{letrec } Env_2 \text{ in } s)}
 \end{array}$$

\square

Lemma 8.29 $((C_{\leftarrow}(\text{amb-o}))$). *Let s, t be terms with $s \xrightarrow{S, \text{amb-o}} t$ and $t \downarrow$, then $s \downarrow$.*

Proof. Let $s = S[s_0]$ and $t = S[t_0]$ with $s \xrightarrow{S, \text{amb-o}} t$ and $t \downarrow$. We show by induction on the lexicographically ordered measure (a, b) where $b = \mu_{ll}(s)$ and

$a = \mathbf{rl}(RED_t)$ where $RED_t \in \mathcal{CON}(t)$, that there exists a sequence of $(\mathcal{S}, \text{allr})$ -transformations starting from s that ends in a WHNF. Then from Theorem 8.14 part 1 follows that $s \downarrow$. If the $(\mathcal{S}, \text{amb-o})$ is also a normal order reduction, then $s \downarrow$. Otherwise, the base case is covered by Lemma 8.26 and for the induction step we apply a commuting diagram from Lemma 8.28 to $s \xrightarrow{S, \text{amb-o}} t \xrightarrow{RED_t}$. With RED' being the suffix of RED_t of length $a - 1$ we have the cases:

$$\begin{array}{cccc}
 \begin{array}{ccc}
 s & \xrightarrow{S, \text{amb-o}} & t \\
 \text{no}, a \downarrow & & \downarrow \text{no}, a \\
 s' & \xrightarrow{S, \text{amb-o}} & t' \\
 & & \downarrow RED'
 \end{array} &
 \begin{array}{ccc}
 s & \xrightarrow{S, \text{amb-o}} & t \\
 \mathcal{S}, a \downarrow & & \downarrow \text{no}, a \\
 s' & \xrightarrow{S, \text{amb-o}} & t' \\
 & & \downarrow RED'
 \end{array} &
 \begin{array}{ccc}
 s & \xrightarrow{S, \text{amb-o}} & t \\
 \text{no}, a \searrow & & \downarrow \text{no}, a \\
 & & t' \\
 & & \downarrow RED'
 \end{array} &
 \begin{array}{ccc}
 s & \xrightarrow{S, \text{amb-o}} & t \\
 \text{no}, \text{lamb} \downarrow & & \downarrow RED_t \\
 s' & \xrightarrow{S, \text{amb-o}} & t'
 \end{array} \\
 (1) & (2) & (3) & (4)
 \end{array}$$

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

Since $\mathbb{C}_{\Rightarrow}(\text{amb-o})$ and $\mathbb{C}_{\Leftarrow}(\text{amb-o})$ hold, we have:

Corollary 8.30. *If $s \xrightarrow{S, \text{amb-o}} t$ then $s \uparrow$ iff $t \uparrow$.*

Lemma 8.31. $\mathbb{D}_{\Leftarrow}(\text{amb-o})$

Proof. Induction on the measure (a, b) where $b = \mu_{ll}(s)$ and $a = \mathbf{rl}(RED_t)$ with $RED_t \in \mathcal{DIV}(t)$ shows that there exists a sequence of $(\mathcal{S}, \text{allr})$ -transformations starting with s and ending in a term that must-diverge. The base case is covered by Corollary 8.30 and the induction step uses the commuting diagrams from Lemma 8.28. As final step Theorem 8.14 part 2 shows that $s \uparrow$. \square

Lemma 8.32. $\mathbb{D}_{\Rightarrow}(\text{amb-o})$

Proof. Induction on $\mathbf{rl}(RED_s)$ with $RED_s \in \mathcal{DIV}(s)$ shows the existence of a sequence of $(\mathcal{S}, \text{allr})$ -transformations from t to a term that must-diverge. The base case for this induction is covered by Corollary 8.30, the induction step uses the forking diagrams from Lemma 8.25. The last step uses Theorem 8.14 part 2 to transform the sequence of $(\mathcal{S}, \text{allr})$ -transformations into a normal order reduction sequence $RED_t \in \mathcal{DIV}(t)$. \square

Since (amb-o) fulfills the \mathbb{CD} -properties we have

Proposition 8.33. *If $s \xrightarrow{\text{amb-o}} t$ then $s \sim_c t$.*

Theorem 8.34. *For all Ω -terms s and expressions t the following holds:*

$$\text{amb } s \ t \sim_c t \sim_c \text{amb } t \ s$$

Proof. Let s be an Ω -term, then from Theorem 8.23 we have $s \sim_c \Omega$, since Ω is an Ω -term. Now since \sim_c is a congruence and using Proposition 8.33 the claim follows. \square

Theorem 8.35. *For all Ω -terms s and expressions t the following holds:*

$$\mathbf{amb} \ s \ t \sim_c t \sim_c \mathbf{amb} \ t \ s$$

Proof. Let s be an Ω -term, then from Theorem 8.23 we have $s \sim_c \Omega$, since Ω is an Ω -term. Now since \sim_c is a congruence and using Proposition 8.33 the claim follows.

8.5 On the Relation Between \leq_c^\downarrow and \leq_c^\uparrow

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

Lemma 8.36. $BA[s] \Downarrow$ 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

$$\begin{aligned} BA[s'] &\xrightarrow{S, \mathbf{seq}} (\mathbf{amb} \ \mathbf{I} \ (\lambda x.\Omega)) \ \mathbf{I} \xrightarrow{S, \mathbf{amb}} (\lambda x.\Omega) \ \mathbf{I} \\ &\xrightarrow{\mathbf{lbeta}} (\mathbf{letrec} \ x = \mathbf{I} \ \mathbf{in} \ \Omega) \xrightarrow{\mathbf{gc}} \Omega \end{aligned}$$

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

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

- $s' \equiv (\mathbf{letrec} \ x_1 = (c_{T,i} \ \overrightarrow{s_i}), \{x_i = x_{i-1}\}_{i=2}^m, Env \ \mathbf{in} \ x_m)$ 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 $t' \equiv (\mathbf{letrec} \text{ Env in } BA[s'])$. If the first reduction of RED'' is an (**amb-l**) reduction, then either

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

or

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

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 **letrecs** over the context BA . Now $s \xrightarrow{RED_s} t''$, with $t'' \equiv s'$ or $t'' \equiv (\mathbf{letrec} \ \text{Env in } s')$, i.e. t'' is a WHNF, and hence $s\downarrow$. \square

Proposition 8.37. $\leq_c^\downarrow \subseteq (\leq_c^\downarrow)^{-1}$

Proof. Let s, t be arbitrary terms with $s \leq_c^\downarrow t$. Then $\forall C \in \mathcal{C} : C[s]\downarrow \implies C[t]\downarrow$ and thus also $\forall C \in \mathcal{C} : BA[C[s]]\downarrow \implies BA[C[t]]\downarrow$. Using Lemma 8.36 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 $t \leq_c^\downarrow s$. \square

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 8.38. *If $s \leq_c t$ then $s \sim_c^\downarrow t$.*

A consequence of Proposition 8.37 is that contextual equivalence can be defined using only must-convergence.

Corollary 8.39. *$s \sim_c t$ 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.

Proposition 8.40. *Let s be a Ω -term and t be an arbitrary term, then $s \leq_c^\downarrow t$.*

Proof. Let (rplom) be the transformation, that replaces a simple Ω -term by an arbitrary term. A complete set of forking diagrams for (S , rplom) is:

$$\begin{array}{ccc}
 \begin{array}{c} \cdot \\ \xrightarrow{S, \text{rplom}} \cdot \\ \text{no}, a \downarrow \quad \downarrow \text{no}, a \\ \cdot \\ \xrightarrow{S, \text{rplom}} \cdot \\ a \text{ arbitrary} \end{array} &
 \begin{array}{c} \cdot \\ \xrightarrow{S, \text{rplom}} \cdot \\ \text{no}, \text{III} \downarrow \quad \downarrow \text{no}, \text{gc}^{-1} \\ \cdot \\ \xrightarrow{S, \text{rplom}} \cdot \end{array} &
 \begin{array}{c} \cdot \\ \xrightarrow{S, \text{rplom}} \cdot \\ \text{no}, a \downarrow \quad \swarrow \text{no}, a \\ \cdot \end{array} \\
 & & \text{for } a \in \{\text{case}, \text{seq}, \text{amb-l}, \text{amb-r}\}
 \end{array}$$

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 simple Ω -term is floated out, via an (III)-reduction, then it needs to be deleted via an (S , gc)-transformation, or the simple 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 $l = \text{rl}(RED_s)$, 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 $\xleftarrow{RED_s} s_0 \xrightarrow{S, \text{rplom}}$. With RED' being the suffix of RED_s of length $l - 1$ we have the cases:

$$\begin{array}{ccc}
 \begin{array}{c} s_0 \xrightarrow{S, \text{rplom}} t_0 \\ \text{no}, a \downarrow \quad \downarrow \text{no}, a \\ s_1 \xrightarrow{S, \text{rplom}} t_1 \\ \text{RED}' \downarrow \quad \downarrow \text{RED}'' \end{array} &
 \begin{array}{c} s_0 \xrightarrow{S, \text{rplom}} t_0 \\ \text{no}, \text{III} \downarrow \quad \downarrow \text{no}, \text{gc}^{-1} \\ s_1 \xrightarrow{S, \text{rplom}} t_1 \\ \text{RED}' \downarrow \quad \downarrow \text{RED}'' \end{array} &
 \begin{array}{c} s_0 \xrightarrow{S, \text{rplom}} t_0 \\ \text{no}, a \downarrow \quad \swarrow \text{no}, a \\ s_1 \\ \text{RED}' \downarrow \end{array} \\
 (1) & (2) & (3)
 \end{array}$$

For cases (1) and (2) we apply the induction hypothesis to $\xleftarrow{RED'} s_1 \xrightarrow{S, \text{rplom}} t_1$ and have a sequence RED'' of (S , allr)-transformations starting with t_1 that ends in a WHNF. By appending $t_0 \xrightarrow{S, \text{allr}} t_1$ to RED'' we have such a sequence for t_0 . Case (3) is trivial. Finally, the context lemma for may-convergence shows that $s \leq t$ for s being a simple Ω -term. Using Theorem 8.23 we can generalise this result to all Ω -terms, since \leq_c is a precongruence. \square

Lemma 8.41. *\leq_c is not symmetric.*

Proof. Let $s \equiv \text{choice } \Omega \mathbf{I}$ and $t \equiv \mathbf{I}$. From Lemma 8.40 we have $s \leq_c^\downarrow t$. For all surface contexts S we can $S[s]$ transform into $S[t]$:

$$\begin{aligned}
 S[s] &\equiv S[(\text{amb } (\lambda x. \Omega) (\lambda x. \mathbf{I})) \text{ True}] \xrightarrow{S, \text{amb}} S[(\lambda x. \mathbf{I}) \text{ True}] \\
 &\xrightarrow{S, \text{lbeta}} S[(\text{letrec } x = \text{True in } \mathbf{I})] \xrightarrow{S, \text{gc}} S[\mathbf{I}] \equiv S[t]
 \end{aligned}$$

From the Standardisation Theorem follows that for all surface contexts $S[t]\uparrow \implies S[s]\uparrow$. Hence, using Corollary 4.15 we have $s \leq_c t$. Obviously $s\uparrow$ and $t\downarrow$. Thus, the empty context shows that $t \not\leq_c^\downarrow s$. \square

9 Conclusion and Further Research

We presented an extended call-by-need lambda-calculus with a non-deterministic **amb**-operator together with a fair small-step reduction semantics. The appropriate program equivalence is the contextual equivalence that is based on may- and must-termination. We proved that all deterministic reduction rules and several additional program transformations preserve contextual equivalence, which permits useful program transformation, and in particular partial evaluation using deterministic reductions. This clearly is an improvement of the results in Moran's thesis, since we also proved correctness w.r.t. must-convergence. The proof methodology consists of a context lemma which restricts the number of contexts that need to be examined, and computing complete sets of commuting and forking diagrams for the reductions and transformations.

A remarkable result is that contextual preorder \leq_c and equivalence \sim_c can be defined by observing only the must-convergent behaviour. Concerning must-divergent expressions, we showed that all must-divergent expressions are in the same equivalence class of \sim_c and that must-divergent expressions are not least w.r.t. \leq_c .

With the developed proof tools it appears promising to prove correctness of further program transformations, e.g. a rule for inlining expressions that are used only once, or for deterministic expressions (i.e. do not contain **amb**-expressions, also satisfying some other conditions). Future research should investigate also more involved inductive proof rules like Bird's take-lemma.

Another research direction would be to apply the used methods (proving a context lemma, computing sets of reduction diagrams, and using an unfair normal order reduction to ease proofs) to other non-deterministic calculi.

A further challenge would be to perform an investigation similar to [Sab03b,Sab03a] and thus to prove correctness of program transformations used in Haskell [Pey03] compilers w.r.t. an extension by **amb**. After switching off incorrect transformations the result would be a semantics preserving compiler for Haskell extended with **amb**.

Acknowledgements

We would like to thank the anonymous referees of the journal *Mathematical Structures in Computer Science* for their valuable comments and suggestions.

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. André Rauber Du Bois, Robert F. Pointon, Hans-Wolfgang Loidl, and Philip W. Trinder. Implementing declarative parallel bottom-avoiding choice. In *SBAC-PAD*, pages 82–92. IEEE Computer Society, 2002.
- CHS05. Arnaud Carayol, Daniel Hirschhoff, and Davide Sangiorgi. On the representation of McCarthy’s amb in the pi-calculus. *Theor. Comput. Sci.*, 330(3):439–473, 2005.
- DP92. B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, Cambridge, 1992.
- FK03. Maribel Fernández and Lionel Khalil. Interaction nets with McCarthy’s amb: Properties and applications. *Nordic J. Comput.*, 10(2):134–162, 2003.
- 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 Johan Jeuring and Erik Meijer, editors, *Advanced Functional Programming*, volume 925 of *Lecture Notes in Comput. Sci.*, pages 137–182. Springer, 1995.
- 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 Bøgh 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 Bøgh 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 Bøgh Lassen and Andrew Moran. Unique fixed point induction for McCarthy’s amb. In Mirosław Kutylowski, Leszek Pacholski, and Tomasz

- Wierzbicki, editors, *MFCS*, volume 1672 of *Lecture Notes in Comput. Sci.*, pages 198–208. Springer, 1999.
- 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 Wolfgang 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.
- MT00. Elena Machkasova and Franklyn A. Turbak. A calculus for link-time compilation. In Gert Smolka, editor, *ESOP*, volume 1782 of *Lecture Notes in Comput. Sci.*, pages 260–274. Springer, 2000.
- 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.
- Sab03a. David Sabel. A guide through HasFuse, 2003. available from <http://www.ki.informatik.uni-frankfurt.de/research/diamond/hasfuse/>.
- Sab03b. 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.

- WPK03. J. B. Wells, Detlef Plump, and Fairouz Kamareddine. Diagrams for meaning preservation. In Robert Nieuwenhuis, editor, *RTA*, volume 2706 of *Lecture Notes in Comput. Sci.*, pages 88–106. Springer, 2003.