On Equivalences and Standardization in a Non-Deterministic Call-by-Need Lambda Calculus

Manfred Schmidt-Schauß and Matthias Mann

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

Technical Report Frank-31

15. August 2007

Abstract. The goal of this report is to prove correctness of a considerable subset of transformations w.r.t. contextual equivalence in a an extended lambda-calculus with case, constructors, seq, let, and choice, with a simple set of reduction rules. Unfortunately, a direct proof appears to be impossible.
The correctness proof is by defining another calculus comprising the complex variants of copy, case-reduction and seq-reductions that use variable-binding chains. This complex calculus has well-behaved diagrams and allows a proof that of correctness of transformations, and also that the simple calculus defines an equivalent contextual order.

1 Introduction

Motivation The motivation for this report is to provide support for the proof method of simulation [Abr90,How89,Gor99] in a non-deterministic call-by-need lambda-calculi (see e.g. [AFM$^+$95,AF97,MOW98]) with let, case, constructors and seq. Simulation, and thus also bisimulation, is a proof tool for contextual preorder in deterministic lambda-calculi (see [Abr90,How89,Gor99,Pit97]), that covers a large amount of similarities and equalities that are out of reach for the approach using a context-lemma, overlap diagrams and transformations. The advantage of simulation-based equivalence proofs is that they lead, in a significant number of cases, in particular for programming languages with constructors, to a proof after a finite number of steps, and if not, it is often possible to use co-induction. The simulation-based method in its basic version appears to be automatable. In contrast, the proof method based on a context-lemma and overlap diagrams usually requires ad-hoc computations of overlaps, or the creative invention of helper-transformations including the computation of overlaps, and

appears to be hard to automate. The safety of the proof tool of simulation requires a proof to show that similarity is equivalent to contextual preorder in the respective calculi, or that simulation implies contextual preorder.

For non-deterministic call-by-name lambda-calculi, which are not the appropriate model for programming languages if non-determinism is involved, there is a corresponding proof in [How89]. For non-deterministic call-by-need lambda-calculi with non-recursive let, a proof of the "implies"-property is in [Man05a,Man05b], but in a calculus without constructors, i.e. lists and list-functions are not available. This work is extended to case and constructors for call-by-need-calculi in [MSS06,MSS07] thus extending the power of similarity. A shortcoming of this method is that for non-deterministic call-by-need lambda-calculi the normal-order reduction cannot be used for simulation proofs. Instead another variant of a standardizing reduction, the approximation reduction, is the appropriate one. Thus there remains the proof burden to show that the approximation reduction generates the same contextual preorder as the normal-order reduction. This is the goal of this research report, which proves exactly this relationship for a sufficiently expressive non-deterministic call-by-need lambda-calculus extended with case, constructors and seq.

Structure of the Report In this report we define three variants of a non-deterministic call-by-need lambda-calculus with case, constructors, seq, let and choice, and prove their equivalence w.r.t. contextual equivalence.

The first calculus L is a non-deterministic call-by-need lambda-calculus, where the reductions for case, seq and the copy-rule exploit variable-variable binding chains, and which is a calculus that prefers sharing over copying also for values. This calculus is rather well-behaved in a proof of correctness of reductions as transformations using a context lemma and overlap diagrams.

The second calculus L_S is a variant of the first one, but has simpler reduction rules, and prefers copying of values over looking for references. This calculus resists direct proofs of correctness of reductions as transformations w.r.t. contextual equivalence using a context lemma and overlap diagrams, since the overlap diagrams (i.e. commuting of reduction steps) cannot be controlled by a well-founded measure that allows induction proofs. However, it is possible to use the calculus L and to show that L and L_S have equivalent contextual pre-order, and thus this provides an indirect proof of correctness of lots of transformations w.r.t. the simple calculus L_S.

The third calculus L_A is a so-called approximation calculus that is an intermediate step to exploit (bi)simulation proofs of behavioral preorders and equivalences. This calculus instead of a normal-order reduction follows the strategy of reducing expressions top-down and approximately, without any let-shuffling rules. It can be seen as a call-by-value variant of call-be-need, however, in general a single normal-order reduction sequence is represented as an infinite number of reductions in the call-by-value calculus. The outcomes of the infinitely many reductions may be seen as approximates. They are derived by cutting reduction

in a certain depth and then removing the sharing, i.e. removing the top-let-environments.

Now proofs by simulation can use the reduction rules of the calculus L_A, since it is the appropriate one for simulation proofs in a call-by-need non-deterministic calculus as shown in [MSS06,MSS07].

2 The Call-By-Need Calculus With Chaining

The call-by-need calculus L has binary application, non-recursive `let`, , `lambda`, `seq`, `case` and constructors, with a normal-order reduction that defines evaluation to weak head normal forms. The calculus L is an adapted extension of the corresponding calculus in [Man05a,Man05b] by case and constructors, see also [SSSS04,MSC99] for nondeterministic calculi with case and constructors. The slightly more complicated definitions of (case) and (cp) are borrowed from the calculus in [SSSS07]. The reason is that for simpler formulations an equivalence proof appears to be only constructible via the more complicated calculus. Note that the exact definition of the syntax and rules may vary slightly in different papers, in particular the syntax and the rules dealing with case-expressions.

We describe the syntax. There is an infinite set V of variables and a finite set K of constructors with fixed arities. The canonical operators are the constructors from K and λ. The syntax is as follows, where E means expressions and $c, c_i \in K$:

$$\begin{aligned} E ::= {} & V \mid (E\ E) \mid \lambda x.E \mid (\texttt{let}\ V = E\ \texttt{in}\ E) \mid (\texttt{choice}\ E\ E) \mid (\texttt{seq}\ E\ E) \\ & \mid (c\ E_1 \ldots E_{ar(c)}) \mid (\texttt{case}\ E\ \texttt{of}\ (c_1\ V \ldots V) \to E; \ldots; (c_m\ V \ldots V) \to E) \end{aligned}$$

The untyped `case`-construct is assumed to have a pattern $(c\ x_1 \ldots x_{ar(c)})$ for every constructor $c \in K$, where the variables in a pattern have to be distinct. The scoping rules are as usual, where `let` is non-recursive, and hence the scope of x in $(\texttt{let}\ x = s\ \texttt{in}\ t)$ is the term t. We assume that expressions satisfy the distinct variable convention before reduction is applied, which can be achieved by a renaming of bound variables. A nested let-expression is written as $(\texttt{let}\ x_1 = s_1, \ldots, x_n = s_n\ \texttt{in}\ t)$, meaning $(\texttt{let}\ x_1 = s_1\ \texttt{in}\ (\texttt{let}\ x_2 = s_2 \ldots\ \texttt{in}\ (\texttt{let}\ x_n = s_n\ \texttt{in}\ t) \ldots))$.

We use labels indicating the normal order redex, where T means the top-term, S means a subterm reduction, V, W mean visited, where W marks positions that are not target for a (cp)-reduction, and M is a meta-variable that may stand for S or T. The shifting algorithm (unwind) starts with t^T, where no subexpression of t is labeled, and uses the following rules exhaustively.

$$\begin{aligned} (s\ t)^M &\to (s^S\ t)^V \\ (\texttt{let}\ x = s\ \texttt{in}\ t)^T &\to (\texttt{let}\ x = s\ \texttt{in}\ t^T)^V \\ (\texttt{let}\ x = s\ \texttt{in}\ C[x^M]) &\to (\texttt{let}\ x = s^S\ \texttt{in}\ C[x^V]) \\ &\qquad \text{if } C \neq C'[(\texttt{let}\ z = [.]\ \texttt{in}\ r)] \\ (\texttt{let}\ x = s\ \texttt{in}\ C[x^M]) &\to (\texttt{let}\ x = s^S\ \texttt{in}\ C[x^W]) \\ &\qquad \text{if } C = C'[(\texttt{let}\ z = [.]\ \texttt{in}\ r)] \\ (\texttt{seq}\ s\ t)^M &\to (\texttt{seq}\ s^S\ t)^V \\ (\texttt{case}\ s\ alts)^M &\to (\texttt{case}\ s^S\ alts)^V \end{aligned}$$

(lbeta)	$((\lambda x.s)^S\ r) \to (\texttt{let}\ x = r\ \texttt{in}\ s)$
(cp)	$(\texttt{let}\ x = v^S\ \texttt{in}\ C[y^V]) \to (\texttt{let}\ x = v\ \texttt{in}\ C[v])$
	where v is an abstraction and v is bound by y
(case-c)	$(\texttt{case}\ (c\ t_1 \dots t_n)^S \dots ((c\ y_1 \dots y_n) \to s) \dots)$
	$\to\ (\texttt{let}\ y_1 = t_1, \dots, y_n = t_n\ \texttt{in}\ s)$
(case-e)	$(\texttt{let}\ x = (c\ t_1 \dots t_n)^S\ \texttt{in}\ C[(\texttt{case}\ y^V \dots ((c\ y_1 \dots y_n) \to s) \dots)])$
	$\to\ \texttt{let}\ z_1 = t_1, \dots, z_n = t_n, x = (c\ z_1 \dots z_n)$
	$\texttt{in}\ C[(\texttt{let}\ y_1 = t_1, \dots, y_n = t_n\ \texttt{in}\ s)]$
	Here the expression $(c\ t_1 \dots t_n)$ is bound by y
(seq-c)	$(\texttt{seq}\ v^S\ t) \to t$ if v is a value
(seq-e)	$(\texttt{seq}\ x^V\ t) \to t$ if some value is bound by x
(choicel)	$(\texttt{choice}\ s\ t)^{S \vee T} \to s$
(choicer)	$(\texttt{choice}\ s\ t)^{S \vee T} \to t$
(llet)	$(\texttt{let}\ x_1 = (\texttt{let}\ x_2 = s_2\ \texttt{in}\ s_1)^L\ \texttt{in}\ t)$
	$\to (\texttt{let}\ x_2 = s_2, x_1 = s_1\ \texttt{in}\ t)$
(lapp)	$((\texttt{let}\ x = s_1\ \texttt{in}\ s_2)^S\ t) \to (\texttt{let}\ x = s_1\ \texttt{in}\ (s_2\ t))$
(lseq)	$(\texttt{seq}\ (\texttt{let}\ x = s_1\ \texttt{in}\ s_2)^S\ t) \to (\texttt{let}\ x = s_1\ \texttt{in}\ (\texttt{seq}\ s_2\ t))$
(lcase)	$(\texttt{case}\ (\texttt{let}\ x = s\ \texttt{in}\ t)^S\ alts) \to (\texttt{let}\ x = s\ \texttt{in}\ (\texttt{case}\ t\ alts))$

Fig. 1. Normal-order rules

In figure 1 the normal-order reduction rules are defined, where the rules are applied by first running the unwind-labeling on a label-free expression, and then applying the reduction rule, where L means any label. A *cv-expression* means an expression of the form $(c\ x_1 \dots x_n)$ where c is a constructor and x_i are variables. A *value* is an abstraction or a constructor-expression $(c\ t_1 \dots t_n)$. We use the following abbreviation for reduction rules: (choice) = (choicer) ∪ (choicel), (case) = (case-c) ∪ (case-e), (seq) = (seq-c) ∪ (seq-e), (lll) = (llet) ∪ (lapp) ∪ (lseq) ∪ (lcase). We say an expression r is bound by a variable x in t (which satisfies the distinct variable convention), if there is a chain of variables $x_i, i = 1, \dots, n$, such that $(\texttt{let}\ x_n = r\ \texttt{in}\ \dots)$, and $(\texttt{let}\ x_i = x_{i+1}\ \texttt{in}\ \dots)$ for $i = 1, \dots, n-1$ are subexpressions of t.

A weak head normal form (WHNF) is an expression $(\texttt{let}\ x_1 = t_1, \dots, x_n = t_n\ \texttt{in}\ v)$, where v is a value, or $(\texttt{let}\ x_1 = t_1, \dots, x_n = t_n\ \texttt{in}\ x_i)$, where x_i is bound to a constructor-expression. A *reduction context*is defined as a context, where the hole will be labeled with S or T by the labeling algorithm. We denote the class of reduction contexts by $\mathcal{R}$. A term s *converges*, iff $s \xrightarrow{*} s_1$ for some WHNF s_1 by a normal order reduction sequence, denoted as $s \Downarrow$. Two terms s, t are related by *contextual preorder*, $s \leq_c t$, iff $\forall C : C[s] \Downarrow \implies C[t] \Downarrow$, and s, t are *contextually equivalent*, $s \sim_c t$, iff $s \leq_c t$ and $t \leq_c s$. It is easy to verify that $\leq_c$ is a precongruence, i.e. it is an preorder and $s \leq_c t \implies C[s] \leq_c C[t]$ for all C; and that $\sim_c$ is an congruence, i.e. it is a precongruence that is also an equivalence relation.

Proving properties of the calculus, like correctness of the deterministic reduction rules as transformations can be done as e.g. in [SSSS04]. The context-lemma for L follows immediately from [SSS07].

3 Correctness of Transformations in the Chaining Calculus

In this section we prove correctness of a considerable set of transformations and also some properties w.r.t. different kinds of reduction lengths The technique is to analyze the contextual equivalence and preorder of L.

3.1 Context Lemma and Program Transformations

First, it is easy to see that the context lemma holds for L:

Lemma 3.1. *Let s, t be any expressions. Then $s \leq_c t$ iff for all reduction contexts R: $R[s] \Downarrow \Longrightarrow R[t] \Downarrow$.*

Proof. This holds since the general result for context lemmas of [SSS07] is applicable to L and its normal-order reduction. The proof is very similar to the proof in [SSSS07].

Lemma 3.2. *If $s \to t$ by one of the reductions (lbeta), (case-c), (seq-c), (lapp), (lseq), (lcase), then $s \sim_c t$.*

Proof. By the context lemma it is sufficient to treat the case that $s \to t$ on toplevel by one of the reductions, and to show that $R[s] \Downarrow \Leftrightarrow R[t] \Downarrow$ for any reduction context R. For all the mentioned reductions, it is easy to see by inspecting the rules and the definition of reduction contexts that $s \to t$ implies that $R[s] \xrightarrow{no} R[t]$, and that the normal-order reduction is unique. Hence $R[s] \Downarrow \Leftrightarrow R[t] \Downarrow$.

Lemma 3.3. *If $s \to t$ by one of the reductions (choicel), (choicer), then $t \leq_c s$.*

Proof. By the context lemma it is sufficient to treat the case that $s \to t$ on toplevel by one of the reductions, and to show that $R[t] \Downarrow \Leftrightarrow R[s] \Downarrow$ for any reduction context R. As in the proof of the previous lemma, it is easy to see that $s \to t$ implies that $R[s] \xrightarrow{no} R[t]$. Hence $R[t] \Downarrow \Leftrightarrow R[s] \Downarrow$.

The missing rules from the calculus are (case-e), (seq-e), (cp) and (llet). We will show in the following that these are also correct transformations, i.e. that $s \xrightarrow{a} t$ implies $s \sim_c t$. A *surface-context* S is a context, such that the hole is not within an abstraction. The class of let-right-contexts, defined by the grammar $\mathcal{L}_R := [] \mid \texttt{let}x = E \texttt{ in } \mathcal{L}_R$ will be used in a transformation rule. In figure 2 we define more transformation rules in L to complete the overall proof.

Lemma 3.4. *Let $s \xrightarrow{a} t$ by a reduction rule a in figure 1. If s is a WHNF, then t is a WHNF, and if t is a WHNF, then the reduction is a normal-order reduction.*

Proof. By inspection of the transformation rules in figure 1 and also the definition of unwind and reduction contexts.

(gc)	$\texttt{let}\ x = s\ \texttt{in}\ t$	$\to t$	if x does not occur in t
(cpx)	$\texttt{let}\ x = y\ \texttt{in}\ C[x]$	$\to \texttt{let}\ x = y\ \texttt{in}\ C[y]$	
(abs)	$(c\ t_1 \dots t_n)$	$\to (\texttt{let}\ x_1 = t_1, \dots, x_n = t_n\ \texttt{in}\ (c\ x_1 \dots x_n))$	
(cpcx)	$\texttt{let}\ x = v\ \texttt{in}\ C[x]$	$\to \texttt{let}\ x = v\ \texttt{in}\ C[v]$	if v is a cv-expression
($cp\mathcal{S}$)	$\texttt{let}\ x = v\ \texttt{in}\ S[y]$	$\to \texttt{let}\ x = v\ \texttt{in}\ S[v]$	if v is a an abstraction, v is bound by y and S a surface context
(cpd)	$\texttt{let}\ x = v\ \texttt{in}\ C[x]$	$\to \texttt{let}\ x = v\ \texttt{in}\ C[v]$	if v is an abstraction, v is bound by y and C a non-surface context

Fig. 2. Transformation Rules in L

Lemma 3.5. *Let $s \xrightarrow{a} t$ by a transformation rule a in figure 2. Then the following holds:*

- *For all a: if s is a WHNF then t is a WHNF.*
- *If $a \in \{(gc), (cpd), (cpx), (cpcx), \}$, then s is a WHNF iff t is a WHNF.*
- *If $a = (abs)$, t is a WHNF and s is not a WHNF, then $s \xrightarrow{no,lll,*} t$.*
- *If $a = (cp\mathcal{S})$ and t is a WHNF, and s is not a WHNF, then $(cp\mathcal{S})$ is in fact a normal-order (cp)-reduction.*

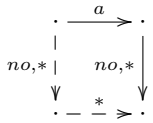
Proof. By inspection of the transformation rules in figure 2.

3.2 Forking and Commuting Diagrams

We compute for the reductions and transformations ($cp\mathcal{S}$), (lll), (cpd), (gc), (cpx) respectively, a complete set of forking and a complete set of commuting diagrams, restricted to reductions within surface contexts. The reductions that are not normal-order are always reductions in a surface context. (for a detailed definition of the complete sets of diagrams see [SSSS07]).

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

which is represented as follows, where a straight arrow means given reduction and a dashed arrow means existential reduction,

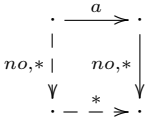


where the $\xrightarrow{*}$-sequence at the bottom may consist of several, different reductions. There may also be exceptional diagrams, where some edges are not present.

The condition for completeness is that for every reduction sequence $\cdot \xleftarrow{a} \cdot \xrightarrow{no,+} \cdot$, there is an applicable transformation of reduction sequences in the set.

A *complete set of commuting diagrams* for a transformation $\xrightarrow{a}$ is a finite set of transformation rules for reduction sequences of the form

which is represented as

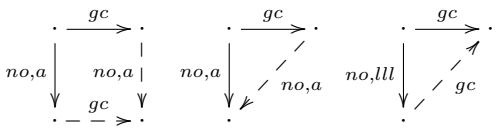


where the $\xrightarrow{*}$-sequence at the bottom may consist of several, different reductions. An implicit transformation rule is that a reduction step using a reduction from the calculus may become a normal-order step after the rearrangement. There may also be exceptional diagrams, where the $\xrightarrow{a}$ is omitted.

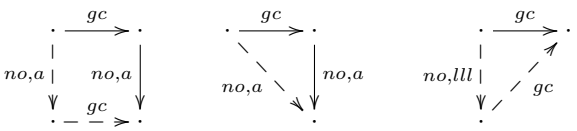
The condition for completeness is that for every reduction sequence $\cdot \xrightarrow{a} \cdot \xrightarrow{no,+} \cdot$, there is an applicable transformation of reduction sequences in the set.

3.3 All the Diagrams

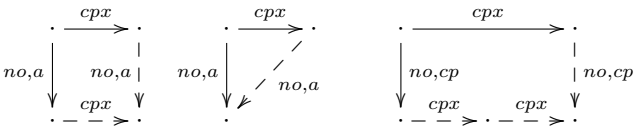
Diagrams for (gc)-reductions The forking diagrams for (gc) are



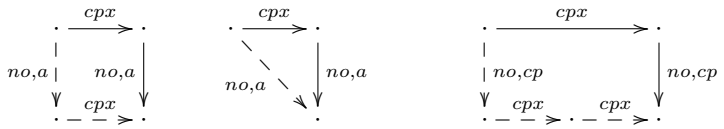
The commuting diagrams for (gc) are



Diagrams for (cpx) The forking diagrams for (cpx) are

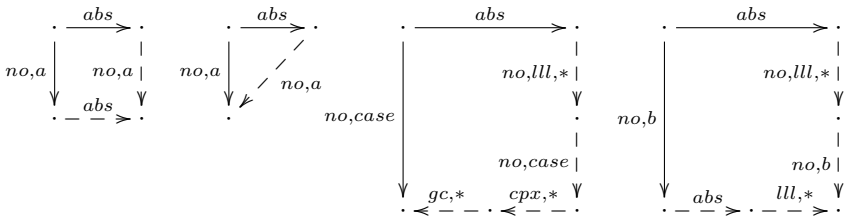


The commuting diagrams for (cpx) are



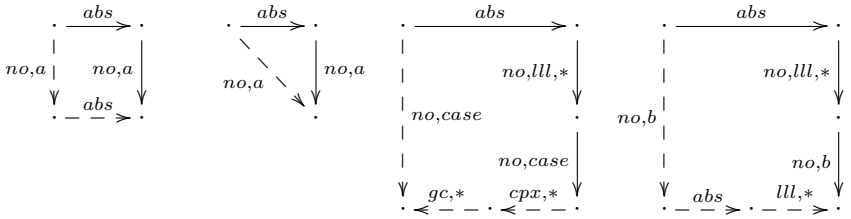
Diagrams for (abs)

The forking diagrams for (abs) are



where $b \in \{(\text{seq}), (\text{case})\}$.

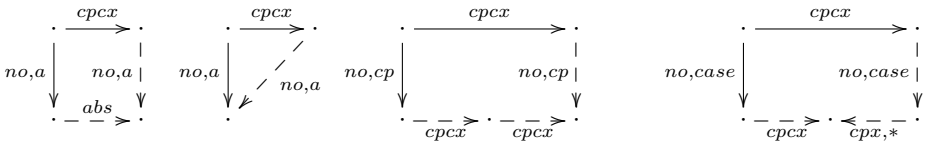
The commuting diagrams for (abs) are



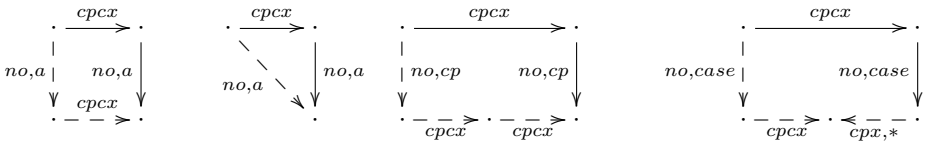
where $b \in \{(\text{seq}), (\text{case})\}$.

Diagrams for (cpcx)

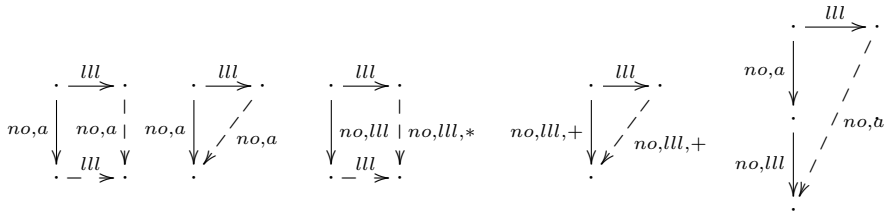
Note that the rule (cpcx) is different from the rule in [SSSS07].
The forking diagrams for (cpcx) are



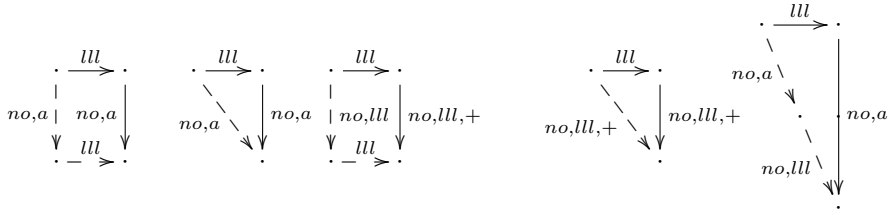
The commuting diagrams for (cpcx) are



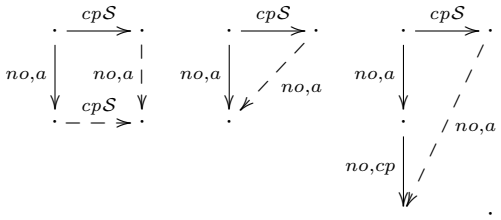
Diagrams for (lll) The forking diagrams for (lll) are



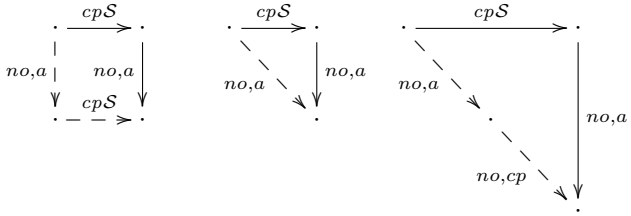
The commuting diagrams for (lll) are



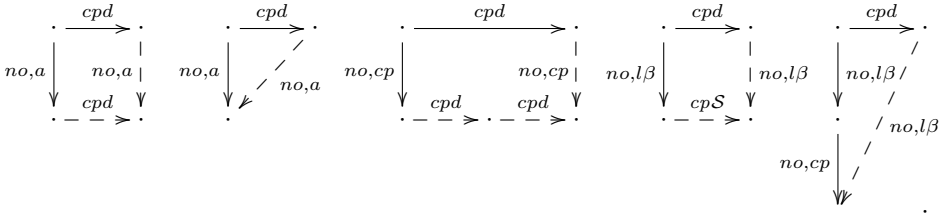
Diagrams for ($cp\mathcal{S}$)-reductions The forking diagrams for ($cp\mathcal{S}$) are



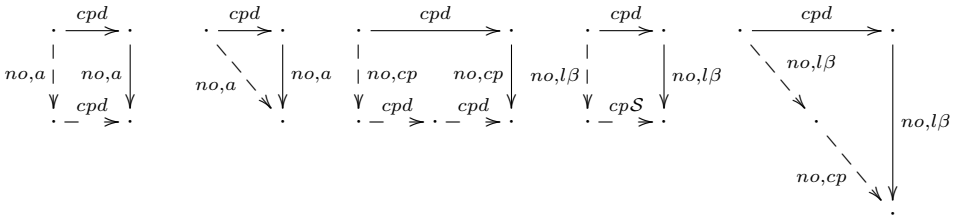
The commuting diagrams for ($cp\mathcal{S}$) are



Diagrams for (cpd)-reductions The forking diagrams for (cpd) are



The commuting diagrams for (cpd) are



Note that the fourth and fifth diagram turn a (cpd) into a $(cp\mathcal{S})$.

3.4 Proofs of Properties of Transformations

Lemma 3.6. *The transformation (lll) terminates. I.e. there are no infinite reduction sequences consisting only of (lll)-reductions.*

Proof. The terminating measure is the sum of the following number for every subexpression that is a let-expression s: the number of non-let-expressions and bindings $x = r$ that are above the subexpression s.

Lemma 3.7. *The transformation $\xrightarrow{(lll)\cup(\mathcal{S},(cp))}$ terminates. I.e. there are no infinite reduction sequences consisting only of $\xrightarrow{(lll)}$- and $\xrightarrow{\mathcal{S},(cp)}$-reductions.*

Proof. The terminating measure is a lexicographically ordered pair where the first component is the number of occurrences of variables in surface positions, and the second is the sum of the following number for every subexpression that is a let-expression s: the number of non-let-expressions and bindings $x = r$ that are above the subexpression s. The first component is strictly decreased by every $\xrightarrow{\mathcal{S},(cp)}$-reduction. The reduction $\xrightarrow{(lll)}$ leaves the first component invariant and strictly decreases the second. Hence the lemma holds.

Lemma 3.8. *The transformation (abs) is terminating. I.e. there are no infinite reduction sequences consisting only of (abs)-reductions.*

Proof. Obvious.

We define different length measures for finite, maximal reduction sequences U, which are reduction sequences that end in a WHNF, and where WHNFs are not reduced further. Note that the sequences might in general be non-normal-order reduction sequences. A maximal non-normal-order reduction sequence is also called *evaluation*.

Definition 3.9. *For reduction sequences U and a set $M \subseteq \{(case), (cp), (seq), (lbeta), (lll), (choice)\} = M_{max}$, we define $len_M(U)$ to be the number of reductions from M in U.*

The correctness proof are usually made using the following schema: To show that $s_0 \xrightarrow{\mathcal{S},a} t_0$ is correct, we use the context lemma, which shows that it is sufficient to show that for all reduction contexts R, $R[s_0] \Downarrow \Leftrightarrow R[t_0] \Downarrow$. Since R is also a surface context, it is sufficient to show for all expressions s, t with $s \xrightarrow{\mathcal{S},cpd} t$ and for every evaluation U of s, there is an evaluation U' of t, and vice versa, perhaps combined with a length claim concerning U, U'. Since the diagrams are always computed for surface context reductions, they are applicable.

Lemma 3.10. *The transformation (gc) is correct.*

Proof. Using the context lemma, we have to show that for $s \xrightarrow{\mathcal{S},gc} t$ and all reduction contexts R, $R[s] \Downarrow \Leftrightarrow R[t] \Downarrow$. Showing $R[s] \Downarrow \implies R[t] \Downarrow$ requires to use the forking diagrams by induction on the length of normal-order reduction sequences. The direction $R[t] \Downarrow \implies R[s] \Downarrow$ requires the commuting diagrams, induction on the length $len_{M_{max}}$ of normal-order reduction sequences, and Lemma 3.6. The base cases are in Lemma 3.5.

Lemma 3.11. *Let $s \xrightarrow{\mathcal{S},gc} t$, and let $M \subseteq \{(case), (cp), (seq), (lbeta), (choice)\}$. Then for every evaluation U of s there is an evaluation U' of t with $len_M(U) = len_M(V')$, and for every evaluation U' of t there is an evaluation U of s with $len_M(U) = len_M(V')$.*

Proof. This follows using exactly the same constructions as in Lemmas 3.10 and due to the diagrams for (gc). Note that the number of (lll)-reductions may be different.

Lemma 3.12. *The transformation (cpx) is correct and the following holds for the modification of the length of evaluations:*
Let $s \xrightarrow{\mathcal{S},cpx} t$, and let $M \subseteq \{(case), (cp), (seq), (lbeta), (lll), (choice)\}$. Then for every evaluation U of s there is an evaluation U' of t with $len_M(U) = len_M(U')$, and for every evaluation U' of t there is an evaluation U of s with $len_M(U) = len_M(U')$.

Proof. Using the context lemma, we have to show that for $s \xrightarrow{\mathcal{S},cpx} t$ and all reduction contexts R, $R[s] \Downarrow \Leftrightarrow R[t] \Downarrow$. This follows by induction on the length of normal-order reduction sequences using the diagrams: Showing $R[s] \Downarrow \implies R[t] \Downarrow$ requires to use the forking diagrams. For the induction the additional induction hypothesis on the above-mentioned length must be proved within the same induction due to third diagram for (cpx).

The direction $R[t] \Downarrow \implies R[s] \Downarrow$ requires the commuting diagrams and also a combined induction hypothesis The base cases are in Lemma 3.5.

Lemma 3.13. *The transformation (cpcx) is correct and the following holds for the modification of the length of evaluations:*
Let $s \xrightarrow{\mathcal{S},cpcx} t$, and let $M \subseteq \{(case), (cp), (seq), (lbeta), (lll), (choice)\}$. Then for every evaluation U of s there is an evaluation U' of t with $len_M(U) = len_M(U')$, and for every evaluation U' of t there is an evaluation U of s with $len_M(U) = len_M(U')$.

Proof. Using the context lemma, we have to show that for $s \xrightarrow{\mathcal{S},cpcx} t$ and all reduction contexts R, $R[s]\Downarrow \Leftrightarrow R[t]\Downarrow$. Showing $R[s]\Downarrow \implies R[t]\Downarrow$ requires to use the forking diagrams, and the Lemma 3.12 on reduction length of (cpx), where an induction on the length of a normal-order reduction is possible. We have to prove the combined induction hypothesis on the existence of a normal-order reduction and on the above-mentioned length The direction $R[t]\Downarrow \implies R[s]\Downarrow$ requires the commuting diagrams, and Lemma 3.12, and we can again use induction on the length of a normal-order-reduction, where again the combined induction hypothesis must be proved. The base cases are in Lemma 3.5.

Lemma 3.14. *The transformation (lll) is correct.*

Proof. Using the context lemma, we have to show that for $s \xrightarrow{\mathcal{S},lll} t$ and all reduction contexts R, $R[s]\Downarrow \Leftrightarrow R[t]\Downarrow$. Showing $R[s]\Downarrow \implies R[t]\Downarrow$ requires to use the forking diagrams, and the direction $R[t]\Downarrow \implies R[s]\Downarrow$ requires the commuting diagrams, where in both cases an induction on the length of a normal-order-reduction is possible. The base cases are in Lemma 3.5.

Lemma 3.15. *Let $s \xrightarrow{\mathcal{S},lll} t$, and let $M \subseteq \{(case),(cp),(seq),(lbeta),(choice)\}$. Then for every evaluation U of s there is an evaluation U' of t with $len_M(U) = len_M(U')$, and for every evaluation U' of t there is an evaluation U of s with $len_M(U) = len_M(U')$.*

Proof. This follows using exactly the same constructions as in Lemma 3.4, and due to the diagrams for (lll). Note that the number of (lll)-reductions may be different in the respective U, and U'.

Lemma 3.16. *The transformation (abs) is correct and the following holds for the modification of the length of evaluations:*
Let $s \xrightarrow{\mathcal{S},abs} t$, and let $M \subseteq \{(case),(cp),(seq),(lbeta),(choice)\}$. Then for every evaluation U of s there is an evaluation U' of t with $len_M(U) = len_M(U')$, and for every evaluation U' of t there is an evaluation U of s with $len_M(U) = len_M(U')$.

Proof. Using the context lemma, we have to show that for $s \xrightarrow{\mathcal{S},abs} t$ and all reduction contexts R, $R[s]\Downarrow \Leftrightarrow R[t]\Downarrow$. Showing $R[s]\Downarrow \implies R[t]\Downarrow$ requires to use the forking diagrams for (abs) with normal-order reduction, where the measure for induction is as follows: First $len_M(U)$, and second the number of all normal-order reductions. Lemmas 3.15, 3.11, 3.12 show that this is the correct induction measure, where the third and fourth diagrams strictly decrease the first component.

The direction $R[t]\Downarrow \implies R[s]\Downarrow$ requires the commuting diagrams and the same induction measure as for the previous case, and also the length properties of Lemmas 3.15, 3.11, 3.12. The base cases are in Lemma 3.5.

Lemma 3.17. *The transformation ($cp\mathcal{S}$) is correct.*

Proof. Using the context lemma, we have to show that for $s \xrightarrow{\mathcal{S},cp\mathcal{S}} t$ and all reduction contexts R, $R[s] \Downarrow \Leftrightarrow R[t] \Downarrow$. Showing $R[s] \Downarrow \implies R[t] \Downarrow$ requires to use the forking diagrams, where the induction is on the number of normal-order reductions. The direction $R[t] \Downarrow \implies R[s] \Downarrow$ requires the commuting diagrams and the same induction measure as for the previous case. The base cases are in Lemma 3.5.

Lemma 3.18. *Let $s \xrightarrow{\mathcal{S},cp\mathcal{S}} t$, and let $M \subseteq \{(case), (seq), (lbeta), (lll), (choice)\}$. Then for every evaluation U of s there is an evaluation U' of t with $len_M(U) = len_M(U')$, and for every evaluation U' of t there is an evaluation U of s with $len_M(U) = len_M(U')$. In addition, for $M' \subseteq \{(case), (seq), (cp), (lbeta), (choice)\}$, and for every evaluation U of s there is an evaluation U' of t with $len_M(U) \geq len_M(U')$.*

Proof. This follows using exactly the same constructions as in the proof of Lemma 3.17, and due to the diagrams for $(cp\mathcal{S})$. Note that the number of $(cp\mathcal{S})$-reductions may be different in the respective sequences U and U'.

The additional claim of the existence of some reduction sequence U' with $len_M(U) \geq len_M(U')$ also follows from the forking diagrams of $(cp\mathcal{S})$ by induction.

Lemma 3.19. *The transformation (cpd) is correct.*

Proof. Using the context lemma, we have to show that for $s_0 \xrightarrow{\mathcal{S},cpd} t_0$ and all reduction contexts R, $R[s_0] \Downarrow \Leftrightarrow R[t_0] \Downarrow$.
Showing $R[t_0] \Downarrow \implies R[s_0] \Downarrow$ requires to use the commuting diagrams: Since R is also a surface context, it is sufficient to show for all expressions s, t with $s \xrightarrow{\mathcal{S},cpd} t$ and for every evaluation U' of s, that there is an evaluation U of t, with $len_M(U) = len_M(U')$ for $M = \{(\text{case}), (\text{seq}), (\text{lbeta}), (\text{lll}), (\text{choice})\}$; i.e. the number of all non-cp-reductions. This is done by induction on the following measure of (t, U): first $len_M(U')$, second the number of occurrences of variables in surface contexts in t. Inspecting all the cpd- and $cp\mathcal{S}$-diagrams, the induction step can be proved. Note that for the third cpd-diagram that duplicates the (cpd) at the bottom arrow, the induction hypothesis is also applicable to the middle term, since (cp) strictly decreases the measure, but (cpd) does not change the second part of the measure.

The direction $s \Downarrow \implies t \Downarrow$ requires the commuting diagrams and can be done using a simpler induction measure: simply count the number of normal-order reductions in U.

The base cases are in Lemma 3.5.

Correctness of Copying Ω For the application to the approximation calculus we also need the correctness of the transformation
(cpbot) $(\texttt{let}\ x = \Omega\ \texttt{in}\ C[x]) \to (\texttt{let}\ x = \Omega\ \texttt{in}\ C[\Omega])$
where $\Omega = \Omega_1\ \Omega_1$ and $\Omega_1 = (\lambda x.(x\ x))$.

Lemma 3.20. *For all reduction contexts $R : R[\Omega]$ has no evaluation. This immediately implies via the context lemma, that for all $t : \Omega \leq_c t$.*

Proof. Every reduction of $R[\Omega]$ is of the form: $R[\Omega] \xrightarrow{no,l\beta} R[(\texttt{let}\ x = \Omega_1\ \texttt{in}\ (x\ x))] \xrightarrow{no,lll,*} R_1[(\texttt{let}\ x = \Omega_1\ \texttt{in}\ R_2[(x\ x)])]$, such that $(x\ x)$ is in a reduction context. The reduction will be continued as follows: $\xrightarrow{no,cp} R_1[(\texttt{let}\ x = \Omega_1\ \texttt{in}\ R_2[(\Omega_1\ x)])] \xrightarrow{no,l\beta} R_1[(\texttt{let}\ x = \Omega_1\ \texttt{in}\ R_2[(\texttt{let}\ x_1 = x\ \texttt{in}\ (x_1\ x_1))])]$. By induction it is easy to see that the normal-order reduction uses the same scheme again and again, and thus is infinite.

Lemma 3.21. *The reduction $\xrightarrow{s,(cpbot)}$ is correct. Moreover, if $s \xrightarrow{(cpbot)} t$, then for every $M \subseteq \{(case), (cp), (seq), (lbeta), (choice), (lll)\}$: for every evaluation U of s there is an evaluation U' of t with $len_M(U) = len_M(U')$, and for every evaluation U' of t there is an evaluation U of s with $len_M(U) = len_M(U')$.*

Proof. The simple argument is that whenever $s \xrightarrow{s,(\text{cpbot})} t$, then for every evaluation of s, the Ω, and also the corresponding x are never in a reduction context, and thus are not used in the reduction, hence the evaluations of s and t are the same in structure, and the intermediate expressions only may differ at non-reduction positions.

4 The Simple Calculus L_S and its Equivalence

In this section we describe a simpler variant L_S of the calculus L, where the reduction rules are a simplified, and show that the corresponding convergence and contextual equivalences are equivalent. The syntax is the same, only the reductions differ.

4.1 A Simple Call-By-Need Calculus

Now we define the corresponding notions of normal-order reduction for the calculus L_S. We will omit the label L_S in the notation in this subsection, but will distinguish the notions for L and L_S, if it is necessary.

We use labels indicating the normal order redex, where T means the top-term, S means a subterm reduction and M is a meta-variable that may stand for S or T. The shifting algorithm (unwind) starts with t^T and uses the following rules exhaustively.

$$
\begin{array}{ll}
(s\ t)^M & \to (s^S\ t)^M \\
(\texttt{let}\ x = s\ \texttt{in}\ t)^T & \to (\texttt{let}\ x = s\ \texttt{in}\ t^T)^T \\
(\texttt{let}\ x = s\ \texttt{in}\ C[x^M]) & \to (\texttt{let}\ x = s^S\ \texttt{in}\ C[x^S]) \ \text{ for a context } C \\
(\texttt{seq}\ s\ t)^M & \to (\texttt{seq}\ s^S\ t)^M \\
(\texttt{case}\ s\ alts)^M & \to (\texttt{case}\ s^S\ alts)^M
\end{array}
$$

The normal-order reduction rules are as defined in figure 3.

(lbeta)	$((\lambda x.s)^S\ r) \to (\texttt{let}\ x = r\ \texttt{in}\ s)$
(cp)	$(\texttt{let}\ x = v^S\ \texttt{in}\ C[x^S]) \to (\texttt{let}\ x = v\ \texttt{in}\ C[v])$
	where v is an abstraction or a cv-expression
(abs)	$(c\ t_1 \ldots t_n)^S \to (\texttt{let}\ x_1 = t_1, \ldots, x_n = t_n\ \texttt{in}\ (c\ x_1 \ldots x_n))$
	if $(c\ t_1 \ldots t_n)$ is not a cv-expression, where x_i are fresh variables
(case)	$(\texttt{case}\ (c\ t_1 \ldots t_n)^S \ldots ((c\ y_1 \ldots y_n) \to s) \ldots)$
	$\to\ \ (\texttt{let}\ y_1 = t_1, \ldots, y_n = t_n\ \texttt{in}\ s)$
(seq)	$(\texttt{seq}\ v^S\ t) \to t$ if v is a value
(choicel)	$(\texttt{choice}\ s\ t)^{S \vee T} \to s$
(choicer)	$(\texttt{choice}\ s\ t)^{S \vee T} \to t$
(llet)	$(\texttt{let}\ x_1 = (\texttt{let}\ x_2 = s_2\ \texttt{in}\ s_1)^S\ \texttt{in}\ t)$
	$\to (\texttt{let}\ x_2 = s_2, x_1 = s_1\ \texttt{in}\ t)$
(lapp)	$((\texttt{let}\ x = s_1\ \texttt{in}\ s_2)^S\ t) \to (\texttt{let}\ x = s_1\ \texttt{in}\ (s_2\ t))$
(lseq)	$(\texttt{seq}\ (\texttt{let}\ x = s_1\ \texttt{in}\ s_2)^S\ t) \to (\texttt{let}\ x = s_1\ \texttt{in}\ (\texttt{seq}\ s_2\ t))$
(lcase)	$(\texttt{case}\ (\texttt{let}\ x = s\ \texttt{in}\ t)^S\ alts) \to (\texttt{let}\ x = s\ \texttt{in}\ (\texttt{case}\ t\ alts))$

Fig. 3. Normal-order rules of L_S

A weak head normal form (WHNF) is an expression $(\texttt{let}\ x_1 = t_1, \ldots, x_n = t_n\ \texttt{in}\ v)$, where v is a value. A term s *converges*, iff $s \xrightarrow{*} v$ for some WHNF v by a normal order reduction, denoted as $s \Downarrow$.

Two terms s, t are related by contextual preorder, $s \leq_c t$, iff $\forall C : C[s] \Downarrow \Longrightarrow C[t] \Downarrow$, and s, t are contextually equivalent, $s \sim_c t$, iff $s \leq_c t$ and $t \leq_c s$.

4.2 Equivalence of L and L_S

In the following theorem we use the measure $len_M(\cdot)$ also for L_S-evaluations.

Theorem 4.1. *Let s be an expression. Then $s \Downarrow_{L_S} \ \Leftrightarrow s \Downarrow_L$.*
Let $M \subseteq \{(case), (seq), (lbeta), (choice)\}$. Then for every L-evaluation U of s there is an L_S-evaluation U' of t with $len_M(U) = len_M(U')$, and for every L_S-evaluation U' of t there is an L-evaluation U of s with $len_M(U) = len_M(U')$.

Proof. We show the directions separately:

Let s be an expression with $s \Downarrow_{L_S}$. Then we show that $s \Downarrow_L$. We assume $s_0 \xrightarrow{no,L_S,k} s_k$ where s_k is a L_S-WHNF and $k \geq 0$. We show $s_0 \Downarrow_L$ by induction on k. For the base case s_0 is a L_S-WHNF and obviously also a L-WHNF. For the induction step let $k > 0$ and $s_0 \xrightarrow{no,L_S} s_1 \xrightarrow{no,L_S,k-1} s_k$. From the induction hypothesis we have $s_1 \Downarrow_L$. If $s_0 \xrightarrow{no,L_S,choice} s_1$ then Lemma 3.3 and $s_1 \Downarrow_L$ imply $s_0 \Downarrow_L$. Otherwise, $s_0 \Downarrow_L$ follows from the lemmas in subsection 3.4 showing that all L_S-reduction except for $(choice)$ are contained in $\sim_{c,L}$.

Let $M = M_{max} \setminus \{(\text{lll})\}$ and let U be a normal order L reduction for s. We show by induction on $len_M(U)$ that there exists a normal order L_S-reduction U_{L_S} for s.

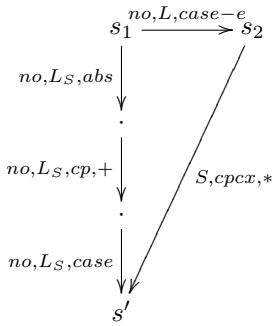
For the base case let $len_M(U) = 0$. Then U consists only of (lll)-reductions. We show by a sub-induction on the number of (lll)-reductions of U that there exists a normal-order L_S-reduction for s. If U is empty then s is an L-WHNF. Then either s is also an L_S-WHNF, or it can be reduced to an L_S-WHNF by a $\xrightarrow{(no,L_S,abs)}$ if necessary and subsequent $\xrightarrow{(no,L_S,cp)}$-reductions. If U is not empty then every (lll)-reduction in U is also a (no, L_S, lll) normal order reduction and the claim follows obviously.

Now we treat the case $len_M(U) > 0$. Let $U = s \xrightarrow{no,L,lll,*} s_1 \xrightarrow{U'}$, where the first reduction of U' is not an (lll)-reduction. For the construction of U_{L_S} we will use as prefix $s \xrightarrow{no,L,lll,*} s_1$ since these reduction sequence are also a normal-order L_S-reduction sequence. Obviously $len_M(U') = len_M(U)$.

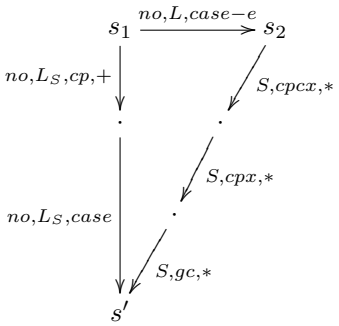
Now let $s_1 \xrightarrow{no,L,a} s_2$ be the first reduction of U', where a is not an (lll)-reduction. If $a \in \{(\text{lbeta}), (\text{case-c}), (\text{seq-c}), (\text{choice})\}$ then $\xrightarrow{no,L,a}$ is also a normal order reduction for L_S and the measure len_M is strictly decreased. Using the induction hypothesis we derive the demanded normal-order L-reduction-sequence.

Otherwise, i.e. if the L-reduction exploits bindings over several variable-variable-bindings, the following diagrams show how normal-order L_S-reduction sequence can be derived for the first reduction:

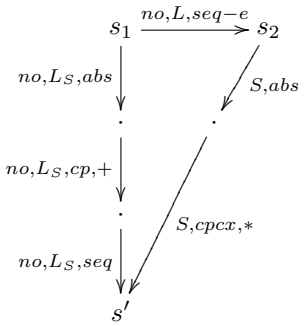
$x = (c\ t_1\ t_2)$
`in case` x ...



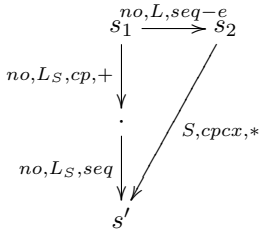
$x = (c\ x_1\ x_2)$
`in case` x ...



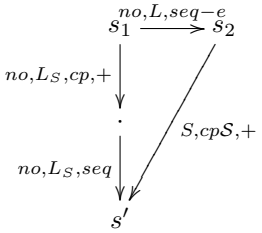
$x = (c\ t_1\ t_2)$
`in seq` x ...



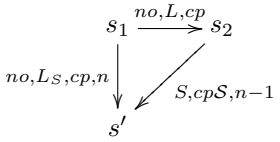
$x = (c\ x_1\ x_2)$
`in seq` x ...



$x = (\lambda x.s)$
`in seq` x ...



$x = \lambda z.s, y = x$
`in` $R[y]$ ...



Let $U' = s_1 \xrightarrow{no,L,a} s_2\ U''$. Then by the lemmas in subsection 3.4, in particular the claims on the reduction lengths, we derive a normal-order L-reduction $U_{s'}$ for s' with $len_M(U_{s'}) \leq len_M(U'')$. Note that only the reductions (abs) and (gc) modify the number of (lll)-reductions.

Since $len_M(U'') < len_M(U') = len_M(U)$ we can apply the induction hypothesis to $U_{s'}$ and hence have a normal order L_S reduction for s' and using the diagrams above also a normal order L_S reduction for s.

Corollary 4.2. *The contextual preorders and contextual equivalence of L and L_S are identical, i.e. $\leq_{c,L} = \leq_{c,L_S}$ and $\sim_{c,L} = \sim_{c,L_S}$.*

Proof. This follows from Theorem 4.1 and since the contexts of L and L_S are identical.

5 The Approximation Calculus

We define the approximation calculus L_A related to L_S, such that L_A is a SHOCS (see [MSS06,MSS07]), the contextual preorders of the two calculi are equivalent, and such that the result that simulation implies contextual preorder can be applied to L_A (see [MSS06,MSS07]). The calculus L_A is related to the one in [Man05a,Man05b], however, extended and modified, insofar as all the let-shuffling rules from L are omitted, and constructors, a `case` and `seq` are added.

5.1 Rules of the Approximation Calculus

The syntax is the same as for L and L_S, where in addition a non-constructor constant $\circledcirc$ is permitted as expression. We will also use *non-closing surface-contexts*, which are surface contexts such that the hole is not in the scope of a let-binder of the context. We denote the class of contexts as $\mathcal{NS}$. The canonical operators in the sense of the higher-order abstract syntax are λ and the constructors. A *pseudo-value* in this calculus is built from constructors, abstractions, and $\circledcirc$, and *var-pseudo-value* is built from constructors, abstractions, variables and $\circledcirc$, and an *answer* is a var-pseudo-value that is not the $\circledcirc$-constant and not a variable. The rules of L_A are defined in figure 4.

A term s converges w.r.t. L_A: $t \Downarrow_A$ iff $t \xrightarrow{A,\mathcal{NS},*} v$, where v is an answer. Two terms s, t are related by contextual preorder, $s \leq_{c,A} t$, iff $\forall C : C[s] \Downarrow_A \Longrightarrow C[t] \Downarrow_A$, and s, t are contextually equivalent, $s \sim_{c,A} t$, iff $s \leq_{c,A} t$ and $t \leq_{c,A} s$.

5.2 Equivalence of L and L_A

We have to argue that the addition of $\circledcirc$ to the syntax of L is not significant. Let $L_{\circledcirc}$ be the calculus L on the expressions and contexts that may contain $\circledcirc$,

(cpA)	$(\texttt{let}\ x = v\ \texttt{in}\ s)$	$\to_A\ s[v/x]$	if v is a var-pseudo-value
(stop)	s	$\to_A\ \circledcirc$	if s is not an answer
(choicel)	$(\texttt{choice}\ s\ t)$	$\to_A\ s$	
(choicer)	$(\texttt{choice}\ s\ t)$	$\to_A\ t$	
(seq)	$(\texttt{seq}\ v\ t)$	$\to_A\ t$	if v is an answer
(betaA)	$((\lambda x.s)\ v)$	$\to_A\ s[v/x]$	if v is a var-pseudo-value
(caseA)	$(\texttt{case}\ (c\ v_1 \dots\ v_n)$ of $\dots (c\ x_1 \dots\ x_n \to t) \dots)$	$\to_A\ t[\overline{v_i}/\overline{x_i}]$	if v_i are var-pseudo-values

The L_A-reduction $\xrightarrow{A,\mathcal{NS}}$ is defined as any of the reductions in non-closing surface contexts.

Fig. 4. Reductions in L_A: Approximation Reductions

but with the same definition of reductions rules, normal-order reduction and WHNFs. The corresponding convergence is denoted using the suffix $L_{\circledcirc}$. Let ϕ be the translation of expression from $L_{\circledcirc}$ to expressions of L which replaces every occurrence of $\circledcirc$ by Ω. Then the following holds:

Proposition 5.1. *Let s be an $L_{\circledcirc}$-expression. Then $s \Downarrow_{L_{\circledcirc}} \iff \phi(s) \Downarrow_L$. Moreover, for all $L_{\circledcirc}$-expressions s, t the relation $s \leq_{c,L_{\circledcirc}} t \iff \phi(s) \leq_{c,L_{\circledcirc}} \phi(t) \iff \phi(s) \leq_{c,L} \phi(t)$ holds.*

Proof. The main argument is that $\circledcirc$ does not converge, since it is not a constructor. If $s \Downarrow_{L_{\circledcirc}}$ then the unwind and the normal-order reduction are the same (related by ϕ) whatever is replaced for $\circledcirc$, hence $\phi(s) \Downarrow_L$. If $\phi(s) \Downarrow_L$, then the subexpression Ω is not touched by the reduction rules, since $R[\Omega]$ diverges for every reduction context R. Hence $s \Downarrow_{L_{\circledcirc}}$. The equivalence of the preorders follows from the fact that $\phi(C[s]) = \phi(C)[\phi(s)]$ for all $L_{\circledcirc}$-contexts C and $L_{\circledcirc}$-expressions s.

For the following, we drop the distinction between L and $L_{\circledcirc}$, and assume that the calculus $L_{\circledcirc}$ is used. We can use all the lemmas on equivalences and lengths of reductions w.r.t. L also for L_A.

Now we show the equivalence of L and L_A by proving that $s \Downarrow \iff s \Downarrow_A$ for all s for all expression s.

Theorem 5.2. *For all expressions s: $s \Downarrow_A \implies s \Downarrow \iff s \Downarrow_{L_S}$.*

Proof. We use the lemmas available for the calculus L and show that the L_A-reduction rules retain contextual equivalence w.r.t. L:

Let $s \Downarrow_A$. The reduction starts as $s \xrightarrow{a,L_A} s_1$ with $s_1 \Downarrow_A$. Then we show by induction on the length of the L_A-reduction of s_1 that there is also an L-evaluation of s_1.

The base case is trivial. For the induction hypothesis assume that $s \xrightarrow{a,L_A} s_1$ and that $s_1 \Downarrow_L$.

- If a is a (cpA)-reduction, then it is translated as a sequence of $\xrightarrow{\mathcal{S},(abs)}$, $\xrightarrow{\mathcal{S},lll}$, $\xrightarrow{\mathcal{S},(cpcx)}$, $\xrightarrow{\mathcal{S},(cpx)}$, $\xrightarrow{\mathcal{S},(cp)}$ and $\xrightarrow{\mathcal{S},(cpbot)}$-reductions with subsequent $\xrightarrow{(gc)}$-reductions. The lemmas in subsection 3.4 show that all these reductions retain the L-equivalence class and hence that $s \Downarrow_L$ holds.
- If the reduction a is a (stop)-reduction, then $s \geq_c s_1$ by Lemma 3.20, and hence $s \Downarrow_L$.
- If the reduction a is a (choice)-reduction, then $s \geq_c s_1$ by Lemma 3.3, and hence $s \Downarrow_L$.
- If the reduction a is a (seq)-reduction, then it is a (seq-c)-reduction in L, and hence $s \Downarrow_L$ by Lemma 3.2.
- If the reduction a is a (case)-reduction, then it is a (case-c)-reduction in L, and hence $s \Downarrow_L$ by Lemma 3.2.
- If the reduction a is a (betaA)-reduction, then it can be simulated by an (lbeta) and then treated as a (cpA)-reduction, hence $s \Downarrow_L$ by the above arguments and by Lemma 3.2.

We conclude that $s \Downarrow_{L_S}$.
The second claim follows from Theorem 4.1.

5.3 Approximation-Convergence Implies L-Convergence

The direction $s \Downarrow_L \implies s \Downarrow_A$ for all s requires to find an approximation reduction from s to an answer, given a normal-order reduction of s, in particular, the reduction must be free of (let)-shuffling rules. How this can be achieved is demonstrated for a small non-deterministic calculus in [Man05a,Man05b]. We will show a specialized claim, where $s \Downarrow_A$ is shown for a restricted approximation reduction.

5.4 A Less Non-Deterministic Approximation Reduction

The goal now is to show that for every s with $s \Downarrow_{L_S}$, there is an approximation reduction of s to an answer. It is more convenient to show that there is a special kind of approximation reduction; an AP-reduction, defined as follows.

Definition 5.3. *Let AP be a class of contexts, where the hole can be reached by a label-shift of S using the following rules exhaustively:*

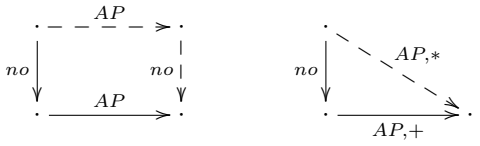
$$
\begin{array}{lll}
(s\ t)^S & \to (s^S\ t) & \\
(v^S\ t) & \to (v\ t^S) & \textit{if } v \textit{ is a var-pseudo-value} \\
(\texttt{let}\ x = s\ \texttt{in}\ t)^S & \to (\texttt{let}\ x = s^S\ \texttt{in}\ t) & \\
(\texttt{seq}\ s\ t)^S & \to (\texttt{seq}\ s^S\ t) & \\
(\texttt{case}\ s\ alts)^S & \to (\texttt{case}\ s^S\ alts) & \\
(c\ s_1\ \ldots s_n)^S & \to (c\ s_1\ \ldots s_i^S \ldots s_n) & \textit{if } s_i \textit{ is not a var-pseudo-value,} \\
 & & \textit{and for all } j < i\textit{: } s_j \textit{ is a var-pseudo-value.}
\end{array}
$$

Note that AP-contexts are non-closing surface-contexts.
We define $\xrightarrow{AP}$ as a $\xrightarrow{A}$-reduction in an AP-context; and $s \Downarrow_{AP}$ iff $s \xrightarrow{AP,*} v$

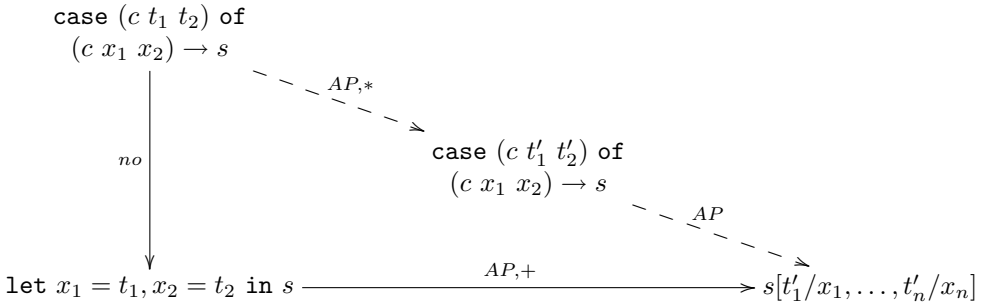
for some answer v.

Now we argue that the following "commuting diagrams" hold between normal-order reductions (L_S, no) and successive $\xrightarrow{AP}$-reduction:

Lemma 5.4. *A complete set of commuting diagrams for a single L_S-evaluation with an AP-reduction is:*



Proof. By inspecting the AP-reductions and the cases of an overlap. We show a critical overlap:



Now we can show the existence of an AP-evaluation for L_S, no-converging expressions.

Proposition 5.5. *Let s be an expression that has an L_S-evaluation. Then there is also an AP-evaluation of s to an answer.*

Proof. The proof is by induction on the length of an L_S-evaluation. Assume that $s \xrightarrow{L_S,k} t$ where t is a WHNF. If $k = 0$ then s is a WHNF, and hence there is an AP-reduction to an answer by removing a potential let-environment using (cpA), where perhaps several $\circledcirc$-reductions may be necessary to enable a (cpA)-reduction; e.g. $(\texttt{let } x = (s_1\ s_2) \texttt{ in } (c\ x)) \xrightarrow{AP,(stop)} (\texttt{let } x = \circledcirc \texttt{ in } (c\ x)) \xrightarrow{AP,(cpA)} (c\ \circledcirc)$.
For the induction step, let $k > 0$, i.e. $s \xrightarrow{L_S} s' \xrightarrow{L_S,k-1} t$. We apply the induction hypothesis to $s' \xrightarrow{L_S,k-1} s$ and obtain an AP-reduction of s' to an answer. We show that for every sequence $s \xrightarrow{L_S} s' \xrightarrow{AP,*} v$ where v is an answer, there exists an AP-reduction of s to an answer using a sub-induction on the length of the AP-reduction $s' \xrightarrow{AP,*} v$. The base case is that s' is an answer. Then $s \xrightarrow{L_S} s'$ must be a (*choice*) or (*seq*) reduction. If it is a

$\xrightarrow{L_S,(choice)}$-reduction, then this reduction can also be used as AP-reduction. If it is a $\xrightarrow{L_S,(seq)}$-reduction, then it is of the form $(\texttt{seq}\ v\ w) \xrightarrow{L_S} w$, where v is a value, and w is an answer. It is obvious that $v \xrightarrow{AP,(stop),*} v'$ such that v' is an answer, hence $(\texttt{seq}\ v\ w) \xrightarrow{AP,(stop),*} (\texttt{seq}\ v'\ w) \xrightarrow{AP} w$ is an AP-reduction to an answer. Otherwise, if the length of $s' \xrightarrow{AP,*} v$ is greater than 0, we apply a commuting diagram from Lemma 5.4: If the first diagram is applicable we have $s \xrightarrow{AP} s'' \xrightarrow{L_s} s''' \xrightarrow{AP,*} v$. Applying the induction hypothesis to $s'' \xrightarrow{L_s} s''' \xrightarrow{AP,*} v$ we have an AP-reduction to an answer for s'' and also for s. If the second diagram is applicable then the AP-reduction for s is derived after the application of the diagram.

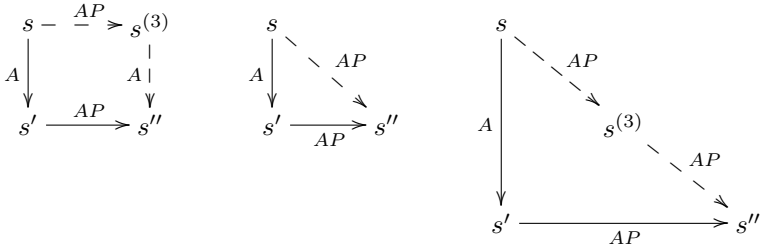
As a summary, we obtain the following:

Theorem 5.6. *Let s be an expression.*
Then $s \Downarrow_L \iff s \Downarrow_S \iff s \Downarrow_A \iff s \Downarrow_{AP}$

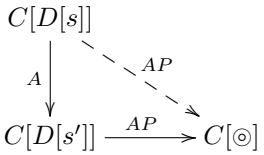
To use AP-reductions instead of A-reductions for simulation-proofs, we need a stronger claim about the relationship between AP-reduction and A-reductions:

Theorem 5.7. *Let s be an expression. Then for all answers v: $s \xrightarrow{AP,*} v \iff s \xrightarrow{A,*} v$*

Proof. The direction $s \xrightarrow{AP,*} v \implies s \xrightarrow{A,*} v$ is trivial, since every AP-reduction is an A-reduction.
For the other direction, we first compute the commuting diagrams for all situations $s \xrightarrow{AP} s' \xrightarrow{A} s''$: The possibilities are:



This can be verified by inspecting the AP-reduction and A-reductions and the cases of an overlap. We show a critical overlap:



The third diagram above occurs if the A-reduction $s \xrightarrow{A} s'$ turns into an AP-reduction.

The proof of $s \xrightarrow{A,*} v \implies \xrightarrow{AP,*} v$ is by induction on the length of the reduction $s \xrightarrow{A,*} v$. If the length is 0, then s is an answer, and the AP-reduction is also of length 0. Now let the length be at least 1. Then we treat the case $s \xrightarrow{A} s' \xrightarrow{A,*} v$, where by induction hypothesis, $s' \xrightarrow{AP,*} v$. In this situation we make a sub-induction on the length of the AP-reduction. If the length of $s' \xrightarrow{AP,*} v$ is 0, then s' is an answer, and then $s \xrightarrow{A} s'$ must also be an AP-reduction. If the length of $s' \xrightarrow{AP,*} v$ is at least 1, then we have $s' \xrightarrow{AP} s'' \xrightarrow{AP,*} v$, and one of the commuting diagrams above is applicable to $s \xrightarrow{A} s' \xrightarrow{AP} s''$. If the first diagram is applicable, we obtain $s \xrightarrow{AP} s^{(3)} \xrightarrow{A} s'' \xrightarrow{AP,*} v$ and by the induction hypothesis, we get an AP-reduction $s^{(3)} \xrightarrow{AP,*} v$, and hence $s \xrightarrow{AP,*} v$. If the second or third diagram applies, then we immediately have a reduction $s \xrightarrow{AP,*} v$.

6 Simulation and Examples

Since all reductions of the approximation calculus L_A of the previous section match a rule-format of a SHOCS (see [MSS07,MSS06], and L and L_A have identical contextual preorder, we can apply the main theorem in [MSS07] which tells us that similarity in L_A is a proof tool for contextual equivalence in L_A and hence in L. Note that the alternative in a case for a zero-ary constructor is encoding like a unary constructor. Instead of the general approximation reduction, we can use the more deterministic AP-reduction (see Proposition 5.5 and Theorem 5.7), which has a smaller set of possibilities of reductions for expressions. The non-deterministic choices are whether the reduction (choice) selects the left or right argument, and whether to use (stop) or another reduction.

The definition of simulation in the calculus L_A, as an instance of the general definition in [MSS07], is as follows:

Definition 6.1. *The* behavioral preorder $\leq_b$ *is the greatest relation on closed expressions satisfying the following two conditions:*

- *For all closed expressions s, t: If $s \Downarrow_{AP} \lambda x.s'$, then $t \Downarrow_{AP} \lambda x.t'$ and for all closed pseudo-values r: $s'[r/x] \leq_b t'[r/x]$.*
- *For all closed expressions s, t: If $s \Downarrow_{AP} (c\ s_1\ \ldots\ s_n)$, then $t \Downarrow_{AP} (c\ t_1\ \ldots\ t_n)$ and for all i: $s_i \leq_b t_i$.*

The relation $\leq_b^o$ on all expressions is defined as the open extension on pseudo-values, i.e. for all $s \leq_b^o t \iff$ for all pseudo-values $r_1, \ldots r_n$: $s[r_1/x_1, \ldots, r_n/x_n] \leq_b t[r_1/x_1, \ldots, r_n/x_n]$, where $\mathcal{FV}(s,t) = \{x_1, \ldots, x_n\}$.

Similarity $\simeq_b$ is defined as $\simeq_b \ := \ \leq_b^o \cap \geq_b^o$.

Theorem 6.2. *The relation $\leq_b^o \ \subseteq \ \leq_c$ holds for the calculi L and L_S.*

Proof. This follows from Theorem 5.6 and the main theorem in [MSS07].

We define some combinators and recursive functions used in the following examples.

$$
\begin{array}{ll}
K & := \lambda xy.x \\
Y & := \lambda f.(\lambda x.f(x\ x))\ (\lambda x.f(x\ x)) \\
Y_2 & := \lambda f.(\lambda xy.f(x\ y\ y))\ (\lambda xy.f(x\ y\ y))\ (\lambda xy.f(x\ y\ y)) \\
map & := Y\ (\lambda m.\lambda f.\lambda xs.\mathtt{case}_{\mathtt{list}}\ xs\ \mathtt{of} \\
 & \qquad\qquad\qquad\qquad (\mathtt{Nil} \to \mathtt{Nil}) \\
 & \qquad\qquad\qquad\qquad (\mathtt{Cons}\ y\ ys) \to \mathtt{Cons}\ (f\ y)\ (m\ f\ ys)) \\
repeat & := Y\ (\lambda r.\lambda x.\mathtt{Cons}\ x\ (r\ x))
\end{array}
$$

It is easy to see that $Y\ K \simeq_b Y_2\ K$ using AP-reduction. The main argument is that all the answers in every recursion of the similarity-test are abstractions. The definition of $\lesssim_b$ shows that the terms $Y\ K$ and $Y_2\ K$ are mutually similar, and hence $(Y\ K) \sim_c (Y_2\ K)$.

A further example where $s \simeq_b t$ is easy to verify using simulation, and hence $s \simeq_c t$ holds, are the two terms

$$
\begin{array}{l}
s_0 := (\mathtt{Cons}\ (choice\ 0\ 1)\ \mathtt{Nil}) \\
t_0 := (choice\ (\mathtt{Cons}\ 0\ \mathtt{Nil})\ (\mathtt{Cons}\ 1\ \mathtt{Nil}))
\end{array}
$$

Note that similarity-testing requires s to AP-reduce to an answer, hence the expression $(choice\ 0\ 1)$ will be reduced before comparisons. Further contextual equivalences that can immediately be derived using bisimilarity are commutativity, idempotency and associativity of `choice` seen as a binary operator.

A third example for the simulation method is the proof of equivalence of

$$
\begin{array}{l}
s_1 := Y\ (\lambda r.\mathtt{Cons}\ (choice\ 0\ 1)\ r) \\
t_1 := map\ (\lambda f.f\ 0)\ (repeat\ (\lambda z.choice\ 0\ 1))
\end{array}
$$

which are lists with elements $0, 1$, where the selection can be made independently for every element.

Using simulation and co-induction these two expressions can be shown as contextually equivalent. We show a particular reduction for each expression.

$$
\begin{array}{ll}
 & (\lambda f.(\lambda x.(f\ (x\ x)))\ (\lambda x.(f\ (x\ x))))\ \ (\lambda r.\mathtt{Cons}\ (choice\ 0\ 1)\ r) \\
\xrightarrow{AP} & (\lambda x.(F\ (x\ x)))\ (\lambda x.(F\ (x\ x))) \\
 & \qquad \text{where } F = (\lambda r.\mathtt{Cons}\ (choice\ 0\ 1)\ r) \\
\xrightarrow{AP} & (F\ (G\ G)) \\
 & \qquad \text{where } G = (\lambda x.(F\ (x\ x))) \\
\xrightarrow{AP} & (F\ \circledcirc) \\
\xrightarrow{AP} & \mathtt{Cons}\ (choice\ 0\ 1)\ \circledcirc \\
\xrightarrow{AP} & Cons\ 0\ \circledcirc
\end{array}
$$

In the reduction of the second expression we assume that *map* and *repeat* are defined as above.

$$
\begin{array}{ll}
 & \mathit{map}\ (\lambda f.f\ 0)\ (\mathit{repeat}\ (\lambda z.(\mathit{choice}\ 0\ 1))) \\
\to & M\ (\lambda f.f\ 0)\ (\mathit{repeat}\ (\lambda z.(\mathit{choice}\ 0\ 1))) \\
 & \quad \text{where } M = (\lambda x.(M_1\ (x\ x)))\ (\lambda x.(M_1(x\ x))) \\
 & \quad \text{and } M_1 \text{ is the case-abstraction of the definition} \\
\xrightarrow{AP,*} & M_2\ (\lambda f.f\ 0)\ (\mathit{repeat}\ (\lambda z.(\mathit{choice}\ 0\ 1))) \\
 & \quad \text{where } M_2 = (M_1\ \circledcirc) \\
\xrightarrow{AP,*} & M_3\ (\lambda f.f\ 0)\ (\mathit{repeat}\ (\lambda z.(\mathit{choice}\ 0\ 1))) \\
 & \quad \text{where } M_3 = \lambda f'.\lambda xs.\mathtt{case}\ xs\ \mathtt{of}\ (\mathtt{Nil} \to \mathtt{Nil}) \\
 & \qquad ((\mathtt{Cons}\ y\ ys) \to \mathtt{Cons}\ (f'\ y)\ (\circledcirc\ f'\ ys))) \\
\xrightarrow{AP,*} & \mathtt{case}\ L\ \mathtt{of}\ (\mathtt{Nil} \to \mathtt{Nil})\ ((\mathtt{Cons}\ y\ ys) \to \mathtt{Cons}\ (F'\ y)\ (\circledcirc\ F'\ ys))) \\
 & \quad \text{where } L = (\mathit{repeat}\ F'') \\
 & \quad \text{and } F' = (\lambda f.f\ 0) \text{ and } F'' = (\lambda z.(\mathit{choice}\ 0\ 1)) \\
\xrightarrow{AP,*} & \mathtt{case}\ L'\ \mathtt{of}\ (\mathtt{Nil} \to \mathtt{Nil})\ ((\mathtt{Cons}\ y\ ys) \to \mathtt{Cons}\ (F'\ y)\ (\circledcirc\ F'\ ys))) \\
 & \quad \text{where } L' = (\mathtt{Cons}\ F''\ (\circledcirc\ F'')) \\
\xrightarrow{AP,*} & \mathtt{case}\ L''\ \mathtt{of}\ (\mathtt{Nil} \to \mathtt{Nil})\ ((\mathtt{Cons}\ y\ ys) \to \mathtt{Cons}\ (F'\ y)\ (\circledcirc\ F'\ ys))) \\
 & \quad \text{where } L'' = (\mathtt{Cons}\ F''\ (\circledcirc\ F'\ ys)) \\
\xrightarrow{AP,*} & \mathtt{Cons}\ ((\lambda z.(\mathit{choice}\ 0\ 1))\ 0)\ (\circledcirc\ F'\ ys) \\
\xrightarrow{AP} & \mathtt{Cons}\ (\mathit{choice}\ 0\ 1)\ (\circledcirc\ F'\ ys) \\
\xrightarrow{AP,*} & \mathtt{Cons}\ 0\ \circledcirc
\end{array}
$$

For every AP-evaluation of the first expression, the second one has a corresponding AP-evaluation.

A further example are the lists

$$
\begin{array}{l}
s_2 := \mathit{repeat}\ (\mathit{choice}\ 0\ 1) \\
t_2 := \mathit{choice}\ (\mathit{repeat}\ 0)\ (\mathit{repeat}\ 1)
\end{array}
$$

It is not hard to see that $s_1 \not\sim_c s_2$, since the latter corresponds either to an infinite lists of 0's or of 1's, and where contexts are easily constructed that distinguish s_1 and s_2.

We show the respective AP-reductions. The AP-reduction of $(Y\ F)$ results in:

$$
\begin{array}{ll}
Y\ F & \xrightarrow{AP,*} F^n(F'\ F') \\
 & \xrightarrow{AP} F^n\ \circledcirc \\
 & \quad \text{where } F^1\ a = Fa \text{ and } F^{n+1}\ a = F\ (F^n\ a) \\
 & \quad \text{and } F' = \lambda x.F\ (x\ x)
\end{array}
$$

Thus the possibilities of AP-reductions of s_2 can be illustrated as follows:

$$
\begin{array}{ll}
 & \mathit{repeat}\ (\mathit{choice}\ 0\ 1) \\
\xrightarrow{AP,*} & (F^n\ \circledcirc\ (\mathit{choice}\ 0\ 1) \\
 & \text{where } F = (\lambda r.\lambda x.\mathtt{Cons}\ x\ (r\ x)) \\
\xrightarrow{AP} & (F^n\ \circledcirc\ 0) \qquad \text{or } (F^n\ \circledcirc\ 1) \\
\xrightarrow{AP,*} & (\mathtt{Cons}\ 0\ (\mathtt{Cons}\ 0 \ldots (\mathtt{Cons}\ 0\ \circledcirc))) \qquad \text{or } (\mathtt{Cons}\ 1\ (\mathtt{Cons}\ 1 \ldots (\mathtt{Cons}\ 1\ \circledcirc)))
\end{array}
$$

The reduction of t_2 results in $(\mathtt{Cons}\ 0\ (\mathtt{Cons}\ 0 \ldots (\mathtt{Cons}\ 0\ \circledcirc)))$ for *repeat* 0 and $(\mathtt{Cons}\ 1\ (\mathtt{Cons}\ 1 \ldots (\mathtt{Cons}\ 1\ \circledcirc)))$ for *repeat* 1, hence t_2 can reduce to $(\mathtt{Cons}\ 0\ (\mathtt{Cons}\ 0 \ldots (\mathtt{Cons}\ 0\ \circledcirc)))$ or to $(\mathtt{Cons}\ 1\ (\mathtt{Cons}\ 1 \ldots (\mathtt{Cons}\ 1\ \circledcirc)))$.

This shows that s_2, t_2 have the same answers using AP-reductions, hence they are bisimilar, and hence contextually equivalent.

7 Conclusion and Further Research

We have show the equivalence of three different reductions in a non-deterministic call-by-need calculus with case, constructors and let, which shows that there is an interesting prototypical instance for a calculus that is covered by SHOCS, where the simulation method as described in [MSS07] can be used advantageously.

Future work will be to extend the simulation method to non-deterministic call-by-need calculi with a letrec (as in Haskell), and to extend the contextual preorder to also include must-convergence, which is required for a fully appropriate contextual equivalence in non-deterministic calculi.

Acknowledgement

We thank David Sabel for his help, for discussions and reading versions of the paper; and also the anonymous referees of the SHOCS-paper for their hints and demands.

References

Abr90. Samson Abramsky. The lazy lambda calculus. In D. A. Turner, editor, *Research Topics in Functional Programming*, pages 65–116. Addison-Wesley, 1990.

AF97. Z. M. Ariola and M Felleisen. The call-by-need lambda calculus. *J. Funct. Programming*, 7(3):265–301, 1997.

AFM+95. Z. M. Ariola, M. Felleisen, J. Maraist, M. Odersky, and P. Wadler. A call-by-need lambda calculus. In *Principles of Programming Languages*, pages 233–246, San Francisco, California, 1995. ACM Press.

Gor99. Andrew D. Gordon. Bisimilarity as a theory of functional programming. *Theoret. Comput. Sci.*, 228(1-2):5–47, October 1999.

How89. D. Howe. Equality in lazy computation systems. In *4th IEEE Symp. on Logic in Computer Science*, pages 198–203, 1989.

Man05a. Matthias Mann. Congruence of bisimulation in a non-deterministic call-by-need lambda calculus. *Electron. Notes Theor. Comput. Sci.*, 128(1):81–101, 2005.

Man05b. Matthias Mann. *A Non-Deterministic Call-By-Need Lambda Calculus: Proving Similarity a Precongruence by an Extension of Howe's Method to Sharing.* PhD thesis, Dept. of Computer Science and Mathematics, J.W.Goethe-Universität, Frankfurt, Germany, 2005.

MOW98. John Maraist, Martin Odersky, and Philip Wadler. The call-by-need lambda calculus. *J. Funct. Programming*, 8:275–317, 1998.

MSC99. Andrew K. D. Moran, David Sands, and Magnus Carlsson. Erratic fudgets: A semantic theory for an embedded coordination language. In *Coordination '99*, volume 1594 of *Lecture Notes in Comput. Sci.*, pages 85–102. Springer-Verlag, 1999.

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

MSS07. Matthias Mann and Manfred Schmidt-Schauß. How to prove similarity a precongruence in a broad class of non-deterministic call-by-need lambda calculi, 2007. submitted.

Pit97. Andrew D. Pitts. Operationally-based theories of program equivalence. In *Semantics and Logics of Computation*. Cambridge University Press, 1997.

SSS07. Manfred Schmidt-Schauß and David Sabel. On generic context lemmas for lambda calculi with sharing. Frank report 27, Inst. f. Informatik, J.W.Goethe-University, Frankfurt, 2007.

SSSS04. Manfred Schmidt-Schauß, Marko Schütz, and David Sabel. On the safety of Nöcker's strictness analysis. Frank report 19, Inst. f. Informatik, J.W.Goethe-University, Frankfurt, 2004.

SSSS07. Manfred Schmidt-Schauß, Marko Schütz, and David Sabel. Safety of Nöcker's strictness analysis. *J. Funct. Programming*, 00(99), 2007. accepted for publication.