Towards Sharing in Lazy Computation Systems

Technical Report Frank-18

Matthias Mann

Institut für Informatik
Johann Wolfgang Goethe-Universität
Postfach 11 19 32
D-60054 Frankfurt, Germany
mann@cs.uni-frankfurt.de

December 28, 2004

Abstract. Work on proving congruence of bisimulation in functional programming languages often refers to [How89,How96], where Howe gave a highly general account on this topic in terms of so-called "lazy computation systems". Particularly in implementations of lazy functional languages, sharing plays an eminent role. In this paper we will show how the original work of Howe can be extended to cope with sharing.
Moreover, we will demonstrate the application of our approach to the call-by-need $\boldsymbol{\lambda}$-calculus $\boldsymbol{\lambda}_{ND}$ which provides an erratic non-deterministic operator `pick` and a non-recursive `let`. A definition of a bisimulation is given, which has to be based on a further calculus named $\boldsymbol{\lambda}_{\approx}$, since the naïve bisimulation definition is useless. The main result is that this bisimulation is a congruence and contained in the contextual equivalence. This might be a step towards defining useful bisimulation relations and proving them to be congruences in calculi that extend the $\boldsymbol{\lambda}_{ND}$-calculus.

Table of Contents

1 Introduction

Contextual equivalence due to [Mor68] is of great interest for $\boldsymbol{\lambda}$-calculi, notably in the important field of correct program transformations. Since it does not, unlike the notion of convertibility, directly depend on the reduction rules of a calculus but rather discriminates terms by their behaviour, usually termination (cf. [Mil77]) in all contexts, it provides a separate justification for the reduction rules in addition to a huge amount of meaningful equations.

But establishing contextual equivalence is rarely straightforward, so the technique of bisimulation (cf. [Mil71,Par81]) recently has attracted interest in the context of functional programming (cf. [Abr90,San91,Gor94,Gor99]). However, in order to apply bisimulation as a tool for showing correctness of program transformations, it has to be a congruence. Proving this is in general a complex effort as e.g. the work in [Abr90,How89,How96] demonstrates. For non-deterministic call-by-need $\boldsymbol{\lambda}$-calculi in particular, it seems to us that up to now there has not been much research in this respect.

So the aim of this paper is twofold. First we will introduce the notions and abstractions which are necessary to adapt the method of Howe (cf. [How89,How96]) to call-by-need $\boldsymbol{\lambda}$-calculi. Secondly, we will demonstrate the feasibility of our approach in that we apply it to set up a bisimulation for a non-deterministic call-by-need $\boldsymbol{\lambda}$-calculus and prove it a congruence. In this calculus, the usual technique to define bisimulation, namely by reducing terms to a weak head normal form and applying these weak head normal forms to arbitrary fresh arguments, will not work, as we will see.

1.1 Outline of the Paper

Therefore, the structure of the paper is as follows. In section 2 we will treat the abstract approach to syntax and evaluation by the so-called lazy computation systems and develop general criteria for bisimulation being a congruence. The non-deterministic call-by-need $\boldsymbol{\lambda}$-calculus $\boldsymbol{\lambda}_{ND}$ with an erratic non-deterministic operator `pick` and a non-recursive `let` is then introduced in section 3. It intentionally is a very basic calculus, since it should act as a starting point for further studies. An example will show that, as mentioned before, a definition of bisimulation in $\boldsymbol{\lambda}_{ND}$ working directly with `let`-environments is problematic. Hence in section 4 with the $\boldsymbol{\lambda}_{\approx}$-calculus we work out a way to prune the evaluation in environments at any arbitrary finite depth. We accomplish this by adapting the reduction rules so that bisimulation may be based upon reduction to pure abstractions without a surrounding `let`-environment while recording every possible outcome of the original environment. This enables us to define bisimulation and prove it a congruence by an extension of Howe's method in [How89,How96], i.e. that the so-called "precongruence candidate" is preserved under reduction. The achievement of section 5 then is to establish the link between the $\boldsymbol{\lambda}_{ND}$- and the $\boldsymbol{\lambda}_{\approx}$-calculus in that the contextual equivalences agree. Figure 1 outlines the dependencies of the major proof steps.

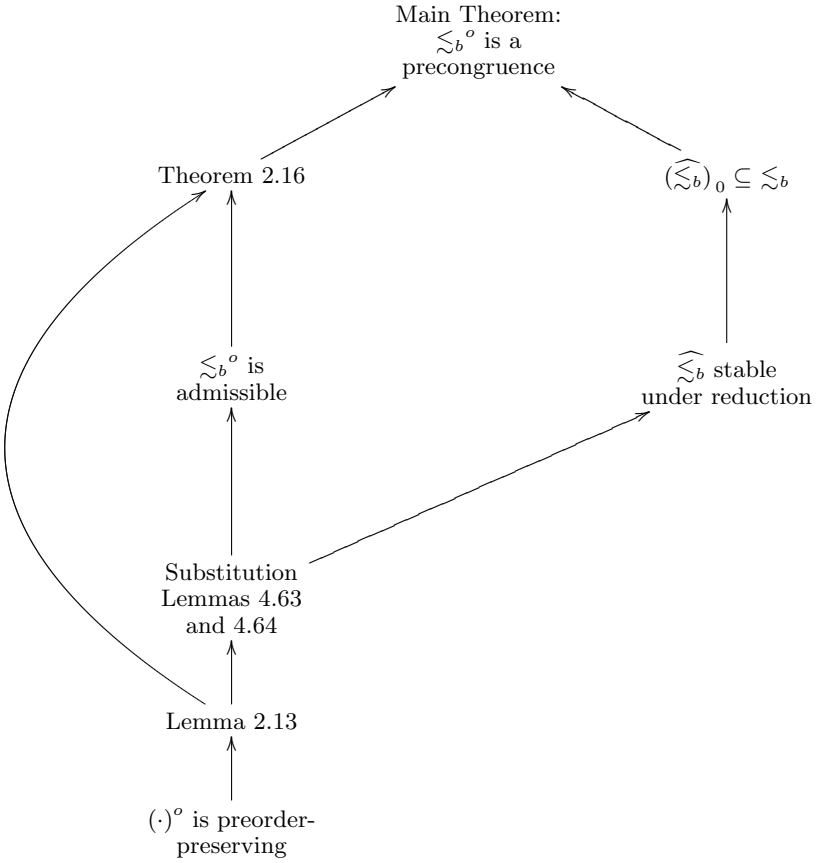

Fig. 1. Structure of the proof

1.2 Related Work

When introducing a non-deterministic construct into a programming language or, e.g. a $\boldsymbol{\lambda}$-calculus, a number of questions have to be clarified. Apart from the classification of non-determinism as e.g. in [SS92], we consider a major issue the decision what kind of terms should be permitted to be copied.

Non-determinism in languages without sharing, i.e. retaining a copying (β)-rule like e.g. [Ong93,San,dP95,LP00], is completely different from our work because it will distinguish $\lambda x.(x + x)$ from $\lambda x.(2 * x)$. Likewise is the situation with [Bou94], since, as usual in explicit substitution-calculi (cf. [ACCL91]) too, substitutions are distributed over applications and hence duplicated.

The deterministic call-by-need calculi of [AFM$^+$95a,AF97,MOW98] realise explicit sharing using a `let`-construct (or special syntactic entities, as is the case in [AF97]) and restrict copying to abstractions. However, their equational theory is based on convertibility rather than on contextual equivalence.

Thus the calculi in [MSC99a,KSS98,Kut99,SS03] which all provide a non-deterministic choice, sharing and contextual equivalence roughly represent the direction of our investigations, though there are a few differences. Since these papers do not discuss bisimulation, it seemed sensible to carry out our studies in a rather elementary calculus, which also should increase readability.

Hence, like the work of [KSS98,Kut99], the $\boldsymbol{\lambda}_{ND}$-calculus only has a non-recursive `let`, whereas the calculi in [MSC99a,SS03] provide recursive bindings. Furthermore, like [KSS98,Kut99] but in contrast to [MSC99a,SS03], the $\boldsymbol{\lambda}_{ND}$-calculus neither has data constructors nor a `case`.

1.3 Mathematical Preliminaries

We assume the reader to be familiar with substitutions on terms, frequently specified in a way like $[s/x]$ or $\{x \mapsto s\}$. With $\boldsymbol{dom}(\cdot)$ we denote the *domain* of a substitution and with $\boldsymbol{rng}(\cdot)$ the *range*. The *restriction* of a substitution σ to a subset D of its domain is written $\sigma|_D$ and interpreted as for usual mappings.

2 Lazy Computation Systems

Since we would like to be able to transfer our results concerning the bisimulation in a non-deterministic call-by-need lambda calculus to different calculi with sharing, we need to specify the syntax and operational semantics of the language in an abstract way. There are quite a few approaches to this subject, spanning e.g. [KvOvR93,MN98,How89]. We decided to stick to Howe's approach since he has made a great deal in formalising conditions when bisimilarity is a precongruence for a large class of languages. Furthermore, by applying it to improvement theory in [San91] even though not regarding sharing, Sands already has demonstrated that it is possible to further enhance Howe's approach.

In this section, we therefore first introduce the concept of a lazy computation language, before we treat preorders and prove criteria sufficient for them to be precongruences. In section 2.3 we then define reduction and evaluation, i.e. operational semantics for lazy computation languages, thus yielding lazy computation systems.

2.1 Language

As in [How89], we characterise a lazy computation language by its operators forming the terms of the language. But here, we will omit the partitioning of the operators into canonical and non-canonical ones.

Definition 2.1. *A signature* $\mathcal{L} = (O, \alpha)$ *consisting of a set* O *of* operators *with their* arity *given by a mapping* $\alpha : O \to \{\langle k_1, \ldots, k_n\rangle \mid \forall 1 \leq i \leq n \in \mathbb{N} : k_i \in \mathbb{N}_0\}$ *is called a* lazy computation language.

Remark 2.2. Note that the arity of an operator is a sequence of integers as we do without *separate kinds* of variables as e.g. in [How96] and [San97], since we concentrate on *lazy* evaluation.

Definition 2.3. *Given a (possibly infinite) set V of variables we inductively define the sets ${}^{i}\mathcal{T}(\mathcal{L})$ as follows:*

- $V \subseteq {}^{0}\mathcal{T}(\mathcal{L})$
- *If* $t \in {}^{0}\mathcal{T}(\mathcal{L})$ *and* $x_1, \ldots, x_n \in V$ *are distinct then* $x_1, \ldots, x_n.t \in {}^{n}\mathcal{T}(\mathcal{L})$
- *If* $\tau \in O$ *with arity* $\alpha(\tau) = \langle k_1, \ldots, k_n \rangle$ *and* $t_j \in {}^{k_j}\mathcal{T}(\mathcal{L})$ *for* $j \in \{1, \ldots, n\}$ *then* $\tau(t_1, \ldots, t_n) \in {}^{0}\mathcal{T}(\mathcal{L})$.

Now the terms *of the language $\mathcal{L}$ are given by $\mathcal{T}(\mathcal{L}) = {}^{0}\mathcal{T}(\mathcal{L})$.*

The elements of the sets ${}^{n}\mathcal{T}(\mathcal{L})$ for $n > 0$ are called *operands*, the constitution of an operand $\overline{x}.t$ binds all *free occurrences* in t of variables in $\overline{x}$. A term is *closed* if all of its variables are bound, otherwise it is called *open*. The set of closed terms is denoted by $\mathcal{T}_0(\mathcal{L})$, and if the language $\mathcal{L}$ is clear from context we will simply write $\mathcal{T}$ and $\mathcal{T}_0$ in the remainder of this paper.

We consider terms and operands syntactically equal up-to alpha-renaming, i.e. renaming of bound variables. We denote this syntactic equality with $\equiv_\alpha$ or simply with $\equiv$, e.g. we write $\overline{x}.s \equiv \overline{y}.t$, if there is a renaming of the bound variables (including the possibly empty tuples $\overline{x}$ and $\overline{y}$ respectively) such that the operands become the same, i.e. $s[\overline{z}/\overline{x}] \equiv t[\overline{z}/\overline{y}]$ for fresh variables $\overline{z}$.

Convention 1 (Variable Convention). Since we deal with terms modulo $\equiv_\alpha$, we may assume that in every term t all bound variables are distinct from each other and the free variables. We will extend this convention also to terms that result from transformations of other terms.

Example 2.4. The λ-language consisting of the operators $O = \{\lambda, @\}$ with its arities $\alpha(\lambda) = \langle 1 \rangle$ and $\alpha(@) = \langle 0, 0 \rangle$ forms a lazy computation language where e.g. $@(\lambda(x.s), t)$ is an open term which would usually be denoted by $(\lambda x.s)\ t$ and $\lambda(y.@(y, y))$ is a closed term, written $\lambda y.yy$ in the usual notation.

Example 2.5. A simple `let`-language could be modelled by the set of operators $O = \{\lambda, \texttt{let}, @\}$ with λ and $@$ having arities as in example 2.4 and $\alpha(\texttt{let}) = \langle 1, 0 \rangle$. Thus, the non-recursive `let` $x = t$ `in` s could then be expressed as $\texttt{let}(x.s, t)$.

A *context* is a term with a hole and may be defined like in definition 2.3. We write $\mathcal{C}$ for the set of all such single-hole contexts, as we sometimes also consider *multi-contexts*, i.e. contexts with multiple holes which must clearly be told apart from each other.

Definition 2.6. *Let $\mathcal{D}$ be denoting some set of contexts. Then the sets $\mathcal{D}^*$, $\mathcal{D}^+$, $\mathcal{D}^k$, $\mathcal{D}^{m \vee n}$ are defined by the corresponding symbols for every $D \in \mathcal{D}$:*

$$
\begin{aligned}
D^0 &::= [\,] & \qquad D^{k+1} &::= D[D^k] \\
D^+ &::= D \mid D[D^+] & \qquad D^{m \vee n} &::= D^m \mid D^n \\
D^* &::= D^0 \mid D^+
\end{aligned}
$$

In the following we will usually write e.g. $D^{m \vee n}$ denoting a context $D \in \mathcal{D}^{m \vee n}$.

2.2 Preorders and the Precongruence Candidate

We will now state some general properties for preorders in a lazy computation language. Therefore assume the set $\mathcal{T}$ of terms to be fixed for the remainder of this section. As is known, the key idea of Howe's method for proving bisimulation a congruence is to define a so-called "precongruence candidate" first, which by definition is a reflexive and operator-respecting but not necessarily transitive relation. Showing then that this precongruence candidate coincides with the underlying preorder results in the desired precongruence. So we first clarify the notion of a precongruence.

Definition 2.7. *A relation $\eta \subseteq \mathcal{T}^2$ is called* operator-respecting *if and only if $\overline{a}_i \ \eta \ \overline{b}_i$ implies $\tau(\overline{a}_i) \ \eta \ \tau(\overline{b}_i)$ for all operands a_i, b_i and operators τ.*

Another word for operator-respecting is *compatible*.

Definition 2.8. *A preorder $\eta \subseteq \mathcal{T}^2$ is a* precongruence *if and only if for all terms $s, t \in \mathcal{T}$ we have $s \ \eta \ t \implies \forall C \in \mathcal{C} : \ C[s] \ \eta \ C[t]$.*

As easily could be shown by induction, a preorder is a precongruence if and only if it is operator-respecting. Hence for preorders, we use both notions synonymously in the following. We also extend a relation η to operands by $\overline{x}.s \ \eta \ \overline{y}.t$ if and only if $[\overline{z}/\overline{x}] \ \eta \ t[\overline{z}/\overline{y}]$ for fresh variables $\overline{z}$ holds, i.e., if a consistent renaming of the bound variables exists that relates the underlying terms. The *restriction* of a preorder $\eta \subseteq \mathcal{T}^2$ to closed terms is defined by $\eta_{\,0} = \eta \cap {\mathcal{T}_0}^2$ as usual. From this notion, some easy consequences may be drawn immediately, e.g. that $\eta_1 \subseteq \eta_2$ implies $(\eta_1)_{\,0} \subseteq (\eta_2)_{\,0}$ by monotonicity of set-intersection.

As mentioned before, a relation $\eta \subseteq {\mathcal{T}_0}^2$ must *not* be extended to open terms via the "for-all-substitutions"-notion since this breaks sharing of terms. An alternative for this extension η^o of η to open terms has to be formulated distinctly for every concrete instance of an lcs, which we would like to postpone until later and therefore define only a conceptional notion here.

Definition 2.9. *A mapping $(\cdot)^o : \mathcal{T}_0 \times \mathcal{T}_0 \to \mathcal{T} \times \mathcal{T}$ is said to* extend (relations) to open terms *and η^o is called the* extension of a relation η to open terms.

The definition of the precongruence candidate, which is as usual, works without specific properties, so we just assume $(\cdot)^o$ extending $\eta \subseteq {\mathcal{T}_0}^2$ to open terms.

Definition 2.10. *Let $\eta \subseteq {\mathcal{T}_0}^2$ be a preorder, then define $\widehat{\eta} \subseteq \mathcal{T}^2$ by induction*

- *$x \ \widehat{\eta} \ b$ if $x \ \eta^o \ b$ and $x \in V$.*
- *$\tau(\overline{a}_i) \ \widehat{\eta} \ b$ if $\overline{a}_i \ \widehat{\eta} \ \overline{a}'_i$ and $\tau(\overline{a}'_i) \ \eta^o \ b$ for some $\tau \in O$ and operands $\overline{a}'_i$.*

Remark 2.11. Note that for every operator ζ of arity $\alpha(\zeta) = \langle\rangle$, i.e. ζ has no operands, and every term $t \in \mathcal{T}$ we have $\zeta \ \widehat{\eta} \ t \iff \zeta \ \eta^o \ t$ by the above.

For most of the fundamental properties, it is sufficient to demand that η^o is a preorder whenever η is, hence the following definition.

Definition 2.12. *If $(\cdot)^o$ extends relations to open terms, it is called* preorder-preserving *if and only if for every relation $\eta \subseteq {\mathcal{T}_0}^2$ on closed term the following holds: η is a preorder implies that also η^o is a preorder.*

Proving the analogue to [How96, lemma 3.1] reveals the necessity of $(\cdot)^o$ being preorder-preserving.

Lemma 2.13. *If $\eta \subseteq {\mathcal{T}_0}^2$ and its extension $\eta^o \subseteq \mathcal{T}^2$ both are preorders then*

1. *$\widehat{\eta}$ is reflexive*
2. *$\widehat{\eta}$ and $\widehat{\eta}_0$ are operator-respecting*
3. *$\eta^o \subseteq \widehat{\eta}$*
4. *$\widehat{\eta} \circ \eta^o \subseteq \widehat{\eta}$*

Proof. 1. Let $a \in \mathcal{T}$ be an arbitrary but fixed term, then we show $a \mathrel{\widehat{\eta}} a$ by induction on the structure of a:
   - If $a \in V$ is a variable, we have $a \mathrel{\widehat{\eta}} a$ from the reflexivity of η^o and the base case of definition 2.10.
   - If $a \equiv \tau(\overline{a}_i)$ for some operator τ and operands a_i, we have $a_i \mathrel{\widehat{\eta}} a_i$ from the induction hypothesis and $\tau(\overline{a}_i) \mathrel{\eta^o} \tau(\overline{a}_i)$ from the reflexivity of η^o. So, by definition 2.10, we may compose this to $\tau(\overline{a}_i) \mathrel{\widehat{\eta}} \tau(\overline{a}_i)$.

2. We assume $\overline{a}_i \mathrel{\widehat{\eta}} \overline{b_i}$ and have to show $\tau(\overline{a}_i) \mathrel{\widehat{\eta}} \tau(\overline{b_i})$ for an arbitrary but fixed operator τ. By reflexivity of η^o we have $\tau(\overline{b_i}) \mathrel{\eta^o} \tau(\overline{b_i})$ and from definition 2.10 we conclude $\tau(\overline{a}_i) \mathrel{\widehat{\eta}} \tau(\overline{b_i})$.
   As a consequence if $\tau(\overline{a}_i), \tau(\overline{b_i}) \in \mathcal{T}_0$ are closed, we also have $\tau(\overline{a}_i) \mathrel{\widehat{\eta}_0} \tau(\overline{b_i})$ from $\overline{a}_i \mathrel{\widehat{\eta}} \overline{b_i}$, which implies that $\widehat{\eta}_0$ is operator-respecting too.

3. Assume $a \mathrel{\eta^o} b$ for arbitrary but fixed $a, b \in \mathcal{T}$ and show $a \mathrel{\widehat{\eta}} b$ by induction on the structure of a:
   - If $a \in V$ is a variable, we have $a \mathrel{\widehat{\eta}} b$ directly from $a \mathrel{\eta^o} b$ and the base case of definition 2.10.
   - If $a \equiv \tau(\overline{a}_i)$ for some operator τ and operands a_i, we have $a_i \mathrel{\widehat{\eta}} a_i$ from property (1), the reflexivity of $\widehat{\eta}$. By definition 2.10 then, we may conclude $\tau(\overline{a}_i) \mathrel{\widehat{\eta}} b$.

4. Assume $a \mathrel{\widehat{\eta}} b$ and $b \mathrel{\eta^o} c$, so according to definition 2.10 we have to distinguish the following two cases:
   - a is a variable, then for $a \mathrel{\widehat{\eta}} b$ also $a \mathrel{\eta^o} b$ must hold and thus the proposition by transitivity of η^o and property (3).
   - a has the form $\tau(\overline{a}_i)$, then there is a $\tau(\overline{a}'_i)$ such that $\overline{a}_i \mathrel{\widehat{\eta}} \overline{a}'_i$ and $\tau(\overline{a}'_i) \mathrel{\eta^o} b$. By transitivity of η^o we also have $\tau(\overline{a}'_i) \mathrel{\eta^o} c$ thus $\tau(\overline{a}_i) \mathrel{\widehat{\eta}} c$. □

In order to prove the essential theorem on the criteria for η^o being a precongruence, we have to demand the following properties.

Definition 2.14. *Let $\eta \subseteq {\mathcal{T}_0}^2$ be a relation, then an extension $\eta^o \subseteq \mathcal{T}^2$ of η to open terms is* admissible *only if all of the following conditions are met:*

1. *$(\cdot)^o$ is preorder-preserving*
2. *$(\eta^o)_0 = \eta$*
3. *$\forall \nu : \ \nu \subseteq \eta \implies \nu^o \subseteq \eta^o$*

4. $\widehat{\eta} \subseteq (\widehat{\eta}_0)^o$

Lemma 2.15. *Let* $\eta \subseteq {\mathcal{T}_0}^2$ *be a preorder on closed terms. Then every admissible extension* η^o *contains* η*, i.e.* $\eta \subseteq \eta^o$*.*

Proof. Assume a preorder $\eta \subseteq {\mathcal{T}_0}^2$ on closed terms and η^o admissible. Then from property (2) of definition 2.14 we have $(\eta^o)_0 = \eta$. Since $\nu_0 = \nu \cap {\mathcal{T}_0}^2 \subseteq \nu$ holds for every relation $\nu \subseteq \mathcal{T}^2$ on terms, we thus have $\eta = (\eta^o)_0 = \eta^o \cap {\mathcal{T}_0}^2 \subseteq \eta^o$.

In [How96, Theorem 3.1], the formulation of the following theorem requires a substitution lemma which we cannot provide at this point, since we have made only a few assumptions about η^o. But with having abstracted out the essential properties in definition 2.14, the main result remains provable independently.

Theorem 2.16. *Let* $\eta \subseteq {\mathcal{T}_0}^2$ *be a preorder and* η^o *its admissible extension to open terms. Then the following are equivalent.*

1. η^o *is a precongruence*
2. $\widehat{\eta} \subseteq \eta^o$
3. $\widehat{\eta}_0 \subseteq \eta$

Proof. The claim is shown by a chain of implications.

"1 $\Longrightarrow$ 2": Assuming η^o to be a precongruence and $a\ \widehat{\eta}\ b$, we show $a\ \eta^o\ b$ by induction on the definition of $\widehat{\eta}$.
- If $a \in V$ is a variable, the only possibility is $a\ \eta^o\ b$.
- If $a \equiv \tau(\overline{a}_i)$ for some operator τ and operands a_i, there must have been operands a_i' such that $a_i\ \widehat{\eta}\ a_i'$ for every i and $\tau(\overline{a}_i')\ \eta^o\ b$. From the induction hypothesis we may conclude $\overline{a}_i\ \eta^o\ \overline{a}_i'$, which in turn means $\tau(\overline{a}_i)\ \eta^o\ \tau(\overline{a}_i')$ and furthermore $\tau(\overline{a}_i)\ \eta^o\ b$ since η^o is a precongruence.

"2 $\Longrightarrow$ 3": If $\widehat{\eta} \subseteq \eta^o$ then we immediately have $\widehat{\eta}_0 \subseteq (\eta^o)_0 = \eta$, since η^o is admissible.

"3 $\Longrightarrow$ 1": So it remains to show that η^o is a precongruence under the assumption $\widehat{\eta}_0 \subseteq \eta$. Since η^o is admissible, from $\widehat{\eta}_0 \subseteq \eta$ we have $(\widehat{\eta}_0)^o \subseteq \eta^o$ by monotonicity, i.e. property (3) of definition 2.14. In conjunction with property (4) there, this becomes $\widehat{\eta} \subseteq (\widehat{\eta}_0)^o \subseteq \eta^o$. Hence by property (3) of lemma 2.13 we have $\widehat{\eta} = \eta^o$, thus η^o is operator-respecting. □

The plan is to establish the last set inclusion $\widehat{\eta}_0 \subseteq \eta$ for which we will show that the precongruence candidate $\widehat{\eta}$ is stable under reduction. Therefore we now turn our attention to reduction and evaluation in lazy computation languages.

2.3 Reduction and Evaluation

We now come to the specification of the operational semantics of a lazy computation language. As is known, this could be done in two ways, i.e. by a big step evaluation relation and by a small-step reduction. In this section, we will depict reduction and evaluation to the extent to which they may be applicable to the abstract notion of a lazy computation language.

Definition 2.17 (Lazy Computation System). *A lazy computation language $\mathcal{L}$ together with a binary relation $\Downarrow \subseteq \mathcal{T}^2$ on terms is called a* lazy computation system *(lcs for short) iff $\Downarrow$ is reflexive and satisfies the following condition:*

$$a \Downarrow v \implies \forall v' : \ (v \Downarrow v' \iff v \equiv v') \tag{2.1}$$

We then call $\Downarrow$ evaluation *and simply write $a \Downarrow$ if there exists some b — which we call an* answer *— such that $a \Downarrow b$, and $a \not\Downarrow$ if there is no such b.*

Remark 2.18. Note that the condition (2.1) does not enforce determinism, as $a \Downarrow v$ may hold for different v. But in some way, an answer v may be seen as an "end-point" of the $\Downarrow$-relation.

The previous definition reflects the approach to "big-step" operational semantics. We will now also declare "small-step" reduction semantics. Unlike [Bar84, p. 50], we do not require a *reduction relation* to be operator-respecting in general. Instead, we will use the notation $s \xrightarrow{C,\ a} t$ to indicate that for the reduction from s to t the reduction rule (a) is used in the context C.

Definition 2.19 (Reduction). *A* reduction relation *is a binary relation on terms which may be specified by distinct* reduction rules. *If $s, t \in \mathcal{T}$ are terms so that s reduces to t by the rule (a) we write $s \xrightarrow{a} t$ and speak of a* top-level *reduction. If $C \in \mathcal{C}$ is a context and s reduces to t at top-level by rule (a) then we may* use rule (a) in the context C *which we will denote by $C[s] \xrightarrow{C,\ a} C[t]$. In both cases, s is called a* redex *which is an acronym for "reducible expression".*

Sometimes we will emphasise a top-level reduction by writing $s \xrightarrow{[\,],\ a} t$ and if the reduction rule (a) is omitted it may be any of the respective calculus. If a reduction is superscripted with a context class instead of a distinct context, e.g. $s \xrightarrow{\mathcal{D},\ a} t$, this means a reduction by rule (a) may be performed in any context which belongs to the class $\mathcal{D}$, i.e.

$$\exists D \in \mathcal{D} : \ s \equiv D[s'] \ \wedge \ t \equiv D[t'] \ \wedge \ s' \xrightarrow{a} t' \tag{2.2}$$

Moreover, the transitive and reflexive-transitive closure of $\rightarrow$ will be denoted as usual by $\rightarrow^+$ and $\rightarrow^*$ respectively, while we write $\rightarrow^k$ and $\rightarrow^{<k}$ for a reduction of exactly and less than k steps respectively. Similarly with e.g. $\rightarrow^{k \geq m}$ we will denote a reduction of $k \geq m$ steps. Sometimes further labels will be attached to the symbol $\rightarrow$ in order to specify certain properties of the reduction.

In [How89], Howe defines the notion of a *canonical* operator, i.e. terms whose top-level operator is canonical, are not reduced further. Hence they represent the answers in the sense of definition 2.17. But since we want to model sharing, we must take into account that a term may carry around an *environment* with bindings of variables to be shared as discussed in [AFM+95a] or [MOW98]. As a consequence, a clear distinction between canonical and non-canonical operators cannot be kept up. So it may be an option to describe the notion of an answer by the syntactic structure of the terms, e.g. given by some predicate, say $\mathfrak{F}(\cdot)$,

we then may relate $\Downarrow$ and $\to^*$ in the way $s \Downarrow t \iff s \to^* t \wedge \mathfrak{F}(t)$. It is then often interesting and necessary to argue about different reductions that may lead to the same result. Therefore we first introduce the notion of reduction and conversion sequences.

Definition 2.20 (Reduction and Conversion Sequences). *Let $n \in \mathbb{N}$ be a non-negative integer, $t_1, \ldots, t_{n+1} \in \mathcal{T}$ be terms, $C_{a_1}, \ldots, C_{a_n}$ be contexts and $a_1, \ldots, a_n$ be reduction rules (with possibly further labels attached).*

Then a sequence $(\langle t_i, a_i, C_{a_i}\rangle)_i$ is a reduction sequence (of length n) *if and only if $t_i \xrightarrow{C_{a_i},\ a_i} t_{i+1}$ for every $1 \leq i \leq n$ holds. If for every $1 \leq i \leq n$, we have either $t_i \xrightarrow{C_{a_i},\ a_i} t_{i+1}$ or $t_i \xleftarrow{C_{a_i},\ a_i} t_{i+1}$, then we call $(\langle t_i, a_i, C_{a_i}\rangle)_i$ a* conversion sequence (of length n).

One use for the notion of reduction sequence is the definition of divergence.

Definition 2.21 (Divergence). *We say that a term $s \in \mathcal{T}$* diverges*, denoted by $s \Uparrow$, if there is an infinite reduction sequence starting with s.*

It is important to note that $s \Uparrow$ in general does not imply $s \not\Downarrow$ and vice versa.

Example 2.22. Supposing a nondeterministic construct `pick` which may reduce to either of its arguments, for the term $\texttt{pick}\ \mathbf{I}\ \mathbf{\Omega}$ both relations $\Downarrow$ and $\Uparrow$ hold.

Relevant parts of the following proofs will use transformations on reduction and conversion sequences respectively. One way to describe such transformations is by meta-rules on reduction sequences in form of diagrams.

2.3.1 Reduction Diagrams The technique of complete sets of commuting and forking diagrams respectively is well-established and has yet been proven useful in several situations (cf. [KSS98,Kut99,SS03,Sab03]). We will now expand its definitions to lazy computation systems.

Definition 2.23 (Transformation Rule). *A* transformation *of a conversion sequence $(\langle s_i, a_i, C_{a_i}\rangle)_i$ of length m consists of a conversion sequence $(\langle t_j, b_j, C_{b_j}\rangle)_j$ of length n such that $s_1 \equiv t_1$ and $s_{m+1} \equiv t_{n+1}$.*

A transformation rule *describes a set of possible transformations of conversion sequences in an abstract manner using a notation of the form*

$$\xrightarrow{C_{a_1},\ a_1} \cdot \xleftarrow{C_{a_2},\ a_2} \cdot \ldots \cdot \xleftarrow{C_{a_{m-1}},\ a_{m-1}} \cdot \xrightarrow{C_{a_m},\ a_m} \rightsquigarrow$$

$$\xleftarrow{C_{b_1},\ b_1} \cdot \xrightarrow{C_{b_2},\ b_2} \cdot \ldots \cdot \xrightarrow{C_{b_{n-1}},\ b_{n-1}} \cdot \xleftarrow{C_{b_n},\ b_n}$$

A transformation rule like the above is called applicable to a prefix (suffix) *of a conversion sequence $(\langle s_i, a_i, C_{a_i}\rangle)_i$ of lengh $k \geq m$ if there are terms $t_1, \ldots, t_{n+1} \in \mathcal{T}$ such that $(\langle t_j, b_j, C_{b_j}\rangle)_j$ is a transformation of the prefix-sequence $(\langle s_i, a_i, C_{a_i}\rangle)_{i \leq m}$ (suffix-sequence $(\langle s_i, a_i, C_{a_i}\rangle)_{i > k-m}$ respectively).*

Definition 2.24 (Complete Set of Commuting Diagrams). *A* set of commuting diagrams *for a reduction* $\xrightarrow{\mathcal{D},\ a}$ *w.r.t. two sets* B, O *of reductions, is a set of transformation rules of the form*

$$\xrightarrow{\mathcal{D},\ a} \cdot \xrightarrow{C_{b_1},\ b_1} \cdot \ldots \cdot \xrightarrow{C_{b_l},\ b_l} \quad \rightsquigarrow$$
$$\xrightarrow{C_{b_1},\ b_1} \cdot \ldots \cdot \xrightarrow{C_{b'_m},\ b'_m} \cdot \xrightarrow{C_{a_1},\ a_1} \cdot \ldots \cdot \xrightarrow{C_{a_n},\ a_n}$$

for reduction sequences, where the following conditions are met:

1. $\xrightarrow{C_{b_i},\ b_i} \subseteq B$ *for every* $1 \leq i \leq l$,
2. $\xrightarrow{C_{b'_i},\ b'_i} \subseteq B$ *for every* $1 \leq i \leq m$ *and*
3. $\xrightarrow{C_{a_j},\ a_j} \subseteq O \cup \xrightarrow{\mathcal{D},\ a}$ *for every* $1 \leq j \leq n$

A set of commuting diagrams is called complete *if and only if for every reduction sequence of the form*

$$s_0 \xrightarrow{\mathcal{D},\ a} s_1 \xrightarrow{C_{b_1},\ b_1} \ldots \xrightarrow{C_{b_l},\ b_l} s_{l+1}$$

such that $l > 0$ *and* s_{l+1} *is an answer but* s_0 *is not, there is one transformation rule applicable to a prefix of the sequence.*

By means of a complete set of commuting diagrams for a reduction $\xrightarrow{\mathcal{D},\ a}$ one can show that reductions of this kind may be moved to the end of every reduction sequence consisting only of reductions in B, while possibly auxiliary reduction from O are introduced.

Similarly is with complete sets of forking diagrams, which may guarantee that the application of a reduction does not change termination behaviour, i.e. show some kind of confluence property (cf. [BKvO98]).

Both, complete sets of commuting and forking diagrams, should adhere to the condition that the composition of diagrams from the set terminates. This makes induction applicable.

Definition 2.25 (Complete Set of Forking Diagrams). *A* set of forking diagrams *for a reduction* $\xrightarrow{\mathcal{D},\ a}$ *w.r.t. two sets* B, O *of reductions, is a set of transformation rules of the form*

$$\xleftarrow{C_{b_1},\ b_1} \cdot \ldots \cdot \xleftarrow{C_{b_l},\ b_l} \cdot \xrightarrow{\mathcal{D},\ a} \quad \rightsquigarrow$$
$$\xrightarrow{C_{a_1},\ a_1} \cdot \ldots \cdot \xrightarrow{C_{a_n},\ a_n} \cdot \xleftarrow{C_{b'_1},\ b'_1} \cdot \ldots \cdot \xleftarrow{C_{b'_m},\ b'_m}$$

for conversion sequences, where the following conditions are met:

1. $\xrightarrow{C_{b_i},\ b_i} \subseteq B$ *for every* $1 \leq i \leq l$,
2. $\xrightarrow{C_{b'_i},\ b'_i} \subseteq B$ *for every* $1 \leq i \leq m$ *and*

3. $\xrightarrow{C_{a_j},\ a_j} \subseteq O \cup \xrightarrow{\mathcal{D},\ a}$ *for every* $1 \leq j \leq n$

A set of forking diagrams is called complete *if and only if for every conversion sequence of the form*

$$s_0 \xleftarrow{C_{b_1},\ b_1} \ldots \xleftarrow{C_{b_{l-1}},\ b_{l-1}} s_{l-1} \xleftarrow{C_{b_l},\ b_l} s_l \xrightarrow{\mathcal{D},\ a} s_{l+1}$$

such that $l > 0$ *and* s_0 *is an answer but* s_l *is not, there is one transformation rule applicable to a suffix of the sequence.*

If $O = \emptyset$ in the definitions above we simply leave it out and speak of a set of commuting (forking) diagrams w.r.t. B.

2.3.2 Independent Reductions While diagrams are a very useful tool to prove commuting reductions, there are already some simple cases which may be shown in general. Therefore the notion of *disjoint* contexts is necessary.

Definition 2.26. *Two contexts* $C_a, C_b \in \mathcal{C}$ *are called* disjoint *if there is no other context* $C \in \mathcal{C}$ *such that* $C_a \equiv C_b[C]$ *or* $C_a \equiv C_b[C]$ *holds.*

Lemma 2.27. *Let* $C_a, C_b \in \mathcal{C}$ *be two disjoint contexts. Then for all terms* $s_0, s_1, s_2 \in \mathcal{T}$ *and all reductions* a, b *the following holds:*

$$s_0 \xrightarrow{C_a,\ a} s_1 \xrightarrow{C_b,\ b} s_2 \implies \exists s_2', C_a', C_b' :\ s_0 \xrightarrow{C_b',\ b} s_2' \xrightarrow{C_a',\ a} s_2$$

$$s_1 \xleftarrow{C_a,\ a} s_0 \xrightarrow{C_b,\ b} s_2 \implies \exists s_3, C_a', C_b' :\ s_1 \xrightarrow{C_b',\ b} s_3 \xleftarrow{C_a',\ a} s_2$$

Proof. Assume $C_a, C_b \in \mathcal{C}$ to be disjoint contexts, i.e. neither $C_a \equiv C_b[C]$ nor $C_a \equiv C_b[C]$ for some further context $C \in \mathcal{C}$ holds. We recall that $s_0 \xrightarrow{C_a,\ a} s_1$ means that s_0 and s_1 are of the respective forms $s_0 \equiv C_a[t_1]$ and $s_1 \equiv C_a[t_1']$ such that $t_1 \xrightarrow{[\,],\ a} t_1'$ is a top-level reduction. The same holds for C_b, i.e. in the forking case which we will treat first, we have $s_0 \equiv C_b[t_2]$ and $s_2 \equiv C_b[t_2']$ such that $t_2 \xrightarrow{[\,],\ b} t_2'$ is a top-level reduction. Since C_a and C_b are disjoint, there is a two-hole-context $C[[\,]_1, [\,]_2] \in \mathcal{C}$ such that $C_a \equiv C[[\,]_1, t_2]$ and $C_b \equiv C[t_1, [\,]_2]$. Thus $C_a' \equiv C[[\,]_1, t_2']$ and $C_b' \equiv C[t_1', [\,]_2]$ represent the desired contexts, which clearly permit the reductions $s_1 \xrightarrow{C_a',\ a} s_3$ and $s_2 \xrightarrow{C_b',\ b} s_3$.

For the commuting case we have $s_1 \equiv C_b[t_2]$ and $s_2 \equiv C_b[t_2']$ such that $t_2 \xrightarrow{[\,],\ b} t_2'$ is a top-level reduction. Since C_a and C_b are disjoint, there is a two-hole-context $C[[\,]_1, [\,]_2] \in \mathcal{C}$ such that $C_a \equiv C[[\,]_1, t_2]$ and $C_b \equiv C[t_1', [\,]]$ hold. Thus $C_a' \equiv C[[\,]_1, t_2']$ and $C_b' \equiv C[t_1, [\,]_2]$ are contexts, which permit the desired reductions $s_1 \xrightarrow{C_a',\ a} s_3$ and $s_2 \xrightarrow{C_b',\ b} s_3$ independently. □

In the following, we will develop criteria for how to establish complete sets of commuting and forking diagrams respectively. The key idea is, that for reductions which may be performed inside contexts that are closed under composition, it is sufficient to analyse the empty context in a few base cases. The next lemma demonstrates this for the commutation of two reductions.

Lemma 2.28. *Let* $s_0 \xrightarrow{C_a,\ a} s_1 \xrightarrow{C_b,\ b} s_2$ *be a reduction sequence for arbitrary terms* $s_0, s_1, s_2 \in \mathcal{T}$*. If* $C_b \equiv C_a[C]$ *with some context* $C \in \mathcal{C}$ *then the following holds: If for every reduction sequence* $t_0 \xrightarrow{[\ ],\ a} t_1 \xrightarrow{C,\ b} t_2$ *there exists a term* $t_1' \in \mathcal{T}$ *and a context* C' *such that* $t_0 \xrightarrow{C',\ b} t_1' \xrightarrow{[\ ],\ a} t_2$ *then there is also a term* $s_1' \in \mathcal{T}$ *and a context* $C_a' \in \mathcal{C}$ *with* $s_0 \xrightarrow{C_b,\ b} s_1' \xrightarrow{C_a',\ a} s_2$*.*

Proof. Assume $s_0 \xrightarrow{C_a,\ a} s_1 \xrightarrow{C_b,\ b} s_2$ then s_0 and s_1 must be of the respective forms $s_0 \equiv C_a[t_0]$ and $s_1 \equiv C_a[t_1]$ for some terms $t_0, t_1 \in \mathcal{T}$ and a top-level reduction $t_0 \xrightarrow{[\ ],\ a} t_1$. Since $C_b \equiv C_a[C]$ we have $s_2 \equiv C_a[t_2]$ and $t_1 \xrightarrow{C,\ b} t_2$ from $s_1 \xrightarrow{C_b,\ b} s_2$, hence there are t_1' and C' such that $t_0 \xrightarrow{C',\ b} t_1' \xrightarrow{[\ ],\ a} t_2$ by the premise. This reduction can also be performed inside C_a, thus the claim. □

The last step of the proof using composition of contexts is remarkable. Hence we may adopt the technique to reductions inside a class of contexts which is closed under composition. Also the claim could be extended to reduction and conversion sequences of a length greater than two by similar arguments, hence the summary in the following corollary.

Corollary 2.29. *Let* $(\langle s_i, a_i, C_{a_i}\rangle)_i$ *be a reduction (conversion) sequence of length* m *and* $1 \leq k \leq m$ *an index such that for every* $1 \leq i \leq m$ *there are contexts* C'_{a_i} *with* $C_{a_i} \equiv C_{a_k}[C'_{a_i}]$ *for* $i \neq k$ *and* $C'_{a_k} \equiv [\ ]$*. Furthermore, presume that for every conversion sequence* $(\langle p_i, a_i, C'_{a_i}\rangle)_i$ *there is a transformation* $(\langle q_j, b_j, C_{b_j}\rangle)_j$*. Then there exists also a transformation* $(\langle t_j, b_j, C_k[C_{b_j}]\rangle)_j$ *of the original reduction (conversion) sequence.*

2.4 Contextual Precongruence

Throughout this paper we will use the following definition of contextual precongruence which is based on the observation of convergent behaviour of terms. However, in a nondeterminic calculus as we will treat it in the following section, it usually is sensible to incorporate divergence, i.e. the possibility of infinite reduction sequences, as well. Besides that there are quite different views how to accomplish this task, e.g. [Kut99] in contrast to [MSC99a], the reason to omit divergence here is to keep the presentation as simple as possible. Another justification is that it seems feasible to define a separate approximation for divergence.

Definition 2.30 (Contextual Approximation). *Let* $\lesssim_c \subseteq \mathcal{T}^2$*, the* contextual approximation*, be defined by*

$$s \lesssim_c t \iff (\forall C \in \mathcal{C} :\ C[s] \Downarrow \implies C[t] \Downarrow) \tag{2.3}$$

Since $\lesssim_c$ is defined on arbitrary terms using all contexts, and not only closing ones, it is easily checked that $\lesssim_c$ is reflexive, transitive and operator-respecting, thus a precongruence.

Example 2.31. Regard the closed terms $\mathbf{I}$ and $\texttt{pick}\ \mathbf{I}\ \mathbf{\Omega}$ as of example 2.22. If, in the definition of $\lesssim_c$, we also demand the condition $\forall C:\ C[t] \Uparrow \implies C[s] \Uparrow$ which is motivated in [Kut99] by the reason that t should only considered be "better" than s if it does not introduce non-termination, we will have $\mathbf{I} \not\lesssim_c \texttt{pick}\ \mathbf{I}\ \mathbf{\Omega}$.

But for the similarity $\lesssim_b$ we will define later, this would technically complicate matters, since we would then loose the property that the possible outcomes of a reduction are always smaller w.r.t. $\lesssim_b$ than the original term.

The following lemma is useful to transfer the convergent behaviour of terms from one language to another.

Lemma 2.32. *Let $\mathcal{L}_1, \mathcal{L}_2$ be two lazy computation systems, whose term sets are identical, i.e. $\mathcal{T}(\mathcal{L}_1) = \mathcal{T}(\mathcal{L}_2)$, with respective evaluation $\Downarrow_{\mathcal{L}_1}$ and $\Downarrow_{\mathcal{L}_2}$ and contextual approximation $\lesssim_{\mathcal{L}_1,c}$ and $\lesssim_{\mathcal{L}_2,c}$. Then $\lesssim_{\mathcal{L}_1,c} = \lesssim_{\mathcal{L}_2,c}$ if and only if $r \Downarrow_{\mathcal{L}_1} \iff r \Downarrow_{\mathcal{L}_2}$ for every term $r \in \mathcal{T}$ holds.*

Proof. The "only-if"-part simply follows by choosing $C = [\ ]$ in definition 2.30. By symmetry, it suffices to show $\lesssim_{\mathcal{L}_1,c} \subseteq \lesssim_{\mathcal{L}_2,c}$ for the "if"-part. So assume terms $s, t \in \mathcal{T}$, arbitrary but fixed, and a context $C \in \mathcal{C}$ such that $s \lesssim_{\mathcal{L}_1,c} t$ and $C[s] \Downarrow_{\mathcal{L}_2}$ hold. By the premise $\forall r:\ r \Downarrow_{\mathcal{L}_1} \iff r \Downarrow_{\mathcal{L}_2}$, we also have $C[s] \Downarrow_{\mathcal{L}_1}$ and hence $C[t] \Downarrow_{\mathcal{L}_1}$ since $s \lesssim_{\mathcal{L}_1,c} t$ implies $C[s] \lesssim_{\mathcal{L}_1,c} C[t]$. Thus $C[t] \Downarrow_{\mathcal{L}_2}$ by the premise $\forall r:\ r \Downarrow_{\mathcal{L}_1} \iff r \Downarrow_{\mathcal{L}_2}$ again. □

3 A λ-let-calculus with erratic pick

In this section we will introduce the calculus $\boldsymbol{\lambda}_{ND}$ which provides an erratic non-deterministic operator `pick` and a non-recursive `let`. It closely resembles the calculus of [Kut99] except for the difference that the nondeterministic choice is modelled by the syntactic construct `pick` rather than a constant.

The normal-order reduction that will be defined conforms to [Kut99] in that it will respect sharing, i.e. only abstractions may be copied and non-deterministic choices will not be duplicated. Unlike [Kut99] we will only consider contextual equivalence w.r.t. converging behaviour of terms — including divergence behaviour will be devoted to future research.

We will demonstrate that the usual approach of testing terms by just reducing them to weak head normal form and applying these WHNF's to arbitrary arguments is futile in the $\boldsymbol{\lambda}_{ND}$-calculus.

Definition 3.1 ($\boldsymbol{\lambda}_{ND}$-Language). *The language of the calculus $\boldsymbol{\lambda}_{ND}$ is formed by the set $O = \{@, \lambda, \texttt{let}, \texttt{pick}\}$ of operators with the respective arities*

$$\begin{aligned} \alpha(@) &= \langle 0, 0\rangle \\ \alpha(\lambda) &= \langle 1\rangle \\ \alpha(\texttt{let}) &= \langle 1, 0\rangle \\ \alpha(\texttt{pick}) &= \langle 0, 0\rangle \end{aligned}$$

We make the usual convention to write $\lambda xy.s$ as a synonym for the term $\lambda x.(\lambda y.s)$ and call closed terms *combinators*.

Definition 3.2. *We define abbreviations for the following combinators:*

$$\begin{aligned}\mathbf{S} &\equiv \lambda pqr.p\,r\,(q\,r)\\ \mathbf{K} &\equiv \lambda xy.x\\ \mathbf{K2} &\equiv \lambda xy.y\\ \mathbf{I} &\equiv \lambda x.x\\ \\ \mathbf{Y} &\equiv \lambda f.(\lambda x.f\,(x\,x))\,(\lambda x.f\,(x\,x))\\ \mathbf{\Omega} &\equiv (\lambda x.x\,x)\,(\lambda x.x\,x)\end{aligned}$$

We also introduce useful shortcuts for some particular contexts.

Definition 3.3. *Let e denote an expression. We define the following contexts:*

$$\begin{aligned} L_R &\equiv \texttt{let}\ x = e\ \texttt{in}\ [\,] & A_R &\equiv e\,[\,]\\ L_L &\equiv \texttt{let}\ x = [\,]\ \texttt{in}\ e & A_L &\equiv [\,]\,e\end{aligned}$$

Since contexts of the form L_R^* play a major role they will get an own name.

Definition 3.4 (Environments). *Let $\mathcal{E}$ be the class of* environment contexts *or* environments *for short which is defined by the following syntactic rules for the symbol L where e denotes an expression:*

$$L ::= [\,] \ \mid\ \texttt{let}\ x = e\ \texttt{in}\ L$$

Using environments, we may now explain the notion which corresponds to answers as declared in section 2.3, i.e. the normal-order reduction, which we will define later, stops when reaching a weak head normal form.

Definition 3.5 (Weak Head Normal Form). *A term $t \in \mathcal{T}$ is in* weak head normal form *(WHNF for short) iff $t \equiv L[\lambda x.s]$ for some environment $L \in \mathcal{E}$ and term $s \in \mathcal{T}$.*

In contrast to this definition, we will use the notion of a *value* in the remainder of this paper essentially for terms that may be copied, i.e. in the $\boldsymbol{\lambda}_{ND}$-calculus these are abstractions.

Definition 3.6 (Reduction). *We declare the following reduction rules*

$$\texttt{let}\ x = (\texttt{let}\ y = t_y\ \texttt{in}\ t_x)\ \texttt{in}\ s \xrightarrow{llet} \texttt{let}\ y = t_y\ \texttt{in}\ (\texttt{let}\ x = t_x\ \texttt{in}\ s) \quad \text{(llet)}$$

$$(\texttt{let}\ x = t_x\ \texttt{in}\ s)\,t \xrightarrow{lapp} \texttt{let}\ x = t_x\ \texttt{in}\ (s\,t) \quad \text{(lapp)}$$

$$(\lambda x.s)\,t \xrightarrow{lbeta} \texttt{let}\ x = t\ \texttt{in}\ s \quad \text{(lbeta)}$$

$$\texttt{pick}\ s\ t \xrightarrow{nd,left} s \quad \text{(nd, left)}$$

$$\texttt{pick}\ s\ t \xrightarrow{nd,right} t \quad \text{(nd, right)}$$

$$\texttt{let}\ x = \lambda y.r\ \texttt{in}\ D[x] \xrightarrow{cp} \texttt{let}\ x = \lambda y.r\ \texttt{in}\ D[\lambda y.r] \quad \text{(cp)}$$

which should be understood as templates, i.e. they are valid for all possible contexts D, terms $s, t, \ldots$ and variables $x, y, \ldots$ etc. Furthermore, we adopt the variable convention 1, hence in all terms which arise from reduction, we assume all bound variables to be distinct from each other and from the free variables.

We take the union of some reductions, so let

$$\xrightarrow{nd} = \xrightarrow{nd,left} \cup \xrightarrow{nd,right} \tag{nd}$$

$$\xrightarrow{lll} = \xrightarrow{llet} \cup \xrightarrow{lapp} \tag{lll}$$

One important property of the normal-order reduction will be that it only takes place in specific contexts, the reduction contexts.

Definition 3.7 (Reduction Contexts). *The class $\mathcal{R}$ of* reduction contexts *is defined by the following rule for the symbol R:*

$$R ::= L_R^*[A_L^*] \mid L_R^*[\texttt{let } x = A_L^* \texttt{ in } R[x]]$$

The notion of *normal-order* reduction may then intuitively be described as follows. Descend into contexts of the form L_R and subsequently A_L, until (nd), (lapp) or (lbeta) becomes applicable, the case (1) of definition 3.8. If during this process a variable is encountered, follow its binding. Whenever possible, perform (cp) or (llet) for the variable in question, i.e. cases (3) and (4) respectively. Otherwise, in case (2), if the variable is bound to an application, descend into the A_L^*-context as far as possible in order to apply (nd), (lapp) or (lbeta).

Definition 3.8 (Normal-Order Reduction). *A reduction $s \xrightarrow{\mathcal{R},\ a} t$ is called* normal-order *and depicted by $s \xrightarrow{n,\ a} t$ if and only if it matches one of the following.*

1. *If $s \equiv L_R^*[A_L^*[r]]$ and rule (lapp), (lbeta), (nd, left) or (nd, right) is applied to r.*
2. *If $s \equiv L_R^*[\texttt{let } x = A_L^*[r] \texttt{ in } R[x]]$ with some reduction context R such that rule (lapp), (lbeta), (nd, left) or (nd, right) is applied to r.*
3. *If $s \equiv L_R^*[\texttt{let } x = \lambda y.r \texttt{ in } R[x]] \xrightarrow{n,\ cp} L_R^*[\texttt{let } x = \lambda y.r \texttt{ in } R[\lambda y.r]] \equiv t$ by rule (cp) for some reduction context R.*
4. *If rule (llet) is applied as follows:*

$$s \equiv L_R^*[\texttt{let } x = (\texttt{let } y = t_y \texttt{ in } t_x) \texttt{ in } R[x]] \xrightarrow{n,\ llet} L_R^*[\texttt{let } y = t_y \texttt{ in } (\texttt{let } x = t_x \texttt{ in } R[x])] \equiv t$$

The above definition conforms with [Kut99] and slightly differs from [AFM+95b] as discussed in [Kut99, p. 42]. It ensures that the normal-order *redex*, i.e. the sub-expression to be reduced, is unique and, except for the non-deterministic rules, the reduction itself is also unique. Moreover, the present formulation helps to detect overlaps as we will see.

But for now, let us turn to an example which serves two purposes. First, it demonstrates the interplay of non-determinism and sharing in the normal-order reduction. On the other hand, it shows that in $\boldsymbol{\lambda}_{ND}$ an adoption of the usual approach to define bisimilarity, i.e. reduction to WHNF and application to arbitrary arguments, won't match contextual equivalence.

Example 3.9. We assume the Church numerals $1, 2, \ldots$ be defined as usual with addition $+$ evaluating its first argument previous to the second. Let

$$s \equiv \lambda x.(\texttt{pick}\ (\texttt{pick}\ (1+2)\ (1+3))\ (\texttt{pick}\ (4+2)\ (4+3)))$$
$$t \equiv \texttt{let}\ y = \texttt{pick}\ 1\ 4\ \texttt{in}\ \lambda x.(y + \texttt{pick}\ 2\ 3)$$

then obviously s and t both are in WHNF and every application of s and t respectively to some arbitrary argument e leads to the same set $\{3, 4, 6, 7\}$ of answers.

But s and t could be distinguished by contexts since s is an abstraction and may be copied, whereas in t the subterm `pick` 1 4 is shared. Evaluation in the context $C \equiv \texttt{let}\ f = [\,]\ \texttt{in}\ (f\ 1) + (f\ 1)$ illustrates this. For $C[s]$ we may have

$$\begin{aligned}
&\texttt{let}\ f = \lambda x.(\texttt{pick}\ (\texttt{pick}\ (1+2)\ (1+3))\ \ldots)\ \texttt{in}\ (f\ 1) + (f\ 1)\\
\xrightarrow{n,\ cp}\ &\texttt{let}\ f = \lambda x.(\texttt{pick}\ (\texttt{pick}\ (1+2)\ (1+3))\ \ldots)\ \texttt{in}\ (s\ 1) + (f\ 1)\\
\xrightarrow{n}{}^{*}\ &\texttt{let}\ f = \lambda x.(\texttt{pick}\ (\texttt{pick}\ (1+2)\ (1+3))\ \ldots)\ \texttt{in}\ L_1[6] + (f\ 1)\\
\xrightarrow{n,\ cp}\ &\texttt{let}\ f = \lambda x.(\ldots)\ \texttt{in}\ L_1[6] + (s\ 1)\\
\xrightarrow{n}{}^{*}\ &L_s[6+3]
\end{aligned}$$

while for $C[t]$ this is obviously not possible, e.g.

$$\begin{aligned}
&\texttt{let}\ f = (\texttt{let}\ y = \texttt{pick}\ 1\ 4\ \texttt{in}\ \lambda x.(y + \texttt{pick}\ 2\ 3))\ \texttt{in}\ (f\ 1) + (f\ 1)\\
\xrightarrow{n,\ lllet}\ &\texttt{let}\ y = \texttt{pick}\ 1\ 4\ \texttt{in}\ (\texttt{let}\ f = \lambda x.(y + \texttt{pick}\ 2\ 3)\ \texttt{in}\ (f\ 1) + (f\ 1))\\
\xrightarrow{n,\ cp}\ &\texttt{let}\ y = \texttt{pick}\ 1\ 4\ \texttt{in}\ (\texttt{let}\ f = \lambda x.\ldots\ \texttt{in}\ ((\lambda x.(y + \texttt{pick}\ 2\ 3))\ 1) + (f\ 1))\\
\xrightarrow{n}{}^{*}\ &\texttt{let}\ y = 4\ \texttt{in}\ (\texttt{let}\ f = \lambda x.\ldots\ \texttt{in}\ (\texttt{let}\ x = 1\ \texttt{in}\ (y + \texttt{pick}\ 2\ 3)) + (f\ 1))\\
\xrightarrow{n,\ cp}\ &\texttt{let}\ y = 4\ \texttt{in}\ (\texttt{let}\ f = \lambda x.\ldots\ \texttt{in}\ (\texttt{let}\ x = 1\ \texttt{in}\ (4 + \texttt{pick}\ 2\ 3)) + (f\ 1))\\
\xrightarrow{nd,left}\ &\texttt{let}\ y = 4\ \texttt{in}\ (\texttt{let}\ f = \lambda x.\ldots\ \texttt{in}\ L_1[4+2] + (f\ 1))\\
\xrightarrow{n}{}^{*}\ &\texttt{let}\ y = 4\ \texttt{in}\ (\texttt{let}\ f = \lambda x.\ldots\ \texttt{in}\ L_1[6] + L_2[4 + \texttt{pick}\ 2\ 3])\\
\xrightarrow{n}{}^{*}\ &L_t[6+7]
\end{aligned}$$

In the previous example, one could easily keep track of how the normal-order reduction adheres to a choice, once it is made. Since this property could be lost, if non-normal-order reductions are performed, we will investigate these in more detail below.

Definition 3.10. *A reduction by a rule (a) within a context C is called* internal *and depicted by $\xrightarrow{i,\ C,\ a}$ (or $\xrightarrow{iC,\ a}$ for short) if it is a non-normal-order reduction.*

Because internal reductions which take place in reduction contexts are of special interest, we inspect the structure of reduction contexts making the following observations.

Lemma 3.11. *There are no* $\xrightarrow{i\mathcal{R},\ lapp}$-, $\xrightarrow{i\mathcal{R},\ lbeta}$- *or* $\xrightarrow{i\mathcal{R},\ nd}$*-reductions.*

Lemma 3.12. *A reduction inside a reduction context* $L^*_R[A^*_L]$ *is internal if (llet) or (cp) is applied to the subterm* $\texttt{let}\ x = r\ \texttt{in}\ s$ *of* $L^*_R[A^*_L[\texttt{let}\ x = r\ \texttt{in}\ s]]$ *and either there is no reduction context* R *such that* $s \equiv R[x]$ *holds or* A^*_L *is not the empty context.*

Lemma 3.13. *A reduction for a term* $L^*_R[\texttt{let}\ x = A^*_L[\texttt{let}\ y = r\ \texttt{in}\ s]\ \texttt{in}\ R[x]]$ *is internal within a reduction context if* R *is a reduction context and the rule (llet) or (cp) is applied to* $\texttt{let}\ y = r\ \texttt{in}\ s$*.*

As an analysis shows, a $(iR,\ llet)$-reduction may *only* be of a form as in the previous two lemmas whereas for $(iR,\ cp)$, because the target location of the copy operation does indeed matter, there is one additional possibility.

Corollary 3.14. *Let* $s, t \in \mathcal{T}$ *be terms. Then* $s \xrightarrow{i\mathcal{R}} t$ *if and only if one of the following holds and rule (llet) or (cp) is applied to the subterm denoted by* s'*:*

- $s \equiv L^*_R[\texttt{let}\ x = q\ \texttt{in}\ C[x]]$ *where* $s' \equiv \texttt{let}\ x = q\ \texttt{in}\ C[x]$ *and* C *is* not *a reduction context.*
- $s \equiv L^*_R[A^*_L[s']]$ *where* $s' \equiv \texttt{let}\ y = q\ \texttt{in}\ r$ *and* A^*_L *is not the empty context or there is no reduction context* R *such that* $s \equiv R[y]$*.*
- $s \equiv L^*_R[\texttt{let}\ x = A^*_L[\texttt{let}\ y = q\ \texttt{in}\ r]\ \texttt{in}\ R[x]]$ *where* R *is a reduction context and* $s' \equiv \texttt{let}\ y = q\ \texttt{in}\ r$*.*

Lemma 3.15. *Let* $s, t \in \mathcal{T}$ *be terms such that* $s \xrightarrow{i\mathcal{R}} t$ *holds. Then* t *is a WHNF only if* s *is a WHNF.*

Proof. By case analysis on the contraposition of the claim. □

The notion of *convergence* in the $\boldsymbol{\lambda}_{ND}$-calculus is defined by a normal-order reduction sequence to a term of the form $L^*_R[\lambda x.t]$, i.e. a WHNF. So we write $s \Downarrow t$ if and only if $s \xrightarrow{n}{}^{*} t$ and t is a WHNF, $s \Downarrow$ if there exists such a t and $s \not\Downarrow$ if not. Since neither a reduction inside an abstraction nor a (cp) whose target is not inside a reduction context constitutes a normal-order reduction, the conditions of definition 2.17 are satisfied and $\boldsymbol{\lambda}_{ND}$ forms a lazy computation system.

The following lemma considerably reduces the number of contexts which are necessary to establish the contextual preorder.

Lemma 3.16 (Context Lemma). *Let* $s \in \mathcal{T}$ *be a term and* $T \subseteq \mathcal{T}$ *a countable set of terms satisfying the following: For every reduction context* $R \in \mathcal{R}$ *with* $R[s] \Downarrow$ *there is a term* $t \in T$ *such that* $R[t] \Downarrow$ *holds.*

Then this property is also valid for general contexts, i.e. for every context $C \in \mathcal{C}$ *with* $C[s] \Downarrow$ *there exists* $t \in T$ *such that* $C[t] \Downarrow$ *is true.*

Proof. For $1 \leq i \leq k$ let $s_i \in \mathcal{T}$ be terms and $T_i \subseteq \mathcal{T}$ countable sets of terms. We will then show the following claim:

> If for every reduction context $R \in \mathcal{R}$ the following property holds: $R[s_i] \Downarrow$ implies that there is an $t_i \in T_i$ such that $R[t_i] \Downarrow$ too. Then for every multi-context $C \in \mathcal{C}$ with k holes, $C[s_1, \ldots, s_n] \Downarrow$ implies $\exists (t_1, \ldots, t_k) \in T_1 \times \cdots \times T_k : \ C[t_1, \ldots, t_k] \Downarrow$.

The proof is by induction on the lexicographical ordering consisting of the length of a normal order reduction $C[s_1, \ldots, s_k] \xrightarrow{n}{}^{*} p$ to some weak head normal form p and the number k of holes. For the induction base consider the case for a context C with only one hole, where $C[s_i]$ already is a weak head normal form. Then either $C[t_i]$ for some $t_i \in T_i$ is a weak head normal form too, or the hole is in a reduction context and the precondition proves the claim.

Hence for the induction step we assume the proposition to hold for all contexts smaller than C w.r.t. the given ordering. Then there are the following possibilities:

- In the case the first reduction of the sequence $C[s_1, \ldots, s_k] \xrightarrow{n}{}^{*} p$ is of the form $C[s_1, \ldots, s_k] \xrightarrow{n} C'[s_1, \ldots, s_k]$ the induction hypothesis applied to the context C' proves the claim. Note that this may always be the case for the rules (llet), (lapp) and (lbeta) but for the other reduction rules only if none of the subterms s_j is affected.
- If the reduction is performed inside one of the s_i, the ith hole is found at a necessary position, i.e. $C[s_1, \ldots, s_{i-1}, [\ ], s_{i+1}, \ldots, s_k]$ forms a reduction context. Now let $C' \equiv C[[\ ]_1, \ldots, [\ ]_{i-1}, s_i, [\ ]_{i+1}, \ldots, [\ ]_k]$ then the term $C'[s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_k]$ has the same normal order reduction as $C[s_1, \ldots, s_k]$ because $C'[s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_k] \equiv C[s_1, \ldots, s_k]$. Since C' is smaller than C, i.e. has $k-1$ holes, the induction hypothesis applies, so there are $(t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_k) \in T_1 \times \cdots \times T_{i-1} \times T_{i+1} \times \cdots \times T_k$ such that $C'[t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_k]$ may converge, too. From the above, we have that $R \equiv C[t_1, \ldots, t_{i-1}, [\ ], t_{i+1}, \ldots, t_k]$ is also a reduction context and therefore $R[s_i] \implies \exists t_i \in T_i : \ R[t_i] \Downarrow$ by the precondition. Thus by construction of R there exist $(t_1, \ldots, t_k) \in T_1 \times \cdots \times T_k$ such that $C[t_1, \ldots, t_k]$ converges.
- Now reductions of the form $C[s_1, \ldots, s_k] \xrightarrow{n} C'[s'_1, \ldots, s'_m]$ remain, where only the following rules come into question:
  - Using (nd), a subset of the s_i has been selected, i.e. $m \leq k$ is valid and for every $1 \leq j \leq m$ there is a $1 \leq i \leq k$ such that $s'_j \equiv s_i$ holds. This selection emerges for the reduction $C[t_1, \ldots, t_k] \xrightarrow{n} C'[t'_1, \ldots, t'_m]$ as well and therefore the induction hypothesis may be applied since the length of the normal order reduction sequence to a weak head normal form has been decreased.
  - For (cp), when a hole is within the copied expression, the corresponding s_i has been duplicated, i.e. there is a $1 \leq j \leq m$ and a variable renaming σ such that $s_j \equiv \sigma(s_i)$ holds. Note that, by the variable convention, σ only renames free variables of s_i which become bound by C. Therefore

the precondition $R[s_j] \Downarrow \Longrightarrow \exists t_j \in T_j : R[t_j] \Downarrow$ remains valid for s_j, too. Thus the proposition is shown by the induction hypothesis, since the length of the normal order reduction sequence to a weak head normal form has been decreased. □

The lemma is especially useful in the case where T is a singleton, i.e. $T = \{t\}$.

In preparation for a closer examination of possible deviations from normal-order reduction, we introduce the so-called surface contexts, which obey the significant property that a hole does not occur under an abstraction. This closely relates to the fact that by the rule (cp), only λ-terms may be copied.

Definition 3.17 (Surface Contexts). *The class $\mathcal{S}$ of* surface contexts *is given by the following syntactic rules for the symbol S where e means an expression:*

$$\begin{aligned} S ::= [\,] \;\mid\; S\,e \;\mid\; e\,S \;\mid\; & \\ & \texttt{let}\ x = e\ \texttt{in}\ S \;\mid\; \texttt{let}\ x = S\ \texttt{in}\ e \;\mid \\ & \texttt{pick}\ S\ e \;\mid\; \texttt{pick}\ e\ S \end{aligned}$$

Surface contexts are closed under composition, thus the results of lemma 2.27 and corollary 2.29 apply to them.

4 The Approximation Calculus $\boldsymbol{\lambda_{\approx}}$

The calculus presented in this section represents the fundamental bridge to the use of Howe's technique for proving bisimulation a congruence. Such a bridge is necessary, since the notion of bisimulation in [How89,How96] could not easily be adopted, for several reasons which we will expose subsequently.

First, in the $\boldsymbol{\lambda}_{ND}$-calculus there a distinction between the notion *canonical* and *non-canonical* for the `let`-operator is not possible: Evaluation could end up with a `let`-term as well as reduce it further. Also, `let` appears in the reduction rules as both an outermost operator and a qualifying subterm.

This could be resolved by regarding *answers* like in e.g. [AFM$^+$95a], i.e. values which carry an environment holding shared terms. But then the primary method of Howe's bisimulation, i.e. decomposition of canonical terms, looses sharing of subterms. Therefore we will rather use an experiment which applies those terms to fresh arguments as e.g. in [Abr90]. Furthermore, in the non-deterministic setting even this may be insufficient since it is problematical to fix particular answers during the application to arguments, as example 3.9 has shown.

These issues should not be considered just minor, technical obstacles. They rather impose severe restrictions on the design of the definition for a bisimulation which should be sound w.r.t. contextual equivalence. Another challenge concerns the proof in section 4.11.1, that the precongruence candidate $\widehat{\eta}$ is stable w.r.t. the reduction rules. Proving this property for the rule (llet) has turned out to be intractable for all the other variations of bisimulation-definitions we have tried.

Hence our approach eliminates the necessity for (llet)-reductions in that it may limit the reduction to an arbitrary but fixed depth and instead of answers,

produces pure values in form of abstractions without surrounding environments. Therefore, a further operator $\circledcirc$ is added to the language as a kind of *stop marker*, i.e. though $\circledcirc$ could be copied it does not have any reduction. Since this approximates reduction in some way, we use the term "approximation calculus" for this reason.

Furthermore copying is extended to simultaneously substituting all occurrences of a variable and executing garbage collection afterwards. At first sight, this may look similar to the mapping of λ_{NEED} to λ_{NAME} in [MOW98, sec. 4], but it is different, since only values and $\circledcirc$ may be copied instead of any term bound in a `let`-environment.

4.1 Terms, Contexts and Evaluation

The operators of the language are now $O = \{@, \lambda, \texttt{let}, \texttt{pick}, \circledcirc\}$, and the arity of $\circledcirc$ is $\alpha(\circledcirc) = \langle\rangle$, i.e. $\circledcirc$ is a new constant operator without any arguments. The arities of the other operators are the same as before.

The reduction rules evolve from the ones in $\boldsymbol{\lambda}_{ND}$ as follows. First, by the rule (stop) which may reduce every non-$\circledcirc$ term to $\circledcirc$, a further level of nondeterminism is introduced. As there is no rule for $\circledcirc$, this delimits the reduction, i.e. evaluation is pruned underneath.

Since it is our goal to eliminate top-level environments, it is natural to copy terms that could not be reduced further, namely $\circledcirc$ and abstractions, and garbage-collect their binding with the rule (cpa), so the original (cp)-rule becomes obsolete. Furthermore, we will permit all these reductions inside arbitrary surface contexts, which are denoted by $\mathcal{S}$ as before. Hence there is no need for the rule (llet) either, since we could first reduce inside the binding of a `let`-environment and then collapse it using (cpa).

Definition 4.1 (Reduction). *The calculus $\boldsymbol{\lambda}_{\approx}$ consists of the following rules:*

$$(\texttt{let } x = t_x \texttt{ in } s)\, t \xrightarrow{lapp}_{\boldsymbol{\lambda}_{\approx}} \texttt{let } x = t_x \texttt{ in } (s\, t) \qquad \text{(lapp)}$$

$$(\lambda x.s)\, t \xrightarrow{lbeta}_{\boldsymbol{\lambda}_{\approx}} \texttt{let } x = t \texttt{ in } s \qquad \text{(lbeta)}$$

$$\texttt{pick } s\ t \xrightarrow{nd,left}_{\boldsymbol{\lambda}_{\approx}} s \qquad \text{(nd, left)}$$

$$\texttt{pick } s\ t \xrightarrow{nd,right}_{\boldsymbol{\lambda}_{\approx}} t \qquad \text{(nd, right)}$$

$$\texttt{let } x = s \texttt{ in } t \xrightarrow{cpa}_{\boldsymbol{\lambda}_{\approx}} t[s/x] \quad \textit{where } s \equiv \lambda z.q \textit{ or } s \equiv \circledcirc \qquad \text{(cpa)}$$

$$s \xrightarrow{stop}_{\boldsymbol{\lambda}_{\approx}} \circledcirc \quad \textit{if } s \not\equiv \circledcirc \qquad \text{(stop)}$$

In the remainder, we will omit the subscript $\boldsymbol{\lambda}_{\approx}$ if it is clear from context, the reductions of which calculus are meant, in particular if (cpa) and (stop) are used.

Note that in contrast to $\boldsymbol{\lambda}_{ND}$ the notions of answer and value are the same in the approximation calculus. Again, we will use the notation $s \xrightarrow{C,\ a}_{\boldsymbol{\lambda}_\approx} t$ to indicate that for the reduction from s to t the reduction rule (a) is used in the context C, i.e. $s \equiv C[s']$ and $t \equiv C[t']$. As before we will speak of a *top-level* reduction if $s \xrightarrow{[\,],\ a}_{\boldsymbol{\lambda}_\approx} t$ and will use the respective symbols for the transitive and reflexive-transitive closure of $\rightarrow_{\boldsymbol{\lambda}_\approx}$ freely. We are now able to declare the kind of reduction which should be carried out in the $\boldsymbol{\lambda}_\approx$-calculus.

Definition 4.2. *We define the following unions of reductions:*

$$\xrightarrow{nd} = \xrightarrow{nd,left} \cup \xrightarrow{nd,right} \tag{nd}$$

$$\xrightarrow{p} = \xrightarrow{\mathcal{S},\ lapp} \cup \xrightarrow{\mathcal{S},\ lbeta} \cup \xrightarrow{\mathcal{S},\ nd} \cup \xrightarrow{\mathcal{S},\ cpa} \cup \xrightarrow{\mathcal{S},\ stop} \tag{p}$$

where a further specification of a context class for a $\xrightarrow{p}$-reduction is only permitted if it represents a subset of surface contexts, i.e. $s \xrightarrow{p,\ \mathcal{D},\ a} t$ means that the rule $a \in \{lapp, lbeta, nd, cpa, stop\}$ is applied in a context $D \in \mathcal{D} \subseteq \mathcal{S}$. A reduction of type $\xrightarrow{p}$ is called and approximation *reduction.*

In order to prove the precongruence candidate a simulation, a notion which we will define in section 4.8, we will pursue an approach similar to the *operator extensionality* of [How89, sec. 3.1], since it is more suitable for the small-step reduction of our $\boldsymbol{\lambda}_\approx$-calculus than to provide evidence that the precongruence candidate is *respected by evaluation* as in [How96]. Therefore, we will need a set of evaluation relations $\Downarrow_k$ which fulfils the conditions that

- $a \Downarrow_0 b$ if and only if $a \equiv b$ is an answer.
- If $a \Downarrow_k b$ for some $k > 0$ then a is not an answer but b is.

and write $a \Downarrow b$ if there is some $k \geq 0$ such that $a \Downarrow_k b$. It is then to show that $s' \mathrel{\widehat{\eta}} t$ holds whenever $s \mathrel{\widehat{\eta}} t$ and $s \Downarrow s'$ do. Defining $\Downarrow_k$ in terms of stepwise reduction, this can be accomplished for every reduction rule separately. Hence the following definition.

Definition 4.3 (Evaluation). *For $k \geq 0$ we define the relation $\Downarrow_{\boldsymbol{\lambda}_\approx,k} \subseteq \mathcal{T}^2$ by*

$$s \Downarrow_{\boldsymbol{\lambda}_\approx,k} t \iff \exists \lambda x.t' :\ s \xrightarrow{p}{}^{k}\ t \ \wedge\ t \equiv \lambda x.t' \tag{4.1}$$

and say that s evaluates *or* converges *to t in k steps. We use the symbol $\Downarrow_{\boldsymbol{\lambda}_\approx}$ if there exists a k such that $\Downarrow_{\boldsymbol{\lambda}_\approx,k}$ holds and will simply write $s \Downarrow_{\boldsymbol{\lambda}_\approx}$ if we are not interested in the term s converges to.*

Again, the criteria of definition 2.17 are obviously fulfilled. As with reduction, we may omit the subscript $\boldsymbol{\lambda}_\approx$ if it is clear from context. Since our evaluation relation is highly non-deterministic, it is convenient to gather all possible answers a term converges to. So the following notion makes sense.

Definition 4.4. *We define the* answer set $\mathtt{ans}(s) \subseteq \mathcal{T}$ *of a term* s *as follows.*

$$\mathtt{ans}(s) = \{t \mid s \Downarrow t\}$$

and augment $\mathtt{ans}(\cdot)$ *to sets of terms by* $\mathtt{ans}(S) = \{t \mid s \in S \ \wedge \ t \in \mathtt{ans}(s)\}$.

Closed terms possess only closed answers, since reduction does not introduce free variables, which is the statement of the following lemma.

Lemma 4.5. *Let* $s \in \mathcal{T}_0$ *be a closed term. Then* $\mathtt{ans}(s) \subseteq \mathcal{T}_0$.

It is interesting to note that only two of the above rules may yield an abstraction directly in one step.

Remark 4.6. If $s \xrightarrow{p} \lambda x.t$ for two terms $s, \lambda x.t \in \mathcal{T}$ such that s is not an abstraction, then s must be of one of the following forms where the respective reduction rule is used:

- $s \equiv \mathtt{pick}\ r\ (\lambda x.t) \xrightarrow{[\,],\ nd,right} \lambda x.t$
- $s \equiv \mathtt{pick}\ (\lambda x.t)\ r \xrightarrow{[\,],\ nd,left} \lambda x.t$
- $s \equiv \mathtt{let}\ y = \lambda x.t\ \mathtt{in}\ y \xrightarrow{[\,],\ cpa} \lambda x.t$ as a special case of
- $s \equiv \mathtt{let}\ y = q\ \mathtt{in}\ r \xrightarrow{[\,],\ cpa} r[q/y] \equiv \lambda x.t$

Before we may proceed further, we first need some elementary properties about the reduction rule (cpa) which are the subject of the following section.

4.2 The (cpa)-reduction

Reduction by the rule (cpa) is the key to eliminate environments, and often we will use the fact, that it does not change convergent behaviour. In order to prove this, we need to establish a complete set of forking diagrams for $\xrightarrow{cpa}$ w.r.t. all the remaining $\xrightarrow{p}$-reductions. In the following we will only consider the cases where, according to corollary 2.29, a surrounding surface context already is abandoned, since surface contexts are closed under composition.

If (cpa) is applied in the empty context, the whole expression must be of the form $\mathtt{let}\ x = s\ \mathtt{in}\ t$ with $s \equiv \circledcirc$ or $s \equiv \lambda z.q$. Hence a (lapp)-reduction is possible only in the surface context $\mathtt{let}\ x = e\ \mathtt{in}\ S$, which results in the first diagram. On the other hand, if (lapp) is top-level, a reduction by (cpa) may take place in every non-empty surface context, which could be commuted easily in the following three diagrams. The last represents the case where (cpa) makes the top-level (lapp)-reduction superfluous.

$$\xleftarrow{\mathtt{let}\ x=e\ \mathtt{in}\ S,\ lapp} \cdot \xrightarrow{[\,],\ cpa} \ \rightsquigarrow \ \xrightarrow{[\,],\ cpa} \cdot \xleftarrow{S,\ lapp}$$

$$\xleftarrow{[\,],\ lapp} \cdot \xrightarrow{(\mathtt{let}\ x=t_x\ \mathtt{in}\ s)\,S,\ cpa} \ \rightsquigarrow \ \xrightarrow{\mathtt{let}\ x=t_x\ \mathtt{in}\ (s\,S),\ cpa} \cdot \xleftarrow{[\,],\ lapp}$$

$$\xleftarrow{[\,],\ lapp} \cdot \xrightarrow{(\mathtt{let}\ x=S\ \mathtt{in}\ s)\,t,\ cpa} \ \rightsquigarrow \ \xrightarrow{\mathtt{let}\ x=S\ \mathtt{in}\ (s\,t),\ cpa} \cdot \xleftarrow{[\,],\ lapp}$$

$$\xleftarrow{[\,],\ lapp} \cdot \xrightarrow{(\mathtt{let}\ x=t_x\ \mathtt{in}\ S)\,t,\ cpa} \ \rightsquigarrow \ \xrightarrow{\mathtt{let}\ x=t_x\ \mathtt{in}\ (S\,t),\ cpa} \cdot \xleftarrow{[\,],\ lapp}$$

$$\xleftarrow{[\,],\ lapp} \cdot \xrightarrow{[\,]\,t,\ cpa} \ \rightsquigarrow \ \xrightarrow{[\,],\ cpa}$$

The first case for (lbeta) is the same as for (lapp), whereas for a top-level application of (lbeta), the situation is slightly different since (cpa) is only possible in surface contexts of the form $(\lambda x.e)\,S$ now.

$$\xleftarrow{\texttt{let}\ x{=}e\ \texttt{in}\ S,\ lbeta} \cdot \xrightarrow{[\,],\ cpa} \rightsquigarrow \xrightarrow{[\,],\ cpa} \cdot \xleftarrow{S,\ lbeta}$$

$$\xleftarrow{[\,],\ lbeta} \cdot \xrightarrow{(\lambda x.e)\,S,\ cpa} \rightsquigarrow \xrightarrow{\texttt{let}\ x{=}S\ \texttt{in}\ e,\ cpa} \cdot \xleftarrow{[\,],\ lbeta}$$

The rules (nd, left) and (nd, right) do not pose any problem even if the target of the copy operation lies inside the `pick` in question, consequently the following two diagrams ensue.

$$\xleftarrow{\texttt{let}\ x{=}e\ \texttt{in}\ S,\ nd{,}left} \cdot \xrightarrow{[\,],\ cpa} \rightsquigarrow \xrightarrow{[\,],\ cpa} \cdot \xleftarrow{S,\ nd{,}left}$$

$$\xleftarrow{\texttt{let}\ x{=}e\ \texttt{in}\ S,\ nd{,}right} \cdot \xrightarrow{[\,],\ cpa} \rightsquigarrow \xrightarrow{[\,],\ cpa} \cdot \xleftarrow{S,\ nd{,}right}$$

In the converse case, i.e. if (cpa) is applied inside the `pick`, it may be eliminated.

$$\xleftarrow{[\,],\ nd{,}left} \cdot \xrightarrow{\texttt{pick}\ S\ e,\ cpa} \rightsquigarrow \xrightarrow{S,\ cpa} \cdot \xleftarrow{[\,],\ nd{,}left}$$

$$\xleftarrow{[\,],\ nd{,}right} \cdot \xrightarrow{\texttt{pick}\ S\ e,\ cpa} \rightsquigarrow \xleftarrow{[\,],\ nd{,}right}$$

$$\xleftarrow{[\,],\ nd{,}right} \cdot \xrightarrow{\texttt{pick}\ e\ S,\ cpa} \rightsquigarrow \xrightarrow{S,\ cpa} \cdot \xleftarrow{[\,],\ nd{,}right}$$

$$\xleftarrow{[\,],\ nd{,}left} \cdot \xrightarrow{\texttt{pick}\ e\ S,\ cpa} \rightsquigarrow \xleftarrow{[\,],\ nd{,}left}$$

The cases for the rule (cpa) overlapping itself conceal that a "top-down" strategy seems capable to minimise the number of target locations for (cpa)-reductions.

$$\xleftarrow{\texttt{let}\ x{=}e\ \texttt{in}\ S,\ cpa} \cdot \xrightarrow{[\,],\ cpa} \rightsquigarrow \xrightarrow{[\,],\ cpa} \cdot \xleftarrow{S,\ cpa}$$

$$\xleftarrow{[\,],\ cpa} \cdot \xrightarrow{\texttt{let}\ x{=}e\ \texttt{in}\ S,\ cpa} \rightsquigarrow \xrightarrow{S,\ cpa} \cdot \xleftarrow{[\,],\ cpa}$$

If rule (stop) is applied to the term e to be copied by (cpa), then this has to be compensated by $\#_x(e)$ subsequent (stop)-reductions, where $\#_x(e)$ is the number of occurrences of the variable x in the term e. Note that these (stop)-reduction have to be admitted inside arbitrary contexts. Otherwise, (stop) may be applied to a target of (cpa) or a superterm of this target, or does not interfere with the copy procedure, hence the second diagram. The last diagram covers the case where (stop) is applied in the empty context, while substituting the whole term with $\circledcirc$ and thus deleting every surface (cpa)-redex.

$$\xleftarrow{\texttt{let}\ x{=}[\,]\ \texttt{in}\ e,\ stop} \cdot \xrightarrow{[\,],\ cpa} \rightsquigarrow \xrightarrow{[\,],\ cpa} \cdot \xleftarrow{\mathcal{C},\ stop}{}^{\#_x(e)}$$

$$\xleftarrow{\texttt{let}\ x{=}e\ \texttt{in}\ S,\ stop} \cdot \xrightarrow{[\,],\ cpa} \rightsquigarrow \xrightarrow{[\,],\ cpa} \cdot \xleftarrow{S,\ stop}$$

$$\xleftarrow{[\,],\ stop} \cdot \xrightarrow{\mathcal{S},\ cpa} \rightsquigarrow \xleftarrow{[\,],\ stop}$$

The above observations end up in the following lemma.

Lemma 4.7. *A complete set of forking diagrams for* $\xrightarrow{\mathcal{S},\ cpa}$ *w.r.t. the two sets* $\xrightarrow{p}$ *and* $\xrightarrow{\mathcal{C},\ stop}$ *of reductions is:*

$$\xleftarrow{\mathcal{S},\ a} \cdot \xrightarrow{\mathcal{S},\ cpa} \rightsquigarrow \xrightarrow{\mathcal{S},\ cpa} \cdot \xleftarrow{\mathcal{S},\ a} \tag{4.2}$$

$$\xleftarrow{[\,],\ lapp} \cdot \xrightarrow{[\,]\,t,\ cpa} \rightsquigarrow \xrightarrow{[\,],\ cpa} \tag{4.3}$$

$$\xleftarrow{\mathcal{S},\ nd} \cdot \xrightarrow{\mathcal{S},\ cpa} \rightsquigarrow \xleftarrow{\mathcal{S},\ nd} \tag{4.4}$$

$$\xleftarrow{\mathcal{S},\ stop} \cdot \xrightarrow{[\,],\ cpa} \rightsquigarrow \xrightarrow{[\,],\ cpa} \cdot \xleftarrow{\mathcal{C},\ stop}{}^{*} \tag{4.5}$$

$$\xleftarrow{[\,],\ stop} \cdot \xrightarrow{\mathcal{S},\ cpa} \rightsquigarrow \xleftarrow{[\,],\ stop} \tag{4.6}$$

Proof. According to corollary 2.29, the cases discussed above are exhausting. □

In order to conclude from this complete set of forking diagrams that convergent behaviour is preserved after an application of (cpa), we first have to look into the $\xrightarrow{\mathcal{C},\ stop}$-reductions which are introduced in the diagram (4.5).

4.3 Internal (stop)-reductions

Since in the diagrams for the reduction rule (cpa) there occur (stop)-reductions which are to be carried out in arbitrary rather than surface contexts, we have to show that these are safe for an approximation reduction to reach an abstraction. More precisely, this concerns the reductions by rule (stop) which are not within a surface context, hence the following definition.

Definition 4.8. *A* $\xrightarrow{C,\ stop}$*-reduction is called* internal*, depicted by* $\xrightarrow{i,\ C,\ stop}$*, if* $C \in \mathcal{C}$ *is a context which is not a surface context, i.e.* $C \notin \mathcal{S}$*.*

Therefore, in this section we will establish complete sets of commuting diagrams for $\xrightarrow{i,\ \mathcal{C},\ stop}$- w.r.t. $\xrightarrow{p}$-reductions.

Lemma 4.9. *Following is a complete set of commuting diagrams for* $\xrightarrow{i,\ \mathcal{C},\ stop}$ *w.r.t. approximation reductions:*

$$\xrightarrow{i,\ \mathcal{C},\ stop} \cdot \xrightarrow{\mathcal{S},\ a} \rightsquigarrow \xrightarrow{\mathcal{S},\ a} \cdot \xrightarrow{i,\ \mathcal{C},\ stop}$$

$$\xrightarrow{i,\ \texttt{let}\ x{=}\lambda y.C\ \texttt{in}\ t,\ stop} \cdot \xrightarrow{[\,],\ cpa} \rightsquigarrow \xrightarrow{[\,],\ cpa} \cdot \xrightarrow{i,\ \mathcal{C},\ stop}{}^{\#_x(t)}$$

$$\xrightarrow{i,\ S_1[(\lambda x.S_2)\,t],\ stop} \cdot \xrightarrow{S_1,\ lbeta} \rightsquigarrow \xrightarrow{S_1,\ lbeta} \cdot \xrightarrow{S_1[\texttt{let}\ x{=}t\ \texttt{in}\ S_2],\ stop}$$

Proof. Since — apart from the empty one — the contexts, in which approximation and internal (stop)-reductions respectively may take place, are disjoint by definition, the first diagram is an easy application of lemma 2.27. It also covers approximation reductions other than by rule (cpa) performed in the empty context.

The second diagram then handles just this case, where the internal (stop)-reduction takes place inside the term to be copied by (cpa) which therefore

has to be made up by as many internal (stop)-reductions as the variable x occurs in the target t of the copy operation. The internal (stop)-reduction may also be turned into an ordinary one, since a (lbeta)-reduction can bring the corresponding context to the surface, as the last diagram shows. □

It is easily seen that the application of the above commuting diagrams for internal (stop)-reductions terminates, since they strictly decrease the weight of a reduction sequence under the multi-set ordering on the multi-set containing the number of non-(stop) approximation reductions following each internal (stop)-reduction. Hence by induction on this measure of such a sequence, we may draw the conclusion that for every reduction sequence consisting of $\xrightarrow{p}$- and $\xrightarrow{i,\ stop}$-reductions to an abstraction, there is also an approximation reduction sequence to an abstraction that only differs by internal (stop)-reductions.

Lemma 4.10. *Let $s, \lambda x.t \in \mathcal{T}$ be terms with $s\ (\xrightarrow{p} \cup \xrightarrow{i,\ stop})^*\ \lambda x.t$. Then there is also a reduction $s \xrightarrow{p}{}^*\ \lambda x.t'$ such that $\lambda x.t' \xrightarrow{i,\ stop}{}^*\ \lambda x.t$ holds.*

Now we are in the position to connect this with the forking diagrams for (cpa).

Lemma 4.11. *Let s, t be terms such that $s \xrightarrow{\mathcal{S},\ cpa} t$. Then whenever s has an approximation reduction to an abstraction $\lambda x.s'$, there is also an approximation reduction starting from t leading to an abstraction $\lambda x.t'$ such that $\lambda x.s'$ differs from $\lambda x.t'$ only by internal (stop)-reductions, i.e. $\lambda x.t' \xrightarrow{i,\ stop}{}^*\ \lambda x.s'$ holds.*

Proof. Assume $s \xrightarrow{p}{}^k\ \lambda x.s'$, then we show

$$\exists \lambda x.t' :\ t \xrightarrow{p}{}^*\ \lambda x.t'\ \wedge\ \lambda x.t' \xrightarrow{i,\ stop}{}^*\ \lambda x.s'$$

by an induction on the length k of the reduction sequence.

- If $k = 1$ then according to remark 4.6, only $s \xrightarrow{[\,],\ cpa} \lambda x.s'$ and $s \xrightarrow{[\,],\ nd} \lambda x.s'$ are possible. So in both cases we obviously have $t \xrightarrow{p}{}^{0\vee 1}\ \lambda x.s'$ by the two diagrams (4.2) and (4.4) of lemma 4.7 that come into question.
- For the induction step assume the claim to be valid for sequences of length smaller than k. Then the sequence $s \xrightarrow{p}{}^k\ \lambda x.s'$ may be divided as follows:

  $$s \xrightarrow{p} s_1 \xrightarrow{p}{}^{k-1}\ \lambda x.s'$$

  We may apply the induction hypothesis to $s_1 \xrightarrow{p}{}^{k-1}\ \lambda x.s'$, i.e. if $s_1 \xrightarrow{\mathcal{S},\ cpa} t_1$ then there is a reduction $t_1 \xrightarrow{p}{}^*\ \lambda x.t'$ with $\lambda x.t' \xrightarrow{i,\ stop}{}^*\ \lambda x.s'$. This is applicable to the forking diagrams (4.2), (4.3) and (4.5) of lemma 4.7, so there is also a reduction $t \xrightarrow{p}{}^*\ t_1$, thus $t \xrightarrow{p}{}^*\ \lambda x.t'$.
  In the case of diagram (4.4), using the induction hypothesis is not necessary, since $t \xrightarrow{\mathcal{S},\ nd} t_1 \equiv s_1 \xrightarrow{p}{}^{k-1}\ \lambda x.s'$ directly.
  The last diagram (4.6) is not applicable here, since then s_1 would reduce to the term $\circledcirc$ instead of an abstraction. □

Note that the induction argument in the above lemma is valid because of the special structure of the forking diagrams for (cpa), i.e. there is at most one (cpa)-reduction necessary to clean up the forking situation.

4.4 The (lbeta)-reduction

Also for reduction by rule (lbeta) we may develop a complete set of forking diagrams w.r.t. $\xrightarrow{p}$-reductions.

If (lbeta) is applied in the empty context, the whole expression must be of the form $(\lambda x.t)\, s$. Hence a surface reduction may take place inside s or in the context $[\,]\, s$ only, the former of which obviously may be commuted with a (lbeta)-reduction, the first diagram.

$$\xleftarrow{(\lambda x.t)\, S,\ a} \cdot \xrightarrow{[\,],\ lbeta} \rightsquigarrow \xrightarrow{[\,],\ lbeta} \cdot \xleftarrow{\texttt{let}\ x{=}S\ \texttt{in}\ t,\ a}$$

If rule (lbeta) is applied inside a non-empty surface context $S \in \mathcal{S}$ and (lapp) is top-level, the following three diagrams arise. The fourth covers the situation of (stop) being applied in the empty context eliminating the (lbeta)-reduction while the last handles the case that the function argument of an (lbeta)-redex is canceled by (stop).

$$\xleftarrow{[\,],\ lapp} \cdot \xrightarrow{(\texttt{let}\ x{=}s\ \texttt{in}\ t)\, S,\ lbeta} \rightsquigarrow \xrightarrow{\texttt{let}\ x{=}s\ \texttt{in}\ (t\, S),\ lbeta} \cdot \xleftarrow{[\,],\ lapp}$$

$$\xleftarrow{[\,],\ lapp} \cdot \xrightarrow{(\texttt{let}\ x{=}s\ \texttt{in}\ S)\, t,\ lbeta} \rightsquigarrow \xrightarrow{\texttt{let}\ x{=}s\ \texttt{in}\ (S\, t),\ lbeta} \cdot \xleftarrow{[\,],\ lapp}$$

$$\xleftarrow{[\,],\ lapp} \cdot \xrightarrow{(\texttt{let}\ x{=}S\ \texttt{in}\ s)\, t,\ lbeta} \rightsquigarrow \xrightarrow{\texttt{let}\ x{=}S\ \texttt{in}\ (s\, t),\ lbeta} \cdot \xleftarrow{[\,],\ lapp}$$

$$\xleftarrow{[\,],\ stop} \cdot \xrightarrow{\mathcal{S},\ lbeta} \rightsquigarrow \xleftarrow{[\,],\ stop}$$

$$\xleftarrow{S[[\,]\, s],\ stop} \cdot \xrightarrow{S,\ lbeta} \rightsquigarrow \xrightarrow{S,\ stop} \cdot \xleftarrow{S,\ stop}$$

If the non-deterministic rules (nd) are applied at top-level, the (lbeta)-reduction may be eliminated as the following diagrams illustrate.

$$\xleftarrow{[\,],\ nd{,}left} \cdot \xrightarrow{\texttt{pick}\ S\ e,\ lbeta} \rightsquigarrow \xrightarrow{S,\ lbeta} \cdot \xleftarrow{[\,],\ nd{,}left}$$

$$\xleftarrow{[\,],\ nd{,}right} \cdot \xrightarrow{\texttt{pick}\ S\ e,\ lbeta} \rightsquigarrow \xleftarrow{[\,],\ nd{,}right}$$

$$\xleftarrow{[\,],\ nd{,}right} \cdot \xrightarrow{\texttt{pick}\ e\ S,\ lbeta} \rightsquigarrow \xrightarrow{S,\ lbeta} \cdot \xleftarrow{[\,],\ nd{,}right}$$

$$\xleftarrow{[\,],\ nd{,}left} \cdot \xrightarrow{\texttt{pick}\ e\ S,\ lbeta} \rightsquigarrow \xleftarrow{[\,],\ nd{,}left}$$

Overlapping (lbeta) with itself only modifies the surface context where the reduction takes place.

$$\xleftarrow{(\lambda x.t)\, S,\ lbeta} \cdot \xrightarrow{[\,],\ lbeta} \rightsquigarrow \xrightarrow{[\,],\ lbeta} \cdot \xleftarrow{\texttt{let}\ x{=}S\ \texttt{in}\ t,\ lbeta}$$

$$\xleftarrow{[\,],\ lbeta} \cdot \xrightarrow{(\lambda x.t)\, S,\ lbeta} \rightsquigarrow \xrightarrow{\texttt{let}\ x{=}S\ \texttt{in}\ t,\ lbeta} \cdot \xleftarrow{[\,],\ lbeta}$$

So let us consider the remaining case where rule (cpa) is applied in the empty context, i.e. the term has to be of the form $\texttt{let}\ x = \lambda z.q\ \texttt{in}\ t$ or $\texttt{let}\ x = \circledcirc\ \texttt{in}\ t$ and hence a surface reduction is possible inside t only.

$$\xleftarrow{[\,],\ cpa} \cdot \xrightarrow{\texttt{let}\ x=\lambda z.q\ \texttt{in}\ S,\ lbeta} \rightsquigarrow \xrightarrow{S,\ lbeta} \cdot \xleftarrow{[\,],\ cpa}$$

$$\xleftarrow{[\,],\ cpa} \cdot \xrightarrow{\texttt{let}\ x=\circledcirc\ \texttt{in}\ S,\ lbeta} \rightsquigarrow \xrightarrow{S,\ lbeta} \cdot \xleftarrow{[\,],\ cpa}$$

Again we refer to corollary 2.29 and collect these diagrams in a separate lemma.

Lemma 4.12. *A complete set of forking diagrams for* $\xrightarrow{\mathcal{S},\ lbeta}$ *w.r.t.* $\xrightarrow{p}$ *is:*

$$\xleftarrow{\mathcal{S},\ a} \cdot \xrightarrow{\mathcal{S},\ lbeta} \rightsquigarrow \xrightarrow{\mathcal{S},\ lbeta} \cdot \xleftarrow{\mathcal{S},\ a}$$

$$\xleftarrow{\mathcal{S},\ nd} \cdot \xrightarrow{\mathcal{S},\ lbeta} \rightsquigarrow \xleftarrow{\mathcal{S},\ nd}$$

$$\xleftarrow{[\,],\ stop} \cdot \xrightarrow{\mathcal{S},\ lbeta} \rightsquigarrow \xleftarrow{[\,],\ stop}$$

$$\xleftarrow{S[[\,]\,s],\ stop} \cdot \xrightarrow{S,\ lbeta} \rightsquigarrow \xrightarrow{S,\ stop} \cdot \xleftarrow{S,\ stop}$$

Before we can show that also reduction by rule (lbeta) preserves the approximation reduction to an abstraction, we need the following result about the correspondence of $\circledcirc$ and its application in surface contexts.

Lemma 4.13. *Let* $t \in \mathcal{T}$ *be a term and* $S \in \mathcal{S}$ *a surface context such that* $S[\circledcirc\, t] \xrightarrow{p}{}^{*} \lambda z.q$*. Then also* $S[\circledcirc] \xrightarrow{p}{}^{*} \lambda z.q$ *holds.*

Proof. W.l.o.g. we may require S to be non-empty, i.e. $S \not\equiv [\,]$, because $\circledcirc\, t$ obviously has no approximation reduction to an abstraction, since the rules (lapp) and (lbeta) would be the only possibilities to get rid of the top-level @-operator. So assume $S[\circledcirc\, t] \xrightarrow{p}{}^{k} \lambda z.q$ for an induction on the length k of the reduction sequence.

- In the case $S[\circledcirc\, t] \xrightarrow{p} \lambda z.q$ for $k = 1$ it is easily checked that only the rules (cpa) or (nd) could have been used to produce an abstraction. Since the term $\circledcirc\, t$ in question is located in a surface context, i.e. not under an abstraction, it must be discarded by (nd), whereas (cpa) is not possible.
- If $S[\circledcirc\, t] \xrightarrow{p} s \xrightarrow{p}{}^{k} \lambda z.q$ then either $s \equiv S'[\circledcirc\, t]$, $s \equiv S[\circledcirc\, t']$ or $s \equiv S[\circledcirc]$ hold, for the latter of which the proposition becomes trivial. On the other hand, to both $S'[\circledcirc\, t]$ and $S[\circledcirc\, t']$ the induction hypothesis is applicable, thus the claim holds. □

Now we can use an argument similar to, but in this case simpler than the one in lemma 4.11 to prove the following lemma.

Lemma 4.14. *Let* s, t *be terms such that* $s \xrightarrow{\mathcal{S},\ lbeta} t$*. Then whenever* s *has an approximation reduction to an abstraction there is also an approximation reduction sequence starting from* t *leading to the same abstraction, i.e.*

$$s \xrightarrow{p}{}^{*} \lambda z.q \implies t \xrightarrow{p}{}^{*} \lambda z.q$$

Proof. Assume $s \xrightarrow{p}{}^{k} \lambda z.q$ for an induction on the length k of the reduction sequence.

- If $k = 1$ then according to remark 4.6, only $s \xrightarrow{[\,],\ cpa} \lambda z.q$ and $s \xrightarrow{[\,],\ nd} \lambda z.q$ are possible, for both of which we obviously have $t \xrightarrow{p}{}^{0\vee1} \lambda z.q$ by the first two diagrams of lemma 4.12.
- For the induction step assume the claim to be valid for sequences of length smaller than k. Then the sequence $s \xrightarrow{p}{}^{k} \lambda z.q$ may be divided as follows:

$$s \xrightarrow{p} s_1 \xrightarrow{p}{}^{k-1} \lambda z.q$$

We may apply the induction hypothesis to $s_1 \xrightarrow{p}{}^{k-1} \lambda z.q$, i.e. if $s_1 \xrightarrow{\mathcal{S},\ lbeta} t_1$ then there is a reduction $t_1 \xrightarrow{p}{}^{*} \lambda z.q$. This is the case only for the first forking diagram of lemma 4.12, so there is also a reduction $t \xrightarrow{p}{}^{*} t_1$, thus $t \xrightarrow{p}{}^{*} \lambda z.q$.
In the case of the second and third diagram, using the induction hypothesis is not necessary, since $t \xrightarrow{\mathcal{S},\ nd} t_1 \equiv s_1 \xrightarrow{p}{}^{k-1} \lambda z.q$ directly.
If the last diagram is applicable, the special treatment it needs is done in lemma 4.13 thus the claim holds. □

Corollary 4.15. *Let s, t be terms such that $s \xrightarrow{\mathcal{S},\ lbeta} t$. Then for all abstractions $\lambda z.q$ the following holds:*

$$s \xrightarrow{p}{}^{*} \lambda z.q \iff t \xrightarrow{p}{}^{*} \lambda z.q$$

Proof. The "only-if"-part is just the statement of the previous lemma where the "if"-part results from the fact that $s \xrightarrow{\mathcal{S},\ lbeta} t$.

4.5 The (lapp)-reduction

In order to establish a complete set of forking diagrams, we have only to examine (lapp) in combination with (nd), (stop) and itself now, since (lbeta) and (cpa) are covered by the previous sections.

As an analysis shows, overlapping (lapp) with itself is no problem while the cases for (stop) and (nd) look similar to the ones for (lbeta):

$$\xleftarrow{[\,],\ stop} \cdot \xrightarrow{\mathcal{S},\ lapp} \rightsquigarrow \xleftarrow{[\,],\ stop}$$

$$\xleftarrow{S[[\,]\,s],\ stop} \cdot \xrightarrow{S,\ lapp} \rightsquigarrow \xrightarrow{S,\ stop} \cdot \xleftarrow{S,\ stop}$$

$$\xleftarrow{[\,],\ nd,left} \cdot \xrightarrow{\texttt{pick}\ S\ e,\ lapp} \rightsquigarrow \xrightarrow{S,\ lapp} \cdot \xleftarrow{[\,],\ nd,left}$$

$$\xleftarrow{[\,],\ nd,right} \cdot \xrightarrow{\texttt{pick}\ S\ e,\ lapp} \rightsquigarrow \xleftarrow{[\,],\ nd,right}$$

$$\xleftarrow{[\,],\ nd,right} \cdot \xrightarrow{\texttt{pick}\ e\ S,\ lapp} \rightsquigarrow \xrightarrow{S,\ lapp} \cdot \xleftarrow{[\,],\ nd,right}$$

$$\xleftarrow{[\,],\ nd,left} \cdot \xrightarrow{\texttt{pick}\ e\ S,\ lapp} \rightsquigarrow \xleftarrow{[\,],\ nd,left}$$

Hence the following lemma corresponds closely to its counterpart.

Lemma 4.16. *A complete set of forking diagrams for* $\xrightarrow{\mathcal{S},\ lapp}$ *w.r.t.* $\xrightarrow{p}$ *is:*

$$\xleftarrow{\mathcal{S},\ a} \cdot \xrightarrow{\mathcal{S},\ lapp} \rightsquigarrow \xrightarrow{\mathcal{S},\ lapp} \cdot \xleftarrow{\mathcal{S},\ a}$$

$$\xleftarrow{\mathcal{S},\ nd} \cdot \xrightarrow{\mathcal{S},\ lapp} \rightsquigarrow \xleftarrow{\mathcal{S},\ nd}$$

$$\xleftarrow{[\ ],\ stop} \cdot \xrightarrow{\mathcal{S},\ lapp} \rightsquigarrow \xleftarrow{[\ ],\ stop}$$

$$\xleftarrow{\mathcal{S},\ stop} \cdot \xrightarrow{\mathcal{S},\ lapp} \rightsquigarrow \xrightarrow{\mathcal{S},\ stop} \cdot \xleftarrow{\mathcal{S},\ stop}$$

4.6 Some Specific Cases of Reduction

This section discusses some specific cases in which approximation reductions may be commuted.

Corollary 4.17. *Let* $S \in \mathcal{S}$ *be a surface context and* $r, s, t \in \mathcal{T}$ *be terms. Then for all types* a *of reductions the following diagrams commute:*

$$\begin{array}{ccc} S[(\lambda x.s)\,t] & \xrightarrow{S,\ (lbeta)} & S[\texttt{let}\ x = t\ \texttt{in}\ s] \\ \downarrow{\scriptstyle S[(\lambda x.s)\,[\ ]],\ a} & & \downarrow{\scriptstyle S[\texttt{let}\ x=[\ ]\ \texttt{in}\ s],\ a} \\ S[(\lambda x.s)\,t'] & \xrightarrow{S,\ (lbeta)} & S[\texttt{let}\ x = t'\ \texttt{in}\ s] \end{array}$$

$$\begin{array}{ccc} S[(\texttt{let}\ x = r\ \texttt{in}\ s)\,t] & \xrightarrow{S,\ (lapp)} & S[\texttt{let}\ x = r\ \texttt{in}\ s\,t] \\ \downarrow{\scriptstyle S[(\texttt{let}\ x=r\ \texttt{in}\ s)\,[\ ]],\ a} & & \downarrow{\scriptstyle S[\texttt{let}\ x=r\ \texttt{in}\ s\,[\ ]],\ a} \\ S[(\texttt{let}\ x = r\ \texttt{in}\ s)\,t'] & \xrightarrow{S,\ (lapp)} & S[\texttt{let}\ x = r\ \texttt{in}\ s\,t'] \end{array}$$

Proof. By lemma 2.27 since the respective surface contexts are disjoint. □

Lemma 4.18. *Let* $S \in \mathcal{S}$ *be a surface context,* $\lambda z.q \in \mathcal{T}$ *some abstraction and* $t \in \mathcal{T}_0$ *a closed term such that* $S[t] \Downarrow \lambda z.q$ *holds. Then either there is a closed abstraction* $\lambda x.s \in \mathcal{T}_0$ *satisfying* $t \Downarrow \lambda x.s$ *and* $S[\lambda x.s] \Downarrow \lambda z.q$*, or* $S[\circledcirc]$ *converges.*

Proof. Since for $S \equiv [\ ]$ or $t \equiv \lambda z.q$ there is nothing to show, we rule out these cases. The proof is then by induction on the length k of an approximation reduction sequence $S[t] \xrightarrow{\mathcal{S}}{}^{k}_{\boldsymbol{\lambda}_{\approx}} \lambda z.q$ to an abstraction. Since S cannot be the empty context $k = 0$ is not possible and $k = 1$ forms the induction base. So if $S[t] \xrightarrow{\mathcal{S}}_{\boldsymbol{\lambda}_{\approx}} \lambda z.q$ then also $S[\circledcirc] \Downarrow$ as t is not an abstraction.

Hence for the induction step we may assume that the claim holds for all sequences of length k. Then for the first reduction $S[t] \xrightarrow{\mathcal{S}}_{\boldsymbol{\lambda}_{\approx}} r$ of an approximation sequence $S[t] \xrightarrow{\mathcal{S}}_{\boldsymbol{\lambda}_{\approx}} r \xrightarrow{\mathcal{S}}{}^{k}_{\boldsymbol{\lambda}_{\approx}} \lambda z.q$ the following cases have to be distinguished:

– For $r \equiv S[t']$, i.e. the reduction took place within t, the proposition is shown by the induction hypothesis for the remaining sequence.

- If the reduction is performed on the context S so that the hole is deleted by rule (stop), obviously already $S[\circledcirc]$ will converge. For any other reduction rule, we will obtain $r \equiv S'[t]$ to which the induction hypothesis applies. Rule (cpa) is no exception here, since t is closed and therefore will not be altered.
- For a reduction of the form $S[t] \equiv S'[(\lambda y.p)\, t] \xrightarrow{S',\ lbeta} S'[\texttt{let}\ y = t\ \texttt{in}\ p]$ the term t remains in a surface context and we therefore may apply the induction hypothesis to $S''[\,] \equiv S'[\texttt{let}\ y = [\,]\ \texttt{in}\ p]$. Hence, there are the following possibilities:
  - In the case of $S''[\circledcirc] \Downarrow$ we obviously have $S[\circledcirc] \Downarrow$ too, since the reduction $S[\circledcirc] \equiv S'[(\lambda y.p) \circledcirc] \xrightarrow{S',\ lbeta} S'[\texttt{let}\ y = \circledcirc\ \texttt{in}\ p] \equiv S''[\circledcirc]$.
  - On the other hand, if there is a closed abstraction $\lambda x.s$ such that $t \Downarrow \lambda x.s$ and $S''[\lambda x.s] \Downarrow \lambda z.q$ both hold, we may construct an approximation sequence ending in $\lambda z.q$ for $S[t]$ as follows. In order to move the aforesaid reduction $S[t] \equiv S'[(\lambda y.p)\, t] \xrightarrow{S',\ lbeta} S'[\texttt{let}\ y = t\ \texttt{in}\ p] \equiv S''[t]$ from the beginning to the end of the sequence $S''[t] \xrightarrow{\mathcal{S}}{}^{*}_{\boldsymbol{\lambda}_{\approx}} S''[\lambda x.s]$ we deploy corollary 4.17 repeatedly. I.e., by induction on the length j of the approximation sequence $S''[t] \xrightarrow{\mathcal{S}}{}^{j}_{\boldsymbol{\lambda}_{\approx}} S''[\lambda x.s]$ we obtain $t \Downarrow \lambda x.s$ and $S[\lambda x.s] \Downarrow \lambda z.q$ as desired.
- An application of the rule (lapp) is treated analogously to the previous case, i.e. either $S[\circledcirc]$ converges anyway, or, by corollary 4.17, every reduction like $S[t] \equiv S'[(\texttt{let}\ y = r\ \texttt{in}\ p)\, t] \xrightarrow{S',\ lapp} S'[\texttt{let}\ y = r\ \texttt{in}\ (p\, t)] \equiv S''[t]$ may be moved after the end of a sequence $S''[t] \xrightarrow{\mathcal{S}}{}^{*}_{\boldsymbol{\lambda}_{\approx}} S''[\lambda x.s]$. □

4.7 Standardisation

We may, in some respect, standardise every reduction leading to an abstraction, which will be helpful later. The goal for a term of the form $\texttt{let}\ x = s\ \texttt{in}\ t$ is to first reduce inside s until $\circledcirc$ or an abstraction is reached and then to collapse the `let`-expression using (cpa) before proceeding with reductions inside t. This means, we have to bring in front all the reductions that take place inside s, i.e. reductions of type $\xrightarrow{\texttt{let}\ x=\mathcal{C}\ \texttt{in}\ t,\ a}_{\boldsymbol{\lambda}_{\approx}}$, a task for which we first establish the following corollary.

Corollary 4.19. *Let* $\texttt{let}\ x = s\ \texttt{in}\ t \in \mathcal{T}$ *be a term and* a, b *two arbitrary reduction types. Then for all contexts* $C_a, C_b \in \mathcal{C}$ *the following diagram commutes*

$$\begin{array}{ccc} \texttt{let}\ x = s\ \texttt{in}\ t & \xrightarrow{\texttt{let}\ x=C_b\ \texttt{in}\ t,\ b} & \texttt{let}\ x = s'\ \texttt{in}\ t \\ \Big\downarrow{\scriptstyle \texttt{let}\ x=s\ \texttt{in}\ C_a,\ a} & & \Big\downarrow{\scriptstyle \texttt{let}\ x=s'\ \texttt{in}\ C_a,\ a} \\ \texttt{let}\ x = s\ \texttt{in}\ t' & \xrightarrow{\texttt{let}\ x=C_b\ \texttt{in}\ t',\ b} & \texttt{let}\ x = s'\ \texttt{in}\ t' \end{array}$$

Proof. By a simple application of lemma 2.27. □

Note that an *approximation* reduction $\xrightarrow{\texttt{let } x=\mathcal{C} \texttt{ in } t,\ a}_{\boldsymbol{\lambda}_{\approx}}$ must take place in a surface context, i.e. $C \in \mathcal{S}$ must hold. The following lemma now uses the previous commutativity result to move such $\xrightarrow{\texttt{let } x=\mathcal{S} \texttt{ in } t,\ a}_{\boldsymbol{\lambda}_{\approx}}$-reductions successively forward.

Lemma 4.20. *For every reduction sequence* $\texttt{let } x = s \texttt{ in } t \xrightarrow{p}{}^{n} \lambda z.q$ *with* $n \geq 0$ *there is also an approximation reduction sequence*

$$\texttt{let } x = s \texttt{ in } t \xrightarrow{\texttt{let } x=\mathcal{S} \texttt{ in } t}{}^{k}_{\boldsymbol{\lambda}_{\approx}} \texttt{let } x = s' \texttt{ in } t \xrightarrow{[\,],\ cpa} t[s'/x] \xrightarrow{p}{}^{m} \lambda z.q$$

where s' *represents* $\circledcirc$ *or an abstraction and* $k + 1 + m = n$ *holds.*

Proof. Assume $\texttt{let } x = s \texttt{ in } t \xrightarrow{p}{}^{n} \lambda z.q$. Then the proof is by induction on the length n of the reduction sequence. According to remark 4.6, for a term of the form $\texttt{let } x = s \texttt{ in } t$ the only rule which may produce an abstraction in a single step is (cpa), hence the induction base is clear.

For the induction step, we may split up the above reduction sequence into $\texttt{let } x = s \texttt{ in } t \xrightarrow{p} r \xrightarrow{p}{}^{n} \lambda z.q$. Now either $\texttt{let } x = s \texttt{ in } t \xrightarrow{p} r$ already is of the desired form, thus we apply the induction hypothesis to $r \xrightarrow{p}{}^{n} \lambda z.q$ which proves the claim.

If $\texttt{let } x = s \texttt{ in } t \xrightarrow{\texttt{let } x=s \texttt{ in } \mathcal{S}}_{\boldsymbol{\lambda}_{\approx}} \texttt{let } x = s \texttt{ in } t' \equiv r$ is the reduction, we have to apply the induction hypothesis twice. First, from $\texttt{let } x = s \texttt{ in } t' \xrightarrow{p}{}^{n} \lambda z.q$ we yield an approximation reduction sequence

$$\texttt{let } x = s \texttt{ in } t' \xrightarrow{\texttt{let } x=\mathcal{S} \texttt{ in } t'}{}^{k}_{\boldsymbol{\lambda}_{\approx}} \texttt{let } x = s' \texttt{ in } t' \xrightarrow{[\,],\ cpa} t'[s'/x] \xrightarrow{p}{}^{m} \lambda z.q$$

with $k + 1 + m = n$ for which we distinguish the following alternatives:

- For $s' \equiv s$ if $k = 0$ is the case, s must be $\circledcirc$ or an abstraction and hence the rule (cpa) may already be applied to $\texttt{let } x = s \texttt{ in } t$ with which the reduction $\texttt{let } x = s \texttt{ in } t \xrightarrow{\texttt{let } x=s \texttt{ in } \mathcal{S}}_{\boldsymbol{\lambda}_{\approx}} \texttt{let } x = s \texttt{ in } t'$ obviously may be commuted.
- If $\texttt{let } x = s \texttt{ in } t' \xrightarrow{\texttt{let } x=\mathcal{S} \texttt{ in } t'}{}^{k\geq 1}_{\boldsymbol{\lambda}_{\approx}} \texttt{let } x = s' \texttt{ in } t'$, i.e.

  $$\texttt{let } x = s \texttt{ in } t' \xrightarrow{\texttt{let } x=\mathcal{S} \texttt{ in } t'}_{\boldsymbol{\lambda}_{\approx}} \texttt{let } x = s'' \texttt{ in } t' \xrightarrow{\texttt{let } x=\mathcal{S} \texttt{ in } t'}{}^{k-1}_{\boldsymbol{\lambda}_{\approx}} \texttt{let } x = s' \texttt{ in } t'$$

  then by corollary 4.19 we may commute the first of these reductions with the preceding $\texttt{let } x = s \texttt{ in } t \xrightarrow{\texttt{let } x=s \texttt{ in } \mathcal{S}}_{\boldsymbol{\lambda}_{\approx}} \texttt{let } x = s \texttt{ in } t'$ hence

  $$\texttt{let } x = s \texttt{ in } t \xrightarrow{\texttt{let } x=\mathcal{S} \texttt{ in } t}_{\boldsymbol{\lambda}_{\approx}} \texttt{let } x = s'' \texttt{ in } t \xrightarrow{\texttt{let } x=s'' \texttt{ in } \mathcal{S}}_{\boldsymbol{\lambda}_{\approx}}$$
  $$\texttt{let } x = s'' \texttt{ in } t' \xrightarrow{\texttt{let } x=\mathcal{S} \texttt{ in } t'}{}^{k-1}_{\boldsymbol{\lambda}_{\approx}} \texttt{let } x = s' \texttt{ in } t' \xrightarrow{[\,],\ cpa} t'[s'/x] \xrightarrow{p}{}^{m} \lambda z.q$$

  Now apply the induction hypothesis to the suffix of the above sequence, thus the claim holds. □

Remark 4.21. Note that the situation here is different from the one in lemma 4.11 since there is no need to shift (cpa)-reductions over (stop)-reductions.

Combining the approximation reduction sequence $\xrightarrow{\texttt{let } x=\mathcal{S} \texttt{ in } t}{}^{*}_{\boldsymbol{\lambda}_{\approx}} \cdot \xrightarrow{[\,],\ cpa}$ in one distinct reduction, the result of the previous lemma may then easily be generalised to arbitrarily deep environments by induction.

Definition 4.22. *Let r stand for $\circledcirc$ or an abstraction. Then the reduction rule (olf) is defined by*

$$\texttt{let } x = r \texttt{ in } s \xrightarrow{olf} t \qquad \text{(olf)}$$
$$\text{if } \texttt{let } x = r \texttt{ in } s \xrightarrow{p}{}^{*} \texttt{let } x = r' \texttt{ in } s \xrightarrow{[\,],\ cpa} s[r'/x] \equiv t$$

We then call an approximation reduction sequence of the form $L[t] \xrightarrow{olf}{}^{*} t'$ for some environment $L \in \mathcal{E}$ an outer let first *sequence.*

The following fruitful theorem shows that such an outer let first sequence exists for every converging reduction.

Theorem 4.23 (Standardisation). *Let $t \in \mathcal{T}$ be a term. Then for every approximation reduction $t \xrightarrow{p}{}^{*} \lambda z.q$ to an abstraction there exists also an outer let first sequence $t \xrightarrow{olf}{}^{*} t' \xrightarrow{p}{}^{*} \lambda z.q$ such that t' is not a* `let`*-term.*

Proof. Assuming $t \equiv L[s]$ and using lemma 4.20 for an induction on the size of the environment L. □

4.8 Similarity

In order to respect sharing, some effort in the definition of what a simulation should be, becomes necessary. Since we haven't stated much about the extension of a relation to open terms yet, our definition of an experiment rather matches the applicative bisimulation of [Abr90]. But afterwards, we will point out that in the approximation calculus both views are identical.

Definition 4.24 (Experiment). *The experiment $[\cdot]_{\boldsymbol{\lambda}_{\approx}} : {\mathcal{T}_0}^2 \to {\mathcal{T}_0}^2$ for the calculus $\boldsymbol{\lambda}_{\approx}$ is defined as follows.*

$$s' \ [\eta]_{\boldsymbol{\lambda}_{\approx}} \ t' \iff$$
$$\forall \lambda x.s \in \texttt{ans}(s') : \ \exists \lambda y.t \in \texttt{ans}(t') : \ \forall r : \ r \in \mathcal{T}_0 \implies (\lambda x.s)\, r \ \eta \ (\lambda x.t)\, r$$

Using non-determinism, the above definition exploits that with the rules (stop) and (cpa), it is now possible to equip abstractions with the information about their `let`-environments up to every arbitrary depth.

Remark 4.25. Note that, since we consider convergence to an abstraction, i.e. we do not have to deal with environments around the values, and under the extension η^o which we will give later, the above approach is identical to the Howe-like

$$s' \ [\eta]_{\boldsymbol{\lambda}\approx} \ t' \iff \forall \lambda x.s \in \mathtt{ans}(s') : \ \exists \lambda x.t \in \mathtt{ans}(t') : \ s \ \eta^o \ t$$

where the abstractions would simply be stripped off from the answers.

We proceed with some fundamental properties of $[\eta]_{\boldsymbol{\lambda}\approx}$ first. Since $s \Downarrow$ means $\mathtt{ans}(s) = \emptyset$ we have the following corollary.

Corollary 4.26. *Let $s \in \mathcal{T}_0$ be a closed term such that $s \Downarrow$. Then $s \ [\eta]_{\boldsymbol{\lambda}\approx} \ t$ for every closed term $t \in \mathcal{T}_0$ holds.*

The next lemma reflects the fact, that reduction may only cut down on the approximation reduction sequences to an abstraction.

Lemma 4.27. *If $s, t \in \mathcal{T}_0$ are closed terms such that $s \xrightarrow{p}{}^{k} \ t$ then $t \ [\eta]_{\boldsymbol{\lambda}\approx} \ s$.*

Proof. The claim is obvious since with $t \Downarrow_{k'} \lambda x.r$ also $s \Downarrow_{k+k'} \lambda x.r$ holds. □

An example shows that w.r.t. $\lesssim_b$, the $\boldsymbol{\lambda}_{\approx}$-calculus has incomparable abstractions.

Example 4.28. Consider the combinators **K** and **K2**, which both clearly are abstractions. If applied e.g. to the argument **I** we yield

$$\mathbf{K}\,\mathbf{I} \xrightarrow{lbeta} \mathtt{let}\ x = \mathbf{I}\ \mathtt{in}\ \lambda y.x \xrightarrow{cpa} \lambda y.\mathbf{I}$$
$$\mathbf{K2}\,\mathbf{I} \xrightarrow{lbeta} \mathtt{let}\ x = \mathbf{I}\ \mathtt{in}\ \lambda y.y \xrightarrow{cpa} \lambda y.y$$

Hence both again are abstractions but if applied to $\boldsymbol{\Omega}$ as a further argument, the difference becomes apparent:

$$(\lambda y.\mathbf{I})\,\boldsymbol{\Omega} \xrightarrow{lbeta} \mathtt{let}\ y = \boldsymbol{\Omega}\ \mathtt{in}\ \mathbf{I} \xrightarrow{cpa} \mathbf{I}$$
$$(\lambda y.y)\,\boldsymbol{\Omega} \xrightarrow{lbeta} \mathtt{let}\ y = \boldsymbol{\Omega}\ \mathtt{in}\ y \to \ldots$$

Obviously, the term $\mathtt{let}\ y = \boldsymbol{\Omega}\ \mathtt{in}\ y$ has no approximation to an abstraction, thus $\mathbf{K} \not\lesssim_b \mathbf{K2}$. With a similar argument — just applying first to $\boldsymbol{\Omega}$ and then to **I** — one can show that $\mathbf{K2} \not\lesssim_b \mathbf{K}$ holds.

Without further reference, we will often use the following proof principle for similarity which states that $s \lesssim_b t$ holds, if in every approximation reduction sequence from s to an abstraction, we can find a term s' such that there is also an approximation reduction sequence from t to a term t' with $s' \lesssim_b t'$.

Lemma 4.29. *For all closed terms $s, t \in \mathcal{T}_0$ and all preorders $\eta \subseteq {\mathcal{T}_0}^2$ on closed terms the following holds:*

$$(\forall \lambda x.s'' : \ s \xrightarrow{p}{}^{*} \lambda x.s'' \implies (\exists s', t' : \ s \xrightarrow{p}{}^{*} s' \xrightarrow{p}{}^{*} \lambda x.s'' \ \wedge t \xrightarrow{p}{}^{*} t' \ \wedge \ s' \ [\eta]_{\boldsymbol{\lambda}\approx} \ t')) \implies s \ [\eta]_{\boldsymbol{\lambda}\approx} \ t$$

Proof. Since the claim constitutes a central tool, we will give a detailed proof here. So we assume closed terms $s, t \in \mathcal{T}_0$ as well as a preorder $\eta \subseteq {\mathcal{T}_0}^2$ on closed terms, arbitrary but fixed.

To prove $s\ [\eta]_{\boldsymbol{\lambda}\approx}\ t$, we further assume $s \Downarrow \lambda x.s''$, i.e. there is an approximation reduction sequence $s \xrightarrow{p}{}^{*}\ \lambda x.s''$ for a closed abstraction $\lambda x.s'' \in \mathcal{T}_0$ arbitrary but fixed. So under the precondition

$$\exists s', t' :\ s \xrightarrow{p}{}^{*}\ s' \xrightarrow{p}{}^{*}\ \lambda x.s''\ \wedge t \xrightarrow{p}{}^{*}\ t'\ \wedge\ s'\ [\eta]_{\boldsymbol{\lambda}\approx}\ t' \tag{4.7}$$

we have to show that there is a $\lambda y.t''$ such that $t \Downarrow \lambda y.t''$ and for a closed term $r \in \mathcal{T}_0$ arbitrary but fixed, $(\lambda x.s'')\, r\ \eta\ (\lambda y.t'')\, r$ hold. So assume $s', t' \in \mathcal{T}_0$ to be the closed terms given by (4.7), then from $s'\ [\eta]_{\boldsymbol{\lambda}\approx}\ t'$ we obtain for every term to which s' converges to, in particular for $\lambda x.s''$ fixed above, a $\lambda y.t''$ such that $\forall r \in \mathcal{T}_0 :\ (\lambda x.s'')\, r\ \eta\ (\lambda y.t'')\, r$ holds. Since by $t \xrightarrow{p}{}^{*}\ t'$, there is also an approximation reduction sequence from t to $\lambda y.t''$, the claim is shown. □

The following corollary presents a special case of the precedent lemma, where the intermediate terms s', t' coincide with the abstractions at the end of an approximation reduction sequence.

Corollary 4.30. *For all closed terms $s, t \in \mathcal{T}_0$ and all preorders $\eta \subseteq {\mathcal{T}_0}^2$ on closed terms the following property holds:*

$$(\forall s' \in \mathcal{T}_0 :\ s \Downarrow s' \implies (\exists t' \in \mathcal{T}_0 :\ t \Downarrow t' \wedge s'\ [\eta]_{\boldsymbol{\lambda}\approx}\ t')) \implies s\ [\eta]_{\boldsymbol{\lambda}\approx}\ t \tag{4.8}$$

Remark 4.31. Note that instead of lemma 4.29 we also could have shown

$$(\forall s' :\ s \xrightarrow{p}{}^{*}\ s' \implies (\exists t' :\ t \xrightarrow{p}{}^{*}\ t'\ \wedge\ s'\ [\eta]_{\boldsymbol{\lambda}\approx}\ t')) \implies s\ [\eta]_{\boldsymbol{\lambda}\approx}\ t$$

but that is a rather weak condition, not sufficient for a proof of corollary 4.30.

As an instance of lemma 4.29, a single reduction step by rule (lbeta) will occur frequently, hence the following corollary.

Corollary 4.32. *Let $r, \lambda x.s, \lambda x.t \in \mathcal{T}_0$ be closed terms. Then we have*

$$\texttt{let}\ x = r\ \texttt{in}\ s\ [\eta]_{\boldsymbol{\lambda}\approx}\ \texttt{let}\ x = r\ \texttt{in}\ t \quad\iff\quad (\lambda x.s)\, r\ [\eta]_{\boldsymbol{\lambda}\approx}\ (\lambda x.t)\, r$$

Proof. We first notice the reductions $(\lambda x.s)\, r \xrightarrow{[\,],\ lbeta} \texttt{let}\ x = r\ \texttt{in}\ s$ and $(\lambda x.t)\, r \xrightarrow{[\,],\ lbeta} \texttt{let}\ x = r\ \texttt{in}\ t$ respectively. Hence by corollary 4.15 we have

$$(\lambda x.s)\, r \xrightarrow{p}{}^{*}\ \lambda z.q_s \iff \texttt{let}\ x = r\ \texttt{in}\ s \xrightarrow{p}{}^{*}\ \lambda z.q_s$$

as well as

$$(\lambda x.t)\, r \xrightarrow{p}{}^{*}\ \lambda z.q_t \iff \texttt{let}\ x = r\ \texttt{in}\ t \xrightarrow{p}{}^{*}\ \lambda z.q_t$$

so that the claim holds by an application of lemma 4.27 and 4.29 respectively for each of the implications in the proposition. □

With the experiment $[\cdot]_{\boldsymbol{\lambda}\approx}$ in mind, the notion of a simulation is as usual and the results of the preceding paragraphs also apply to simulations.

Definition 4.33. *Let $\eta \subseteq \mathcal{T}^2$ be a preorder on terms. Then η is called a* simulation*, if and only if it is $[\cdot]_{\boldsymbol{\lambda}\approx}$-dense, i.e. $\eta \subseteq [\eta]_{\boldsymbol{\lambda}\approx}$ holds.*

Since the experiment $[\cdot]_{\boldsymbol{\lambda}\approx}$ clearly is monotonic w.r.t. the inclusion ordering on sets, i.e. $\eta_1 \subseteq \eta_2 \implies [\eta_1]_{\boldsymbol{\lambda}\approx} \subseteq [\eta_2]_{\boldsymbol{\lambda}\approx}$, its greatest fixed point exists by the fixed point theorem (cf. e.g. [DP92]).

Definition 4.34. *Define the* similarity $\lesssim_b$ *to be the greatest fixed point of $[\cdot]_{\boldsymbol{\lambda}\approx}$, i.e. $\lesssim_b = \mathrm{gfp}([\cdot]_{\boldsymbol{\lambda}\approx})$ and the* bisimilarity $\sim_b$ *by $s \sim_b t \iff s \lesssim_b t \ \wedge\ t \lesssim_b s$.*

The following conclusions are either obvious from the definitions or their proof can be found in the literature.

Corollary 4.35. *$\lesssim_b$ is a preorder and $\sim_b$ an equivalence relation.*

Lemma 4.36. *The similarity $\lesssim_b$ is the greatest $[\cdot]_{\boldsymbol{\lambda}\approx}$-dense set, i.e. it can be characterised as the union of all $[\cdot]_{\boldsymbol{\lambda}\approx}$-dense sets:*

$$\lesssim_b \;=\; \bigcup \{\eta \mid \eta \subseteq [\eta]_{\boldsymbol{\lambda}\approx}\}$$

Note that similarity then may be represented in the following recursive manner.

$$\begin{aligned} s' \lesssim_b t' \iff (\forall \lambda x.s \in \mathtt{ans}(s') :\ \exists \lambda y.t \in \mathtt{ans}(t') :\\ \forall r :\ r \in \mathcal{T}_0 \implies (\lambda x.s)\, r \lesssim_b (\lambda x.t)\, r) \end{aligned} \quad (4.9)$$

The next example illustrates that the previous statement is especially useful in proving algebraic properties (e.g. like the ones in [HHS00]) for similarity.

Example 4.37. Let $r, s, t \in \mathcal{T}_0$ be closed terms. Then we have

$$r \lesssim_b t \ \wedge\ s \lesssim_b t \implies \mathtt{pick}\, r\, s \lesssim_b t$$

i.e. if t behaves "better" than both r and s, then it is immaterial which one is chosen thereof. So assume $\mathtt{pick}\ r\ s \Downarrow \lambda y.p$ then we obviously may bring the reduction by rule (nd) forward, hence $r \Downarrow \lambda y.p$ or $s \Downarrow \lambda y.p$. Since by the premise we have $r \lesssim_b t$ and $s \lesssim_b t$, by equation (4.9) the claim is shown. □

Especially useful is the result, that (stop)-reductions in general lead to terms that are w.r.t. $\lesssim_b$ smaller. At the same time the following lemma gives a nice example of a bisimulation proof using the fact that $\lesssim_b$ – as a greatest fixed point of $[\cdot]_{\boldsymbol{\lambda}\approx}$ – contains every $[\cdot]_{\boldsymbol{\lambda}\approx}$-dense set.

Lemma 4.38. *Let $s, t \in \mathcal{T}_0$ be terms such that $s \xrightarrow{\mathcal{C},\ stop}{}^{*} t$ then $t \lesssim_b s$ holds.*

Proof. Since $\lesssim_b$ is the union of all $[\cdot]_{\boldsymbol{\lambda}\approx}$-dense sets, we will simply show that the set $S = \{(t, s) \in {\mathcal{T}_0}^2 \mid s \xrightarrow{\mathcal{C},\ stop}{}^{*} t\}$ is $[\cdot]_{\boldsymbol{\lambda}\approx}$-dense, i.e. $S \subseteq [S]_{\boldsymbol{\lambda}\approx}$. So assume $s \xrightarrow{\mathcal{C},\ stop}{}^{*} t$ such that $t \Downarrow \lambda x.t'$. Then by lemma 4.10, there is also a reduction $s \xrightarrow{p}{}^{*} \lambda x.s'$ such that $\lambda x.s' \xrightarrow{\mathcal{C},\ stop}{}^{*} \lambda x.t'$. Hence for an arbitrary but fixed closed term $r \in \mathcal{T}_0$ we also have $(\lambda x.s')\, r \xrightarrow{\mathcal{C},\ stop}{}^{*} (\lambda x.t')\, r$. Thus by $((\lambda x.t')\, r, (\lambda x.s')\, r) \in S$ the claim is shown. □

Corollary 4.39. *Let $s, t \in \mathcal{T}_0$ be closed terms such that $s \lesssim_b t$ holds. Then for every closed term $r \in \mathcal{T}_0$ also $s\,r \lesssim_b t\,r$ holds.*

Proof. The claim is immediate from the definition of $\lesssim_b$ as a greatest fixed point.

It is important to note that the converse of the previous statement, i.e. the correctness of the extensionality rule $(\forall r : s\,r \sim_b t\,r) \implies s \sim_b t$, does not automatically hold.

Corresponding to [Abr90, p. 71], the example $\lambda x.\mathbf{\Omega}\,x \not\lesssim_b \mathbf{\Omega}$ already demonstrates this and we also have some limited kind of extensionality.

Corollary 4.40. *Let $\lambda x.s, \lambda y.t \in \mathcal{T}_0$ be two closed terms such that*

$$(\lambda x.s)\,r \;\lesssim_b\; (\lambda y.t)\,r$$

for every closed term $r \in \mathcal{T}_0$ holds. Then also $\lambda x.s \;\lesssim_b\; \lambda y.t$ is valid.

Proof. Obviously, both $\lambda x.s$ and $\lambda y.t$ converge, and from condition (2.1) in the definition of a lazy computation system, we know that there are no possibilities other than $\lambda x.s \Downarrow \lambda x.s$ and $\lambda y.t \Downarrow \lambda y.t$. Hence we have

$$\forall \lambda x.s \in \mathtt{ans}(\lambda x.s) :\; \exists \lambda y.t \in \mathtt{ans}(\lambda y.t) :\; \forall r :\; r \in \mathcal{T}_0 \implies (\lambda x.s)\,r \;\lesssim_b\; (\lambda y.t)\,r$$

and thus $\lambda x.s \;\lesssim_b\; \lambda y.t$ holds by definition. □

Remark 4.41. We speak of a "limited" extensionality above, since in our non-deterministic setting we don't even have the *conditional* version of the η-rule

$$s \Downarrow \;\wedge\; x \notin \mathcal{FV}(s) \implies \lambda x.s\,x \;\sim_b\; s$$

like in [Abr90, p. 71]. We give $s \equiv \mathtt{pick}\ \mathbf{K}\ \mathbf{K2}$ as a counter-example here. Obviously, $\lambda x.(s\,x) \Downarrow \lambda x.(s\,x)$ but for s to converge, a choice has to be made in advance, i.e. either $s \Downarrow \mathbf{K}$ or $s \Downarrow \mathbf{K2}$. Since $\mathbf{K}$ and $\mathbf{K2}$ are incomparable as shown in example 4.28, neither $\mathbf{K}$ or $\mathbf{K2}$ alone is capable to exhibit the same convergent behaviour as $\lambda x.(s\,x)$ if applied to arguments.

Bringing together the corollaries 4.39 and 4.40, we obtain the following

Lemma 4.42. *For all closed terms $s', t' \in \mathcal{T}_0$ the following is true:*

$$s' \;\lesssim_b\; t' \iff \forall \lambda x.s \in \mathtt{ans}(s') :\; \exists \lambda y.t \in \mathtt{ans}(t') :\; \lambda x.s \;\lesssim_b\; \lambda y.t$$

Proof. By equation (4.9) we only have to show

$$\begin{aligned}\forall \lambda x.s \in \mathtt{ans}(s') :\; \exists \lambda y.t \in \mathtt{ans}(t') :\; \lambda x.s \;\lesssim_b\; \lambda y.t \quad \iff \\ \forall \lambda x.s \in \mathtt{ans}(s') :\; \exists \lambda y.t \in \mathtt{ans}(t') :\; \forall r \in \mathcal{T}_0 :\; (\lambda x.s)\,r \;\lesssim_b\; (\lambda x.t)\,r\end{aligned}$$

of which the "only-if"-part is by corollary 4.39 and the "if"-part by corollary 4.40 respectively. □

The bottom line of the next lemma, that reduction by rule (cpa) complies to bisimilarity, is integral for arguing within the $\boldsymbol{\lambda}_{\approx}$-calculus. In order to show this, we will fall back upon lemma 4.38, that $s \xrightarrow{i,\ stop} t$ implies $t \lesssim_b s$.

Lemma 4.43. *If* $\texttt{let}\ x = s\ \texttt{in}\ t \xrightarrow{\mathcal{S},\ cpa} t[s/x]$ *then* $\texttt{let}\ x = s\ \texttt{in}\ t \sim_b t[s/x]$.

Proof. Since $t[s/x] \lesssim_b \texttt{let}\ x = s\ \texttt{in}\ t$ is clear by the definition of $\lesssim_b$, we only have to show $\texttt{let}\ x = s\ \texttt{in}\ t \lesssim_b t[s/x]$, i.e. that $\xrightarrow{\mathcal{S},\ cpa}$ does not change the result of a $\xrightarrow{p}{}^{k}$-reduction sequence.

By corollary 4.30, it has to be shown that for every reduction sequence $\texttt{let}\ x = s\ \texttt{in}\ t \xrightarrow{p}{}^{*} \lambda y.p$ there is a corresponding reduction sequence for $t[s/x]$, i.e. $t[s/x] \xrightarrow{p}{}^{*} \lambda z.q$ which satisfies $\lambda y.p \lesssim_b \lambda z.q$. By lemma 4.11 there is an approximation reduction $t[s/x] \xrightarrow{p}{}^{*} \lambda z.q$ such that $\lambda z.q \xrightarrow{i,\ stop}{}^{*} \lambda y.p$ holds. Thus by lemma 4.38 the claim is shown. □

We now briefly discuss the adoption of example 3.9 to the $\boldsymbol{\lambda}_{\approx}$-calculus.

Example 4.44. Let again + stand for addition on the Church numerals and the two closed terms $s, t \in \mathcal{T}_0$ be given by

$$s \equiv \lambda x.(\texttt{pick}\ (\texttt{pick}\ (1+2)\ (1+3))\ (\texttt{pick}\ (4+2)\ (4+3)))$$
$$t \equiv \texttt{let}\ y = \texttt{pick}\ 1\ 4\ \texttt{in}\ \lambda x.(y + \texttt{pick}\ 2\ 3)$$

Then like in the $\boldsymbol{\lambda}_{ND}$-calculus, s and t could be distinguished by the context $C \equiv \texttt{let}\ f = [\,]\ \texttt{in}\ (f\ 1) + (f\ 1)$ in the $\boldsymbol{\lambda}_{\approx}$-calculus too: Evaluating $C[s]$ may yield an answer from the set $\{6,7,8,9,10,11,12,13,14\}$ whereas for $C[t]$ e.g. the answer 9 is not possible.

Furthermore, s and t are not bisimilar, either. Actually s is an answer but t is not, but both possible answers $\texttt{ans}(t) = \{\lambda x.(1 + \texttt{pick}\ 2\ 3), \lambda x.(4 + \texttt{pick}\ 2\ 3)\}$ of t possess only insufficiently many choices if applied to some argument e, i.e.

$$\texttt{ans}(\lambda x.(1 + \texttt{pick}\ 2\ 3)\, e) = \{3,4\}$$
$$\texttt{ans}(\lambda x.(4 + \texttt{pick}\ 2\ 3)\, e) = \{6,7\}$$

We conclude the section with a more sophisticated bisimulation proof using the fact that $\lesssim_b$ – as a greatest fixed point of $[\cdot]_{\boldsymbol{\lambda}_{\approx}}$ – contains every $[\cdot]_{\boldsymbol{\lambda}_{\approx}}$-dense set.

Example 4.45 (A Bisimulation Proof). Let $r, s, t \in \mathcal{T}_0$ be closed terms such that we have $r \lesssim_b \texttt{pick}\ s\ t$ and the set $\texttt{ans}(r)$ has, w.r.t. $\lesssim_b$, one greatest element. Then $r \lesssim_b s$ or $r \lesssim_b t$ holds.

First note, that $r \lesssim_b \texttt{pick}\ s\ t$ implies that there must exist a $[\cdot]_{\boldsymbol{\lambda}_{\approx}}$-dense set R which contains $(r, \texttt{pick}\ s\ t)$, i.e. $R \subseteq [R]_{\boldsymbol{\lambda}_{\approx}}$ and $(r, \texttt{pick}\ s\ t) \in R$, hence $(r, \texttt{pick}\ s\ t) \in [R]_{\boldsymbol{\lambda}_{\approx}}$ as well. Since the union of $[\cdot]_{\boldsymbol{\lambda}_{\approx}}$-dense sets again is $[\cdot]_{\boldsymbol{\lambda}_{\approx}}$-dense, it suffices to show that S or T, with $S = R \cup \{(r,s)\}$ and $T = R \cup \{(r,t)\}$ respectively, is $[\cdot]_{\boldsymbol{\lambda}_{\approx}}$-dense.

Obviously, for $r \not\Downarrow$ there is nothing to show, so we assume $\lambda y.p \in \mathcal{T}_0$ to be the greatest closed abstraction such that $r \Downarrow \lambda y.p$ holds. From the premise

$r\ [R]_{\boldsymbol{\lambda}\approx}\ \texttt{pick}\ s\ t$ we have an abstraction $\lambda z.q$ such that $\texttt{pick}\ s\ t \Downarrow \lambda z.q$ and $\forall u \in \mathcal{T}_0 :\ (\lambda y.p)\ u\ R\ (\lambda z.q)\ u$ is satisfied. Note that this $\lambda z.q$ usually depends on $\lambda y.p$ but under the precondition that $\lambda y.p$ is greater than every closed abstraction r converges to, we may fix $\lambda z.q$ here. So in the approximation reduction sequence $\texttt{pick}\ s\ t \xrightarrow{p}{}^{*}\ \lambda z.q$ the reduction by rule (nd) could be brought forward, hence we may argue that $s \Downarrow \lambda z.q$ or $t \Downarrow \lambda z.q$ must hold.

Since these cases behave symmetrically, we assume $s \Downarrow \lambda z.q$ w.l.o.g., for which we will show $S \subseteq [S]_{\boldsymbol{\lambda}\approx}$, i.e. $(a,b) \in S \implies (a,b) \in [S]_{\boldsymbol{\lambda}\approx}$ for which we distinguish the two cases:

- For $(a,b) \in R$ nothing has to be shown since R is $[\cdot]_{\boldsymbol{\lambda}\approx}$-dense.
- If $(a,b) \equiv (r,s)$, we know from what has been said before that $r \Downarrow \lambda y.p$ is the only possibility. Hence $s \Downarrow \lambda z.q$ with $\forall u \in \mathcal{T}_0 :\ (\lambda y.p)\ u\ R\ (\lambda z.q)\ u$ shows the claim, since $[R]_{\boldsymbol{\lambda}\approx} \subseteq [S]_{\boldsymbol{\lambda}\approx}$ by monotonicity of the $[\cdot]_{\boldsymbol{\lambda}\approx}$-operator. □

4.9 Extending similarity to open terms

Though reduction and convergence are defined not only on closed but also on open terms, it is clear that the notion of similarity cannot directly be applied to open terms, since otherwise all variables would be incomparable.

Instead, e.g. the relation $\lesssim_b$ has to hold for all possible assignments of the free variables. But rather than to simply substitute the corresponding terms for, sharing by `let`-environments has to be used, as the following example shows.

Example 4.46. Consider the open terms $f\,f$ and $\texttt{let}\ x = f\ \texttt{in}\ x\,x$ which are contextual equivalent in the $\boldsymbol{\lambda}_{ND}$-calculus since copying variables is permitted (cf. correctness of rule (lcv) in [Kut99]). But demanding the terms to be bisimilar for every closing substitution is not possible: $(f\,f)[\texttt{pick}\ \mathbf{K}\ \mathbf{K2}/f]$ may yield $\mathbf{K}\,\mathbf{K2}$ which converges if successively applied to $\boldsymbol{\Omega}$, $\boldsymbol{\Omega}$ and $\mathbf{K}$, while $(\texttt{let}\ x = f\ \texttt{in}\ x\,x)[\texttt{pick}\ \mathbf{K}\ \mathbf{K2}/f]$ does not.

Since in the $\boldsymbol{\lambda}_{\approx}$-calculus only $\circledcirc$ and abstractions may be copied, it is indeed sufficient to consider these in closing `let`-environments.

Lemma 4.47. *Let $s,t \in \mathcal{T}$ be terms such that $\texttt{let}\ x = r\ \texttt{in}\ s \lesssim_b \texttt{let}\ x = r\ \texttt{in}\ t$ holds for all r which are either $\circledcirc$ or an arbitrary abstraction. Then also $\forall p \in \mathcal{T}_0 :\ \texttt{let}\ x = p\ \texttt{in}\ s \lesssim_b \texttt{let}\ x = p\ \texttt{in}\ t$ holds.*

Proof. Let $p \in \mathcal{T}_0$ be an arbitrary but fixed closed term. Since otherwise nothing has to be shown, we assume that $\texttt{let}\ x = p\ \texttt{in}\ s$ converges. Legitimated by lemma 4.29, we will prove that for every approximation reduction sequence $\texttt{let}\ x = p\ \texttt{in}\ s \xrightarrow{p}{}^{*}\ \lambda z.q$ yielding an abstraction, there will be an intermediate term $s' \in \mathcal{T}$ with $\texttt{let}\ x = p\ \texttt{in}\ s \xrightarrow{p}{}^{*}\ s' \xrightarrow{p}{}^{*}\ \lambda z.q$ and a corresponding reduction sequence for $\texttt{let}\ x = p\ \texttt{in}\ t \xrightarrow{p}{}^{*}\ t'$ such that $s' \lesssim_b t'$ holds. So assuming $\texttt{let}\ x = p\ \texttt{in}\ s \xrightarrow{p}{}^{*}\ \lambda z.q$ arbitrary but fixed, by the standardisation lemma 4.20 this converging reduction sequence may be reordered to

$$\texttt{let}\ x = p\ \texttt{in}\ s \xrightarrow{\texttt{let}\ x=\mathcal{S}\ \texttt{in}\ s}{}^{*}_{\boldsymbol{\lambda}\approx}\ \texttt{let}\ x = p'\ \texttt{in}\ s \xrightarrow{p}{}^{*}\ \lambda z.q$$

for p' being $\circledcirc$ or an abstraction. Obviously, this reduction sequence is also possible for $\texttt{let}\ x = p\ \texttt{in}\ t$, thus we have

$$\texttt{let}\ x = p\ \texttt{in}\ t \xrightarrow{\texttt{let}\ x=\mathcal{S}\ \texttt{in}\ s}{}^{*}_{\boldsymbol{\lambda}_{\approx}}\ \texttt{let}\ x = p'\ \texttt{in}\ t$$

and in conjunction with lemma 4.29 the claim holds by the premise. □

We are now free to define the extension ${\lesssim_b}^o$ of $\lesssim_b$ to open terms in either of these manners, but it seems more suitable for the majority of the following proofs to restrict on bindings to terms which may be copied. Having said this, the following corollary indicates that for terms which will anyway be copied, using substitution or `let`-environments will make no difference.

Corollary 4.48. *Let $s \in \mathcal{T}$ be an open term with $\mathcal{FV}(s) = \{x\}$ and $\lambda z.q \in \mathcal{T}_0$ a closed abstraction. If $p \equiv \circledcirc$ or $p \equiv \lambda z.q$, then $\texttt{let}\ x = p\ \texttt{in}\ s \sim_b s[p/x]$ holds.*

Proof. By lemma 4.43, since reduction by rule (cpa) immediately applies. □

So it is possible to keep the approach with substitutions like in [How96] whenever it is restrained to terms which may be copied.

Definition 4.49. *Let $s, t \in \mathcal{T}$ be (possibly open) terms. We then write $s \lesssim_b^o t$ if and only if $\sigma(s) \lesssim_b \sigma(t)$ holds for all closing substitutions σ whose range $\boldsymbol{rng}(\sigma)$ satisfy $\boldsymbol{rng}(\sigma) \subseteq \{p \in \mathcal{T}_0 \mid p \equiv \circledcirc \vee p \equiv \lambda z.q\}$, i.e. substitutions that map only to $\circledcirc$ or a closed abstraction.*

Of course, it is adequate to consider closing substitutions with a "minimal" domain in the sense, that for open terms $s, t \in \mathcal{T}$ the domain $\boldsymbol{dom}(\sigma)$ of the substitution σ is just the set of free variables, i.e. $\boldsymbol{dom}(\sigma) = \mathcal{FV}(s) \cup \mathcal{FV}(t)$. The following corollary states this more precise.

Corollary 4.50. *Let $s, t \in \mathcal{T}$ be two (possibly open) terms. Then $s \lesssim_b^o t$ if and only if $\sigma(s) \lesssim_b \sigma(t)$ holds for all substitutions σ with $\boldsymbol{dom}(\sigma) = \mathcal{FV}(s) \cup \mathcal{FV}(t)$ and $\boldsymbol{rng}(\sigma) \subseteq \{p \in \mathcal{T}_0 \mid p \equiv \circledcirc \vee p \equiv \lambda z.q\}$.*

Proof. Since the "only if"-part is obvious, we assume $s, t \in \mathcal{T}$ and an arbitrary but fixed closing substitution σ' with $\boldsymbol{rng}(\sigma') \subseteq \{p \in \mathcal{T}_0 \mid p \equiv \circledcirc \vee p \equiv \lambda z.q\}$. Now $\sigma'|_{\mathcal{FV}(s) \cup \mathcal{FV}(t)}$, the restriction of σ' to the set $\mathcal{FV}(s) \cup \mathcal{FV}(t)$ of free variables of s and t, i.e. $\boldsymbol{dom}(\sigma'|_{\mathcal{FV}(s) \cup \mathcal{FV}(t)}) = \mathcal{FV}(s) \cup \mathcal{FV}(t)$, coincides with σ' on its domain, i.e. $x \in \mathcal{FV}(s) \cup \mathcal{FV}(t) \implies \sigma'(x) \equiv \sigma'|_{\mathcal{FV}(s) \cup \mathcal{FV}(t)}(x)$.

Thus we have $\sigma'(s) \equiv \sigma'|_{\mathcal{FV}(s) \cup \mathcal{FV}(t)}(s) \lesssim_b \sigma'|_{\mathcal{FV}(s) \cup \mathcal{FV}(t)}(t) \equiv \sigma'(t)$ by the premise that $\sigma(s) \lesssim_b \sigma(t)$ holds for all substitutions σ with $\boldsymbol{dom}(\sigma) = \mathcal{FV}(s) \cup \mathcal{FV}(t)$ and $\boldsymbol{rng}(\sigma) \subseteq \{p \in \mathcal{T}_0 \mid p \equiv \circledcirc \vee p \equiv \lambda z.q\}$.

In the remainder we often will, without prior notice, make use of the following lemma which provides evidence that the three concepts discussed above are interchangeable.

Lemma 4.51. *Let $s, t \in \mathcal{T}$ be terms such that its free variables may be specified by $\mathcal{FV}(s) \cup \mathcal{FV}(t) = \{x_i \mid 1 \leq i \leq n\}$. Then the following are equivalent:*

1. $\forall \sigma :\ \boldsymbol{rng}(\sigma) \subseteq \{p \in \mathcal{T}_0 \mid p \equiv \circledcirc \vee p \equiv \lambda z.q\} \implies \sigma(s) \lesssim_b \sigma(t)$
2. $\forall p \in \mathcal{T}_0 : (p \equiv \circledcirc \vee p \equiv \lambda z.q) \implies \texttt{let}\ x = p\ \texttt{in}\ s \lesssim_b \texttt{let}\ x = p\ \texttt{in}\ t$
3. $\forall p \in \mathcal{T}_0 : \texttt{let}\ x = p\ \texttt{in}\ s \lesssim_b \texttt{let}\ x = p\ \texttt{in}\ t$

Proof. We assume arbitrary terms $s, t \in \mathcal{T}$ which meet the preconditions and w.l.o.g. $n > 0$, since nothing has to be proven if $s, t \in \mathcal{T}_0$ are closed. We will then show (1) $\iff$ (2) and (2) $\iff$ (3). The first equivalence may be handled in conjunction with corollary 4.48 by an induction on n, the number of free variables. Using lemma 4.47 again for an induction on n, the number of free variables, the implication (2) $\implies$ (3) of the second equivalence holds. The remaining implication (2) $\impliedby$ (3) is obvious. □

Now the following corollary is immediate.

Corollary 4.52. *Let $s, t \in \mathcal{T}$ be terms with free variables $FV = \mathcal{FV}(s) \cup \mathcal{FV}(t)$ and $\phi : \{1, \ldots, |FV|\} \to FV$ be a bijection. Then we have $s \lesssim_b{}^o\ t$ if and only if*

$$\begin{aligned}\texttt{let}\ \phi(1) = p_1\ \texttt{in let}\ \phi(2) = p_2\ \texttt{in} \ldots \texttt{let}\ \phi(|FV|) = p_{|FV|}\ \texttt{in}\ s \lesssim_b \\ \texttt{let}\ \phi(1) = p_1\ \texttt{in let}\ \phi(2) = p_2\ \texttt{in} \ldots \texttt{let}\ \phi(|FV|) = p_{|FV|}\ \texttt{in}\ t\end{aligned}$$

for all closed $p_i \in \mathcal{T}_0$ with $1 \leq i \leq |FV|$ holds.

The following corollary makes remark 4.25 more precise.

Corollary 4.53. *For all closed terms $s', t' \in \mathcal{T}_0$ the following holds:*

$$s' \lesssim_b t' \iff \forall \lambda x.s \in \texttt{ans}(s') :\ \exists \lambda x.t \in \texttt{ans}(t') :\ s \lesssim_b{}^o\ t$$

Proof. We will show that for all closed $\lambda x.s, \lambda x.t \in \mathcal{T}_0$ the conditions $s \lesssim_b{}^o\ t$ and $\forall r :\ r \in \mathcal{T}_0 \implies (\lambda x.s)\, r \lesssim_b (\lambda x.t)\, r$ are equivalent. Therefore first note that using reduction rule (lbeta)

$$\forall r :\ r \in \mathcal{T}_0 \implies (\lambda x.s)\, r \lesssim_b (\lambda x.t)\, r$$

and

$$\forall r :\ r \in \mathcal{T}_0 \implies \texttt{let}\ x = r\ \texttt{in}\ s \lesssim_b \texttt{let}\ x = r\ \texttt{in}\ t$$

are equivalent by corollary 4.32. Since by the prerequisites x is the only free variable in s and t the claim then holds by corollary 4.52. □

Lemma 4.54. *Let $s, t \in \mathcal{T}$ be terms. Then $s \lesssim_b{}^o\ t$ implies $\rho(s) \lesssim_b{}^o\ \rho(t)$ for every substitution ρ with range $\boldsymbol{rng}(\rho) \subseteq \{p \in \mathcal{T}_0 \mid p \equiv \circledcirc \vee p \equiv \lambda z.q\}$ and domain $\boldsymbol{dom}(\rho) \subseteq \mathcal{FV}(s) \cup \mathcal{FV}(t)$.*

Proof. We assume terms $s, t \in \mathcal{T}$ arbitrary but fixed such that $s \lesssim_b{}^o\ t$ holds, from which, by corollary 4.50, we have $\sigma(s) \lesssim_b \sigma(t)$ for all substitutions σ with range $\boldsymbol{rng}(\sigma) = \{p \in \mathcal{T}_0 \mid p \equiv \circledcirc \vee p \equiv \lambda z.q\}$ and domain $\boldsymbol{dom}(\sigma) = \mathcal{FV}(s) \cup \mathcal{FV}(t)$. For every such σ now, $\sigma \circ \rho$ again meets these conditions, thus the claim. □

Combining the previous lemma with corollary 4.52 we obtain the following.

Corollary 4.55. *Let $s,t \in \mathcal{T}$ be terms and $x \in \mathcal{FV}(s) \cup \mathcal{FV}(t)$. Then the following statements are true:*

$$\begin{array}{rl} & s \lesssim_b^o t \iff (\forall p \in \mathcal{T}_0 : \texttt{let}\ x = p\ \texttt{in}\ s \lesssim_b^o \texttt{let}\ x = p\ \texttt{in}\ t) \\ x \notin \mathcal{FV}(s) \implies & (s \lesssim_b^o t \iff (\forall p \in \mathcal{T}_0 : s \lesssim_b^o \texttt{let}\ x = p\ \texttt{in}\ t)) \\ x \notin \mathcal{FV}(t) \implies & (s \lesssim_b^o t \iff (\forall p \in \mathcal{T}_0 : \texttt{let}\ x = p\ \texttt{in}\ s \lesssim_b^o t)) \end{array}$$

The following justifies the assumption in of $(\cdot)^o$ being preorder-preserving.

Lemma 4.56. *The extension $\lesssim_b^o$ of $\lesssim_b$ to open terms is a preorder.*

Proof. Since $s \equiv s[\circledcirc/x]$ and $s \equiv s[\lambda z.q/x]$ for all closed terms $s \in \mathcal{T}_0$ holds, reflexivity and transitivity transfer from $\lesssim_b$ to $\lesssim_b^o$, thus $\lesssim_b^o$ is a preorder. □

As a consequence of the previous lemmas we have:

Corollary 4.57. $\lesssim_b \circ \lesssim_b^o \subseteq \lesssim_b^o$ *holds.*

Furthermore, since $\circledcirc$ is closed and $\texttt{ans}(\circledcirc) = \emptyset$:

Corollary 4.58. $\circledcirc \lesssim_b s$ *holds for every closed term $s \in \mathcal{T}_0$ and $\circledcirc \lesssim_b^o t$ for every term $t \in \mathcal{T}$.*

Conveniently, for a reduction $\xrightarrow{a}$ we speak of soundness w.r.t. some relation η if $s \xrightarrow{a} t$ implies $s\ \eta\ t$.

Corollary 4.59. *If $s \xrightarrow{p,\ a} t$ for some terms $s,t \in \mathcal{T}$ and the reduction $\xrightarrow{p,\ a}$ is sound w.r.t. $\lesssim_b$ ($\gtrsim_b$) on closed terms then also $s \lesssim_b^o t$ ($s \gtrsim_b^o t$) holds.*

Proof. Let $s,t \in \mathcal{T}$ be open terms such that $s \xrightarrow{p,\ a} t$ and w.l.o.g. assume $\xrightarrow{p,\ a}$ to be sound w.r.t. $\lesssim_b$. Then $s \lesssim_b^o t$ follows from definition 4.49 due to the fact that (stop) is the only reduction rule which may affect[1] the free variables. □

We may now carry over a result on closed terms.

Corollary 4.60. *Let $s,t \in \mathcal{T}$ be terms such that $s \lesssim_b^o t$ holds. Then for every term $r \in \mathcal{T}$ also $s\,r \lesssim_b^o t\,r$ holds.*

Proof. Assume terms $s,t \in \mathcal{T}$ such that $s \lesssim_b^o t$ holds. From definition 4.49 it follows that $\sigma(s) \lesssim_b \sigma(t)$ for every closing substitution with the given properties holds. Let σ' an arbitrary extension of σ so that also $\sigma'(r)$ is closed. Then by corollary 4.39 we have $\sigma'(s)\,\sigma'(r) \lesssim_b \sigma'(t)\,\sigma'(r)$ which is equivalent to $\sigma'(s\,r) \lesssim_b \sigma'(t\,r)$, thus $s\,r \lesssim_b^o t\,r$ follows.

[1] Note, that copies of free variables will always be captured by the closing substitution

4.10 The Precongruence Candidate revisited

Note that the precongruence candidate $\widehat{\lesssim_b}$ could now be characterised by

Definition 4.61. *Let the relation $\widehat{\lesssim_b} \subseteq \mathcal{T}^2$ defined by induction:*

- $x \mathrel{\widehat{\lesssim_b}} b$ *if* $x \in V$ *is a variable and* $x \lesssim_b{}^o b$.
- $\tau(\overline{a}_i) \mathrel{\widehat{\lesssim_b}} b$ *if there exists* $\overline{a}'_i$ *such that* $\overline{a}_i \mathrel{\widehat{\lesssim_b}} \overline{a}'_i$ *and* $\tau(\overline{a}'_i) \lesssim_b{}^o b$.

Since $\circledcirc$ has no operands and by corollary 4.58 we have $\circledcirc \lesssim_b{}^o s$ for every term $s \in \mathcal{T}$, from remark 2.11 immediately follows

Corollary 4.62. *Let $s \in \mathcal{T}$ be a term. Then $\circledcirc \mathrel{\widehat{\lesssim_b}} s$ holds.*

4.10.1 Substitution Lemmas

The following substitution lemmas will present an essential step forward to the proof that $\lesssim_b{}^o$ is a precongruence.

The $\circledcirc$-Substitution Lemma states that it is safe for $\widehat{\lesssim_b}$ to replace free variables by $\circledcirc$. A slightly more complex case, i.e. the substitution of a free variable by a value in a term which are both greater w.r.t. $\widehat{\lesssim_b}$ than their respective counterparts, is treated by the Value-Substitution Lemma.

Lemma 4.63 ($\circledcirc$-Substitution Lemma). *For all $b, b' \in \mathcal{T}$ the following holds:*

$$b \mathrel{\widehat{\lesssim_b}} b' \implies b[\circledcirc/x] \mathrel{\widehat{\lesssim_b}} b'[\circledcirc/x]$$

Proof. We use induction on the definition of $\widehat{\lesssim_b}$.

- If $b \equiv y \mathrel{\widehat{\lesssim_b}} b'$ for a variable $y \in V$, then by definition 4.61 we have $y \lesssim_b{}^o b'$. This then, by definition 4.49, means that $\sigma(y) \lesssim_b \sigma(b')$ for all substitutions, that map variables only to $\circledcirc$ or an abstraction, holds. Since this true also for all those σ' with $\sigma'(x) \equiv \circledcirc$, by $\sigma(y[\circledcirc/x]) \lesssim_b \sigma(b'[\circledcirc/x])$ the claim holds.
- For $b \equiv \tau(\overline{b}_i) \mathrel{\widehat{\lesssim_b}} b'$ there must exist $\overline{b}'_i$ such that $\overline{b}_i \mathrel{\widehat{\lesssim_b}} \overline{b}'_i$ and $\tau(\overline{b}'_i) \lesssim_b{}^o b'$, i.e. $\sigma(\tau(\overline{b}'_i)) \lesssim_b \sigma(b')$ holds for all substitutions σ with appropriate range. From the induction hypothesis, we have $\overline{b}_i[\circledcirc/x] \mathrel{\widehat{\lesssim_b}} \overline{b}'_i[\circledcirc/x]$ and since the condition $\sigma(\tau(\overline{b}'_i)) \lesssim_b \sigma(b')$ is satisfied particularly for those substitutions σ with $\sigma(x) \equiv \circledcirc$, we may substitute $\circledcirc$ for x beforehand, hence

  $$\sigma(\tau(\overline{b}'_i)[\circledcirc/x]) \lesssim_b \sigma(b'[\circledcirc/x])$$

  which implies $\tau(\overline{b}'_i)[\circledcirc/x] \lesssim_b{}^o b'[\circledcirc/x]$ and thus we have $\tau(\overline{b}_i[\circledcirc/x])) \mathrel{\widehat{\lesssim_b}} b'[\circledcirc/x]$ by $\sigma(\tau(\overline{b}'_i[\circledcirc/x])) \equiv \sigma(\tau(\overline{b}'_i)[\circledcirc/x])$ and the induction hypothesis in connection with property (4) of lemma 2.13. □

Lemma 4.64 (Value-Substitution Lemma). *For all $b, b' \in \mathcal{T}$ and closed terms $\lambda z.r, \lambda z.r' \in \mathcal{T}_0$ the following holds:*

$$b \mathrel{\widehat{\lesssim_b}} b' \ \wedge\ \lambda z.r \mathrel{\widehat{\lesssim_b}} \lambda z.r' \implies b[\lambda z.r/x] \mathrel{\widehat{\lesssim_b}} b'[\lambda z.r'/x]$$

Proof. Nothing has to be shown if $x \notin \mathcal{FV}(b) \cup \mathcal{FV}(b')$ so assuming $x \in \mathcal{FV}(b) \cup \mathcal{FV}(b')$ we only have to deal with the cases $x \notin \mathcal{FV}(b)$, $x \notin \mathcal{FV}(b')$ and $x \in \mathcal{FV}(b) \cap \mathcal{FV}(b')$. The proof then is by induction on $\widehat{\lesssim_b}$'s definition.

- If $b \equiv y \mathrel{\widehat{\lesssim_b}} b'$ for a variable $y \in V$ then $y \lesssim_b{}^o\ b'$ and we have to distinguish the two cases $y \equiv x$ and $y \not\equiv x$.
  We begin with the latter for which we have $y[\lambda z.r/x] \equiv y \lesssim_b{}^o\ b'$ by the premise. For $x \notin \mathcal{FV}(b')$ nothing has to be shown since $b'[\lambda z.r'/x] \equiv b'$. If $x \in \mathcal{FV}(b')$ this becomes

  $$\forall p \in \mathcal{T}_0 :\ y \lesssim_b{}^o\ \texttt{let}\ x = p\ \texttt{in}\ b'$$

  by corollary 4.55 which must hold for $p \equiv \lambda z.r'$ in particular. Thus $y[\lambda z.r/x] \equiv y \lesssim_b{}^o\ b'[\lambda z.r'/x]$ from the reduction $\texttt{let}\ x = \lambda z.r'\ \texttt{in}\ b' \xrightarrow{[\,],\ cpa} b'[\lambda z.r'/x]$ by lemma 4.43 and corollary 4.59.
  In the case of $y \equiv x \notin \mathcal{FV}(b')$, from $x \lesssim_b{}^o\ b'$ we have

  $$\forall p \in \mathcal{T}_0 :\ \texttt{let}\ x = p\ \texttt{in}\ x \lesssim_b{}^o\ b'$$

  by corollary 4.55. This must hold for $p \equiv \lambda z.r$ in particular, hence

  $$x[\lambda z.r/x] \equiv \lambda z.r \lesssim_b{}^o\ b' \equiv b'[\lambda z.r'/x]$$

  by composition, c.f. corollary 4.57, since $\lambda z.r \lesssim_b \texttt{let}\ x = \lambda z.r\ \texttt{in}\ x$ follows from the reduction $\texttt{let}\ x = \lambda z.r\ \texttt{in}\ x \xrightarrow{cpa} \lambda z.r$ by lemma 4.27. Thus we have $x[\lambda z.r/x] \mathrel{\widehat{\lesssim_b}} b'[\lambda z.r'/x]$ by property (3) of lemma 2.13. If $y \equiv x \in \mathcal{FV}(b')$, then from corollary 4.55 we have

  $$\forall p \in \mathcal{T}_0 :\ \texttt{let}\ x = p\ \texttt{in}\ x \lesssim_b{}^o\ \texttt{let}\ x = p\ \texttt{in}\ b'$$

  which again must hold for $p \equiv \lambda z.r'$ in particular, hence

  $$\lambda z.r' \lesssim_b \texttt{let}\ x = \lambda z.r'\ \texttt{in}\ x \lesssim_b{}^o\ \texttt{let}\ x = \lambda z.r'\ \texttt{in}\ b' \lesssim_b{}^o\ b'[\lambda z.r'/x]$$

  by two applications of lemma 4.43 in connection with corollary 4.59. From this we then have $\lambda z.r' \lesssim_b{}^o\ b'[\lambda z.r'/x]$ by corollary 4.57. Furthermore we have $\lambda z.r \mathrel{\widehat{\lesssim_b}} \lambda z.r'$ from the premises and thus $x[\lambda z.r/x] \equiv \lambda z.r \mathrel{\widehat{\lesssim_b}} b'[\lambda z.r'/x]$ by composition, i.e. property (4) of lemma 2.13.
- If $b \equiv \tau(\bar{b}_i) \mathrel{\widehat{\lesssim_b}} b'$ with $\bar{b}'_i$ such that $\bar{b}_i \mathrel{\widehat{\lesssim_b}} \bar{b}'_i$ and $\tau(\bar{b}'_i) \lesssim_b{}^o\ b'$, then w.l.o.g. we may assume $x \in \mathcal{FV}(\tau(\bar{b}'_i)) \cap \mathcal{FV}(b')$ so that we have[2]

  $$\forall \lambda z.r' \in \mathcal{T}_0 :\ \texttt{let}\ x = \lambda z.r'\ \texttt{in}\ \tau(\bar{b}'_i) \lesssim_b{}^o\ \texttt{let}\ x = \lambda z.r'\ \texttt{in}\ b'$$

  for the particular case $p \equiv \lambda z.r'$ by corollary 4.55. Using lemma 4.43 twice this becomes

  $$\tau(\bar{b}'_i)[\lambda z.r'/x] \lesssim_b{}^o\ b'[\lambda z.r'/x]$$

[2] Note that the following is also valid for $x \notin \mathcal{FV}(\bar{b}'_i)$ and $x \notin \mathcal{FV}(b')$, but we just do the proof without the possible simplifications which corollary 4.55 may give us.

Since $\widehat{\lesssim_b}$ is operator-respecting, we may apply the induction hypothesis

$$b_i[\lambda z.r/x] \mathrel{\widehat{\lesssim_b}} b_i'[\lambda z.r'/x]$$

to τ-terms which results in

$$\tau(\overline{b}_i)[\lambda z.r/x] \equiv \tau(\overline{b}_i[\lambda z.r/x]) \mathrel{\widehat{\lesssim_b}} \tau(\overline{b}_i'[\lambda z.r'/x]) \equiv \tau(\overline{b}_i')[\lambda z.r'/x]$$

and thus $\tau(\overline{b}_i)[\lambda z.r/x] \mathrel{\widehat{\lesssim_b}} b'[\lambda z.r'/x]$ from $\tau(\overline{b}_i')[\lambda z.r'/x] \lesssim_b{}^o\ b'[\lambda z.r'/x]$ by composition along property (4) of lemma 2.13. □

By means of the substitution lemmas it follows that the precongruence candidate is stable under substitutions which map only to $\circledcirc$ or abstractions.

Corollary 4.65. *Let s, t be terms with $s \mathrel{\widehat{\lesssim_b}} t$. Then for every substitution ρ with range $\boldsymbol{rng}(\rho) \subseteq \{p \in \mathcal{T}_0 \mid p \equiv \circledcirc \vee p \equiv \lambda z.q\}$ also $\rho(s) \mathrel{\widehat{\lesssim_b}} \rho(t)$ holds.*

Proof. Obvious from lemma 4.63 and 4.64, since $\widehat{\lesssim_b}$ is reflexive. □

Now we are in the position to show that $\lesssim_b{}^o$ indeed is admissible.

Lemma 4.66. *The extension $\lesssim_b{}^o$ of $\lesssim_b$ to open terms is admissible.*

Proof. Lemma 4.56 implies that $(\cdot)^o$ is preorder-preserving, thus condition (1) is met. Property (2), i.e. $(\lesssim_b{}^o)_0 = \lesssim_b$, follows directly from the definition of $\lesssim_b{}^o$ since $s \equiv s[\circledcirc/x]$ and $s \equiv s[\lambda z.q/x]$ for all closed terms $s \in \mathcal{T}_0$ holds. Property (3) should be clear from the fact that the substitutions in use map open to closed terms and $\widehat{\lesssim_b} \subseteq (\widehat{\lesssim_b}_0)^o$, i.e. property (4), is a consequence of corollary 4.65. □

Sometimes it is convenient to restrict the definition of $\widehat{\lesssim_b}$ such that for $a \mathrel{\widehat{\lesssim_b}} b$ with a closed term a, the intermediate terms $\overline{a}_i'$ are also closed.

Lemma 4.67. *Let $a, b \in \mathcal{T}$ be terms. Then $a \mathrel{\widehat{\lesssim_b}} b$ if and only if one of the following holds:*

- *$a \equiv x$ for a variable $x \in V$ and $x \lesssim_b{}^o b$.*
- *$a \equiv \tau(\overline{a}_i)$ for some operator $\tau \in O$, operands a_i and there exist operands a_i' such that $\overline{a}_i \mathrel{\widehat{\lesssim_b}} \overline{a}_i'$ and $\tau(\overline{a}_i') \lesssim_b{}^o b$ hold with $\mathcal{FV}(a_i') \subseteq \mathcal{FV}(a_i)$ for every i.*

Proof. Since the "if"-part is merely a special case of definition 4.61, we just show the "only-if"-part. Therefore we assume $a \mathrel{\widehat{\lesssim_b}} b$ for a case analysis along the definition of the precongruence candidate.

- If $a \equiv x \mathrel{\widehat{\lesssim_b}} b$ for a variable x then $x \lesssim_b{}^o b$, so $x \mathrel{\widehat{\lesssim_b}} x$ shows the claim.
- For $a \equiv \tau(\overline{a}_i) \mathrel{\widehat{\lesssim_b}} b$, from definition 4.61 we have $\overline{a}_i''$ such that $\overline{a}_i \mathrel{\widehat{\lesssim_b}} \overline{a}_i''$ and $\tau(\overline{a}_i'') \lesssim_b{}^o b$ hold. W.l.o.g. let $\mathcal{FV}(\overline{a}_i'') \setminus \mathcal{FV}(\overline{a}_i) = \{x_i \mid 1 \leq i \leq n\}$. Then we construct the desired term $\tau(\overline{a}_i')$ by substituting every x_i with $\circledcirc$, i.e.

$$\tau(\overline{a}_i') \equiv \tau(\overline{a}_i'')[\circledcirc/x_1, \ldots, \circledcirc/x_n]$$

From $\tau(\overline{a}_i'') \lesssim_b{}^o\ b$, setting $p_i \equiv \circledcirc$ we have

$$\texttt{let}\ x_1 = \circledcirc\ \texttt{in let}\ x_2 = \circledcirc\ \texttt{in} \ldots \texttt{let}\ x_n = \circledcirc\ \texttt{in}\ \tau(\overline{a}_i'') \lesssim_b{}^o\ b$$

by corollary 4.55, hence $\tau(\overline{a}_i') \lesssim_b{}^o\ b$ with some (cpa)-reductions by lemma 4.43. From lemma 4.63 we have $\tau(\overline{a}_i)\ \widehat{\lesssim_b}\ \tau(\overline{a}_i')$, thus the claim holds. □

Corollary 4.68. *Let $\tau(\overline{a}_i) \in \mathcal{T}_0$ be a closed and $b \in \mathcal{T}$ an arbitrary term such that $\tau(\overline{a}_i)\ \widehat{\lesssim_b}\ b$ holds. Then there are operands $\overline{a}_i'$ such that $\tau(\overline{a}_i')$ is closed too and $\overline{a}_i\ \widehat{\lesssim_b}\ \overline{a}_i'$ as well as $\tau(\overline{a}_i') \lesssim_b{}^o\ b$ hold.*

Proof. The claim immediately follows from lemma 4.67 since $\mathcal{FV}(\tau(\overline{a}_i)) = \emptyset$. □

Lemma 4.69. *If $a'\ \widehat{\lesssim_b}\ b$ for closed terms $a', b \in \mathcal{T}_0$ and a' is a value, then there exists a closed value b' such that $b \Downarrow b'$ and $a'\ [\widehat{\lesssim_b}]_{\boldsymbol{\lambda}\approx}\ b'$ as well as $a'\ \widehat{\lesssim_b}\ b'$.*

Proof. A value is of the form $\lambda x.s$, so assuming $a' \equiv \lambda x.s$ we have s' such that

$$s\ \widehat{\lesssim_b}\ s'\ \wedge\ \lambda x.s' \lesssim_b{}^o\ b$$

from $\lambda x.s\ \widehat{\lesssim_b}\ b$ by definition 4.61. Note that we may assume $\lambda x.s'$ to be closed according to corollary 4.68. By the premise b is closed, hence $\lambda x.s' \lesssim_b{}^o\ b$ is equivalent to $\lambda x.s' \lesssim_b b$. Since $\lambda x.s'$ is a value, we have $b \Downarrow b' \equiv \lambda x.t$ such that $\lambda x.s'\ \lesssim_b\ \lambda x.t$ holds by lemma 4.42. Thus not only $a' \equiv \lambda x.s\ \widehat{\lesssim_b}\ \lambda x.t \equiv b'$ but also $a' \equiv \lambda x.s\ [\widehat{\lesssim_b}]_{\boldsymbol{\lambda}\approx}\ \lambda x.t \equiv b'$. □

Corollary 4.70. *For all closed terms $a, a', b \in \mathcal{T}_0$ we have*

$$a \Downarrow a'\ \wedge\ a'\ \widehat{\lesssim_b}_0\ b \implies (\exists b' :\ b \Downarrow b'\ \wedge\ a\ [\widehat{\lesssim_b}_0]_{\boldsymbol{\lambda}\approx}\ b)$$

Proof. From $a'\ \widehat{\lesssim_b}\ b$ we have a closed b' such that $b \Downarrow b'$ and $a'\ [\widehat{\lesssim_b}]_{\boldsymbol{\lambda}\approx}\ b'$ by lemma 4.69 since a' is a value. Thus $a\ [\widehat{\lesssim_b}_0]_{\boldsymbol{\lambda}\approx}\ b$ follows from equation (4.8) since all the terms are closed. □

4.11 Proving $\lesssim_b{}^o$ a precongruence

For proving $\lesssim_b$ a precongruence it will be argued that the relations $\lesssim_b$ and $(\widehat{\lesssim_b})_0$ coincide. According to theorem 2.16, it suffices to show that $(\widehat{\lesssim_b})_0 \subseteq \lesssim_b$ holds, which on the other hand follows from $(\widehat{\lesssim_b})_0 \subseteq [(\widehat{\lesssim_b})_0]_{\boldsymbol{\lambda}\approx}$, since $\lesssim_b$ as a greatest fixed point, contains all $[\cdot]_{\boldsymbol{\lambda}\approx}$-dense sets.

4.11.1 Stability of $\widehat{\lesssim_b}$ under reduction An important step towards this goal is to show that the precongruence candidate relation $\widehat{\lesssim_b}$ is stable under reduction on closed terms, i.e. $s\ \widehat{\lesssim_b}\ t \wedge s \xrightarrow{p} s' \implies s'\ \widehat{\lesssim_b}\ t$ for s, s' to be closed. In the following, this will be done for each of the reduction rules separately. But we would like to restrict the treatment to reductions at top-level, hence we have to show that for every approximation reduction to an abstraction there is also an approximation reduction sequence to the same abstraction which performs reductions only on closed terms.

Definition 4.71. *The class $\mathcal{N}$ of* non-closing surface contexts *is defined by the following rule for the symbol N:*

$$N ::= [\,] \;\mid\; N\,e \;\mid\; e\,N \;\mid\; \texttt{let}\ x = N\ \texttt{in}\ e \;\mid\; \texttt{pick}\ N\ e \;\mid\; \texttt{pick}\ e\ N$$

This means that non-closing surface contexts are just the subset of surface contexts whose construction does not involve L_R-contexts. Thus a surface context $S \in \mathcal{S} \setminus \mathcal{N}$ is called *closing*, i.e. if it is not a non-closing surface context. We now show that reductions inside closing surface contexts can be moved to the end of an approximation reduction sequence.

Lemma 4.72. *A complete set of commuting diagrams for $\xrightarrow{\mathcal{S}\setminus\mathcal{N},\ a}_{\boldsymbol{\lambda}\approx}$-reductions w.r.t. $\xrightarrow{\mathcal{N},\ b}_{\boldsymbol{\lambda}\approx}$-reductions is*

$$\xrightarrow{\mathcal{S}\setminus\mathcal{N},\ a}_{\boldsymbol{\lambda}\approx} \cdot \xrightarrow{\mathcal{N},\ b}_{\boldsymbol{\lambda}\approx} \;\rightsquigarrow\; \xrightarrow{\mathcal{N},\ b}_{\boldsymbol{\lambda}\approx} \cdot \xrightarrow{\mathcal{S}\setminus\mathcal{N},\ a}_{\boldsymbol{\lambda}\approx}$$
$$\xrightarrow{\mathcal{S}\setminus\mathcal{N},\ a}_{\boldsymbol{\lambda}\approx} \cdot \xrightarrow{\mathcal{N},\ b}_{\boldsymbol{\lambda}\approx} \;\rightsquigarrow\; \xrightarrow{\mathcal{N},\ b}_{\boldsymbol{\lambda}\approx}$$

Proof. First note, that an overlapping closing surface context must be of the form $N[L_R[S]]$ for some surface context S and a non-closing N. Since the property (non-) closing is retained when inserting into a non-closing surface context, it suffices to examine the cases $\xrightarrow{L_R[S],\ a}_{\boldsymbol{\lambda}\approx} \cdot \xrightarrow{N,\ b}_{\boldsymbol{\lambda}\approx}$ and $\xrightarrow{N[L_R[S]],\ a}_{\boldsymbol{\lambda}\approx} \cdot \xrightarrow{[\,],\ b}_{\boldsymbol{\lambda}\approx}$. For the former we immediately obtain the diagram

$$\xrightarrow{L_R[S],\ a}_{\boldsymbol{\lambda}\approx} \cdot \xrightarrow{N,\ b}_{\boldsymbol{\lambda}\approx} \;\rightsquigarrow\; \xrightarrow{N,\ b}_{\boldsymbol{\lambda}\approx} \cdot \xrightarrow{L_R[S],\ a}_{\boldsymbol{\lambda}\approx}$$

since a L_R- is disjoint from any N-context. For $\xrightarrow{N[L_R[S]],\ a}_{\boldsymbol{\lambda}\approx} \cdot \xrightarrow{[\,],\ b}_{\boldsymbol{\lambda}\approx}$ we must distinguish along N. Note that $b = \mathit{stop}$ is not possible, since then there would be no approximation reduction to an abstraction.

- For the empty context $N \equiv [\,]$ we have $b = \mathit{cpa}$ and in contrast to the forking diagrams in 4.7, a (stop)-reduction can only be performed inside the target of the (cpa)-reduction:

$$\xrightarrow{L_R[S],\ a}_{\boldsymbol{\lambda}\approx} \cdot \xrightarrow{[\,],\ \mathit{cpa}} \;\rightsquigarrow\; \xrightarrow{[\,],\ \mathit{cpa}} \cdot \xrightarrow{S,\ a}_{\boldsymbol{\lambda}\approx}$$

- The cases $\texttt{pick}\ N'\ e$, $\texttt{pick}\ e\ N'$ imply that the top-level (b)-reduction is (nd), and the (a)-reduction may be dropped:

$$\xrightarrow{N[L_R[S]],\ a}_{\boldsymbol{\lambda}\approx} \cdot \xrightarrow{[\,],\ \mathit{nd}} \;\rightsquigarrow\; \xrightarrow{[\,],\ \mathit{nd}} \cdot \xrightarrow{N[L_R[S]],\ a}_{\boldsymbol{\lambda}\approx}$$
$$\xrightarrow{N[L_R[S]],\ a}_{\boldsymbol{\lambda}\approx} \cdot \xrightarrow{[\,],\ \mathit{nd}} \;\rightsquigarrow\; \xrightarrow{[\,],\ \mathit{nd}}$$

– If $N \equiv e\,N'$ there may be an overlap with $b \in \{lapp, lbeta\}$ which results in

$$\xrightarrow{(\lambda x.s)\,(N'[L_R[S]]),\ a}_{\boldsymbol{\lambda}_\approx} \cdot \xrightarrow{[\,],\ lbeta} \rightsquigarrow \xrightarrow{[\,],\ lbeta} \cdot \xrightarrow{\texttt{let}\ x{=}(N'[L_R[S]])\ \texttt{in}\ s,\ a}_{\boldsymbol{\lambda}_\approx}$$

$$\xrightarrow{(\texttt{let}\ x{=}s\ \texttt{in}\ t)\,(N'[L_R[S]]),\ a}_{\boldsymbol{\lambda}_\approx} \cdot \xrightarrow{[\,],\ lapp} \rightsquigarrow \xrightarrow{[\,],\ lapp} \cdot \xrightarrow{\texttt{let}\ x{=}s\ \texttt{in}\ (t\,(N'[L_R[S]])),\ a}_{\boldsymbol{\lambda}_\approx}$$

– For $N \equiv N'\,e$ only $b = lapp$ is a possible top-level reduction, hence

$$\xrightarrow{(L_R[S])\,e,\ a}_{\boldsymbol{\lambda}_\approx} \cdot \xrightarrow{[\,],\ lapp} \rightsquigarrow \xrightarrow{[\,],\ lapp} \cdot \xrightarrow{L_R[S\,e],\ a}_{\boldsymbol{\lambda}_\approx}$$

– The case $N \equiv \texttt{let}\ x = N'\ \texttt{in}\ e$ is impossible, since the only top-level reduction would be (cpa).

Lemma 4.73. *Let $s, \lambda x.t \in \mathcal{T}_0$ be closed terms such that $s \Downarrow \lambda x.t$ holds. Then there is an approximation reduction sequence of the form $s \xrightarrow{\mathcal{N}\ *}_{\boldsymbol{\lambda}_\approx} \lambda x.t$ too, i.e. using only reductions in non-closing surface contexts.*

Proof. We do the proof by induction on the length k of an approximation reduction sequence $s \xrightarrow{p\ k} \lambda x.t$.

– If $k = 1$, nothing has to be shown, since by remark 4.6 only reductions in the empty context come into question.
– For the induction step, we split a reduction $s \xrightarrow{p\ k+1} \lambda x.t$ into the sequence $s \xrightarrow{p} s' \xrightarrow{p\ k} \lambda x.t$ and assume the reduction $s \xrightarrow{p} s'$ to take place in a closing surface context, since otherwise the claim immediately follows from an easy application of the induction hypothesis.

  So applying the induction hypothesis to the suffix sequence $s' \xrightarrow{p\ k} \lambda x.t$ delivers an approximation reduction $s' \xrightarrow{\mathcal{N}\ k}_{\boldsymbol{\lambda}_\approx} \lambda x.t$. By the diagrams of lemma 4.72, the reduction $s \xrightarrow{p} s'$ either commutes with this sequence or is superfluous.

Corollary 4.74. *Let $s \in \mathcal{T}_0$ be a closed term and $N \in \mathcal{N}$ a non-closing surface context. Then $N[s] \Downarrow \lambda z.q$ implies that there exists some closed abstraction $\lambda x.s' \in \texttt{ans}(s)$ such that $N[s] \xrightarrow{N\ *}_{\boldsymbol{\lambda}_\approx} N[\lambda x.s'] \xrightarrow{\mathcal{S}\ *}_{\boldsymbol{\lambda}_\approx} \lambda z.q$ holds.*

Proof. By lemma 4.73 there is an approximation reduction sequence where reduction only takes place inside non-closing surface contexts. So it remains to show that this sequence can be reordered such that the reductions inside N are performed first. Inspecting the cases of definition 4.71 shows that non-closing surface contexts are disjoint and the only overlapping with the empty context is in the case $\texttt{let}\ x = N\ \texttt{in}\ e$ for which a reduction by rule (cpa) is impossible. □

We will now present a series of lemmas which show that $s \mathrel{\widehat{\lesssim_b}} t \ \wedge \ s \xrightarrow{[\,],\,a}_{\boldsymbol{\lambda}_\approx} s'$ implies $s' \mathrel{\widehat{\lesssim_b}} t$ for each reduction rule (a) of the $\boldsymbol{\lambda}_\approx$-calculus.

Lemma 4.75 (lapp). *Let $s,t,t_x \in \mathcal{T}$ be terms such that $(\mathtt{let}\ x = t_x\ \mathtt{in}\ s)\,t$ is closed. Then we have*

$$((\mathtt{let}\ x = t_x\ \mathtt{in}\ s)\,t \xrightarrow{lapp}_{\boldsymbol{\lambda}_\approx} \mathtt{let}\ x = t_x\ \mathtt{in}\ (s\,t) \ \wedge \\ (\mathtt{let}\ x = t_x\ \mathtt{in}\ s)\,t \mathrel{\widehat{\lesssim_b}} b) \implies \mathtt{let}\ x = t_x\ \mathtt{in}\ (s\,t) \mathrel{\widehat{\lesssim_b}} b$$

Proof. From $(\mathtt{let}\ x = t_x\ \mathtt{in}\ s)\,t \mathrel{\widehat{\lesssim_b}} b$ we have

$$\exists l', t' :\ \mathtt{let}\ x = t_x\ \mathtt{in}\ s \mathrel{\widehat{\lesssim_b}} l' \ \wedge \ t \mathrel{\widehat{\lesssim_b}} t' \ \wedge \ l'\,t' \mathrel{\lesssim_b{}^o} b \tag{4.10}$$

such that, by corollary 4.68, we may assume $l'\,t'$ and hence the subterms l', t' itself to be closed. Furthermore, $\mathtt{let}\ x = t_x\ \mathtt{in}\ s \mathrel{\widehat{\lesssim_b}} l'$ implies

$$\exists x.s', t'_x :\ x.s \mathrel{\widehat{\lesssim_b}} x.s' \ \wedge \ t_x \mathrel{\widehat{\lesssim_b}} t'_x \ \wedge \ \mathtt{let}\ x = t'_x\ \mathtt{in}\ s' \mathrel{\lesssim_b{}^o} l' \tag{4.11}$$

with $\mathtt{let}\ x = t'_x\ \mathtt{in}\ s'$ again to be closed according to corollary 4.68, so that $\mathtt{let}\ x = t'_x\ \mathtt{in}\ s' \lesssim_b l'$ holds on closed terms. Since $\lesssim_b$ is respected by closed A^*_L-contexts, cf. corollary 4.39, we have

$$(\mathtt{let}\ x = t'_x\ \mathtt{in}\ s')\,t' \ \lesssim_b \ l'\,t'$$

from $\mathtt{let}\ x = t'_x\ \mathtt{in}\ s' \ \lesssim_b \ l'$, and hence $(\mathtt{let}\ x = t'_x\ \mathtt{in}\ s')\,t' \mathrel{\lesssim_b{}^o} b$ by corollary 4.57. Since $p \xrightarrow{}{}^* q$ implies $q \ \lesssim_b \ p$ by lemma 4.27, we may apply the (lapp)-reduction also to $(\mathtt{let}\ x = t'_x\ \mathtt{in}\ s')\,t'$ which results in

$$(\mathtt{let}\ x = t'_x\ \mathtt{in}\ s')\,t' \xrightarrow{lapp}_{\boldsymbol{\lambda}_\approx} \mathtt{let}\ x = t'_x\ \mathtt{in}\ s'\,t' \mathrel{\lesssim_b{}^o} b$$

We have $t_x \mathrel{\widehat{\lesssim_b}} t'_x$, $s \mathrel{\widehat{\lesssim_b}} s'$ and $t \mathrel{\widehat{\lesssim_b}} t'$ by construction and $s\,t \mathrel{\widehat{\lesssim_b}} s'\,t'$ since $\mathrel{\widehat{\lesssim_b}}$ is operator-respecting, thus $\mathtt{let}\ x = t_x\ \mathtt{in}\ s\,t \mathrel{\widehat{\lesssim_b}} b$ holds. □

Lemma 4.76 (lbeta). *Let $s,t \in \mathcal{T}$ be terms such that $(\lambda x.s)\,s$ is closed. Then*

$$(\lambda x.s)\,t \xrightarrow{lbeta}_{\boldsymbol{\lambda}_\approx} \mathtt{let}\ x = t\ \mathtt{in}\ s \ \wedge \ (\lambda x.s)\,t \mathrel{\widehat{\lesssim_b}} b \implies \mathtt{let}\ x = t\ \mathtt{in}\ s \mathrel{\widehat{\lesssim_b}} b$$

Proof. Assume $(\lambda x.s)\,t \xrightarrow{lbeta} \mathtt{let}\ x = t\ \mathtt{in}\ s$ and $(\lambda x.s)\,t \mathrel{\widehat{\lesssim_b}} b$. By corollary 4.68, from the latter we have closed terms $f', t' \in \mathcal{T}_0$ such that $(\lambda x.s) \mathrel{\widehat{\lesssim_b}} f'$ and $t \mathrel{\widehat{\lesssim_b}} t'$, as well as $f'\,t' \mathrel{\lesssim_b{}^o} b$ is valid. Expanding $(\lambda x.s) \mathrel{\widehat{\lesssim_b}} f'$ further, we obtain a closed $\lambda x.s' \in \mathcal{T}_0$ which fulfils $s \mathrel{\widehat{\lesssim_b}} s'$ and $\lambda x.s' \mathrel{\lesssim_b{}^o} f'$, hence $\lambda x.s' \ \lesssim_b \ f'$ for the closed relation, too. By lemma 4.42 this in turn means, that there is a closed abstraction $\lambda x.s'' \in \mathcal{T}_0$ such that $f' \Downarrow \lambda x.s''$ and $\lambda x.s' \ \lesssim_b \ \lambda x.s''$ hold.

We obviously may perform the reduction $f' \xrightarrow{p}{}^{*} \lambda x.s''$ also inside the surface context $[\,]\,t'$ so $(\lambda x.s'')\,t' \;\lesssim_b\; f'\,t'$ holds. Since $\lesssim_b$ is respected by closed A_L^*-contexts, we also have $(\lambda x.s')\,t' \;\lesssim_b\; (\lambda x.s'')\,t'$ and furthermore we may reduce $(\lambda x.s')\,t' \xrightarrow{lbeta} \texttt{let}\ x = t'\ \texttt{in}\ s'$ hence the chain

$$\texttt{let}\ x = t'\ \texttt{in}\ s' \;\lesssim_b\; (\lambda x.s')\,t' \;\lesssim_b\; (\lambda x.s'')\,t' \;\lesssim_b\; f'\,t' \lesssim_b{}^{o}\; b$$

Since then $\texttt{let}\ x = t'\ \texttt{in}\ s' \lesssim_b{}^{o}\; b$ holds by corollary 4.57, we complete the proof by recognising that $\texttt{let}\ x = t\ \texttt{in}\ s \mathrel{\widehat{\lesssim_b}} \texttt{let}\ x = t'\ \texttt{in}\ s'$ holds since $\widehat{\lesssim_b}$ is operator-respecting. Thus $\texttt{let}\ x = t\ \texttt{in}\ s \mathrel{\widehat{\lesssim_b}} b$ by property (4) of lemma 2.13. □

Lemma 4.77 (nd, left). *Let $s, t \in \mathcal{T}$ be terms. Then*

$$\texttt{pick}\ s\ t \xrightarrow{nd,left}_{\boldsymbol{\lambda}\approx} s \;\wedge\; \texttt{pick}\ s\ t \mathrel{\widehat{\lesssim_b}} b \implies s \mathrel{\widehat{\lesssim_b}} b$$

Proof. From $\texttt{pick}\ s\ t \mathrel{\widehat{\lesssim_b}} b$ we have

$$\exists s', t' :\; s \mathrel{\widehat{\lesssim_b}} s' \;\wedge\; t \mathrel{\widehat{\lesssim_b}} t' \;\wedge\; \texttt{pick}\ s'\ t' \lesssim_b{}^{o}\; b$$

Since $p \xrightarrow{p}{}^{*} q \implies q \lesssim_b{}^{o}\; p$ by corollary 4.59 we have

$$s' \lesssim_b{}^{o}\; \texttt{pick}\ s'\ t'$$

from $\texttt{pick}\ s'\ t' \xrightarrow{nd,left}_{\boldsymbol{\lambda}\approx} s'$, hence by transitivity

$$s \mathrel{\widehat{\lesssim_b}} s' \lesssim_b{}^{o}\; \texttt{pick}\ s'\ t' \lesssim_b{}^{o}\; b$$

thus $s \mathrel{\widehat{\lesssim_b}} b$ by composition, i.e. property 4 of lemma 2.13. □

By a symmetric argument we have

Lemma 4.78 (nd, right). *Let $s, t \in \mathcal{T}$ be terms. Then*

$$\texttt{pick}\ s\ t \xrightarrow{nd,right}_{\boldsymbol{\lambda}\approx} s \;\wedge\; \texttt{pick}\ s\ t \mathrel{\widehat{\lesssim_b}} b \implies s \mathrel{\widehat{\lesssim_b}} b$$

The case for a reduction by rule (stop) is obvious by corollary 4.62.

Lemma 4.79 (stop). *Let $s \in \mathcal{T}$ be a term such that $s \not\equiv \circledcirc$. Then*

$$s \xrightarrow{stop} \circledcirc \;\wedge\; s \mathrel{\widehat{\lesssim_b}} b \implies \circledcirc \mathrel{\widehat{\lesssim_b}} b$$

The proof for a reduction by rule (cpa) is more involved.

Lemma 4.80 (cpa). *Let $s, t \in \mathcal{T}$ be terms such that $\texttt{let}\ x = s\ \texttt{in}\ t$ is closed. Then the following is true:*

$$\texttt{let}\ x = s\ \texttt{in}\ t \xrightarrow{cpa} t[s/x] \;\wedge\; \texttt{let}\ x = s\ \texttt{in}\ t \mathrel{\widehat{\lesssim_b}} b \implies t[s/x] \mathrel{\widehat{\lesssim_b}} b$$

Proof. We show the cases for $s \equiv \circledcirc$ and $s \equiv \lambda z.q$ from definition 4.1 separately.

– For $s \equiv \circledcirc$ the proposition is

$$\texttt{let}\ x = \circledcirc\ \texttt{in}\ t \xrightarrow{cpa} t[\circledcirc/x]\ \wedge\ \texttt{let}\ x = \circledcirc\ \texttt{in}\ t\ \widehat{\lesssim_b}\ b\ \Longrightarrow\ t[\circledcirc/x]\ \widehat{\lesssim_b}\ b$$

From $\texttt{let}\ x = \circledcirc\ \texttt{in}\ t\ \widehat{\lesssim_b}\ b$, by definition 4.61, we have s', t' such that

$$\circledcirc\ \widehat{\lesssim_b}\ s'\ \wedge\ x.t\ \widehat{\lesssim_b}\ x.t'\ \wedge\ \texttt{let}\ x = s'\ \texttt{in}\ t' \lesssim_b{}^o\ b$$

hence $t[\circledcirc/x]\ \widehat{\lesssim_b}\ t'[\circledcirc/x]$ from $t\ \widehat{\lesssim_b}\ t'$ by lemma 4.63 and furthermore

$$t'[\circledcirc/x] \lesssim_b{}^o\ \texttt{let}\ x = \circledcirc\ \texttt{in}\ t' \lesssim_b{}^o\ \texttt{let}\ x = s'\ \texttt{in}\ t' \lesssim_b{}^o\ b$$

since $\texttt{let}\ x = s'\ \texttt{in}\ t' \xrightarrow{stop} \texttt{let}\ x = \circledcirc\ \texttt{in}\ t' \xrightarrow{cpa} t'[\circledcirc/x]$. Thus $t[\circledcirc/x]\ \widehat{\lesssim_b}\ b$ holds by composition, i.e. property (4) of lemma 2.13.

– If $s \equiv \lambda z.q$ the claim reads as follows

$$\texttt{let}\ x = \lambda z.q\ \texttt{in}\ t \xrightarrow{cpa} t[\lambda z.q/x]\ \wedge\ \texttt{let}\ x = \lambda z.q\ \texttt{in}\ t\ \widehat{\lesssim_b}\ b\ \Longrightarrow\ t[\lambda z.q/x]\ \widehat{\lesssim_b}\ b$$

From $\texttt{let}\ x = \circledcirc\ \texttt{in}\ t\ \widehat{\lesssim_b}\ b$, by corollary 4.68, we have l', t' such that

$$\lambda z.q\ \widehat{\lesssim_b}\ l'\ \wedge\ t\ \widehat{\lesssim_b}\ t'\ \wedge\ \texttt{let}\ x = l'\ \texttt{in}\ t' \lesssim_b{}^o\ b$$

holds and $\texttt{let}\ x = l'\ \texttt{in}\ t'$ is closed, which implies that l' itself is closed. Expanding the definition of $\widehat{\lesssim_b}$ further, from $\lambda z.q\ \widehat{\lesssim_b}\ l'$ we have q' with

$$q\ \widehat{\lesssim_b}\ q'\ \wedge\ \lambda z.q' \lesssim_b{}^o\ l'$$

which obviously implies $\lambda z.q\ \widehat{\lesssim_b}\ \lambda z.q'$ because $\widehat{\lesssim_b}$ is operator-respecting. Since $\lambda z.q'$ again is closed by corollary 4.68, from $\lambda z.q' \lesssim_b{}^o\ l'$ we even have $\lambda z.q'\ \lesssim_b\ l'$, hence $l' \Downarrow \lambda z.q''$ with

$$\lambda z.q' \lesssim_b \lambda z.q''$$

by lemma 4.42, as $\lambda z.q'$ already is an abstraction. So $\lambda z.q\ \widehat{\lesssim_b}\ \lambda z.q''$ follows from $\lambda z.q\ \widehat{\lesssim_b}\ \lambda z.q'$ in connection with $\lambda z.q' \lesssim_b \lambda z.q''$ by composition, i.e. property (4) of lemma 2.13. Furthermore, the reduction $l' \xrightarrow{p}{}^*\ \lambda z.q''$ can also be performed inside the surface context $\texttt{let}\ x = [\,]\ \texttt{in}\ t'$, hence

$$\texttt{let}\ x = l'\ \texttt{in}\ t' \xrightarrow{p}{}^*\ \texttt{let}\ x = \lambda z.q''\ \texttt{in}\ t' \xrightarrow{cpa} t'[\lambda z.q''/x]$$

Since $a \to b$ implies $b \lesssim_b a$ by lemma 4.27, from $\texttt{let}\ x = l'\ \texttt{in}\ t' \lesssim_b b$, by transitivity of $\lesssim_b$ and corollary 4.57, we have

$$t'[\lambda z.q''/x] \lesssim_b{}^o\ b$$

With $t\ \widehat{\lesssim_b}\ t'$ and $\lambda z.q\ \widehat{\lesssim_b}\ \lambda z.q''$ the preconditions of lemma 4.64 are satisfied, thus $t[\lambda z.q/x]\ \widehat{\lesssim_b}\ b$ follows from $t[\lambda z.q/x]\ \widehat{\lesssim_b}\ t'[\lambda z.q''/x]$ and the above by property (4) of lemma 2.13. □

Now we will carry over the preceding results to reductions within non-closing surface contexts.

Lemma 4.81. *Let $p, q \in \mathcal{T}_0$ be closed terms such that $p \xrightarrow{\mathcal{N},\ a}_{\boldsymbol{\lambda}\approx} q$ with some rule (a) of definition 4.1. Then for every term r: $p \mathrel{\widehat{\lesssim_b}} r$ implies $q \mathrel{\widehat{\lesssim_b}} r$.*

Proof. We assume $p \equiv N[p'] \xrightarrow{N,\ a} N[q'] \equiv q$ with $p' \xrightarrow{[\,],\ a} q'$ for some arbitrary, but fixed non-closing surface context $N \in \mathcal{N}$. Then $p', q' \in \mathcal{T}_0$ have to be closed terms by definition 4.71 and we may use induction over the structure of N:

- For $N \equiv [\,]$ the claim holds by one of the lemmas above.
- If $N \equiv N'\,t$ for some surface context $N' \in \mathcal{N}$ and a term $t \in \mathcal{T}$ such that $p \equiv N'[p']$ holds, then from $p \equiv N'[p'] \mathrel{\widehat{\lesssim_b}} r$ we have s_1, s_2 such that
$$N'[p'] \mathrel{\widehat{\lesssim_b}} s_1 \ \wedge\ t \mathrel{\widehat{\lesssim_b}} s_2 \ \wedge\ s_1\,s_2 \mathrel{{\lesssim_b}^o} r$$
by definition 4.61. Since N has only one unique hole, the (a)-reduction may also take place within N', hence
$$N'[p'] \xrightarrow{N',\ a} N'[q']$$
to which we may apply the induction hypothesis, i.e.
$$N'[p'] \xrightarrow{N',\ a} N'[q'] \ \wedge\ N'[p'] \mathrel{\widehat{\lesssim_b}} s_1 \implies N'[q'] \mathrel{\widehat{\lesssim_b}} s_1$$
thus $q \equiv N[q'] \mathrel{\widehat{\lesssim_b}} r$ immediately follows.
- The cases $s\,N'$, `pick` $N'\ t$, `pick` $s\ N'$ and `let` $x = N$ `in` t can be shown accordingly. □

Generalising this, we conclude $s \mathrel{\widehat{\lesssim_b}} t \ \wedge\ s \Downarrow \lambda x.s' \implies s' \mathrel{\widehat{\lesssim_b}} t$ by induction on the length of some $\mathcal{N}$-approximation reduction sequence.

Proposition 4.82. *The restriction $(\widehat{\lesssim_b})_0$ of $\widehat{\lesssim_b}$ to closed terms is a simulation, i.e. the inclusion $(\widehat{\lesssim_b})_0 \subseteq [(\widehat{\lesssim_b})_0]_{\boldsymbol{\lambda}\approx}$ is valid.*

Proof. Let $s, t \in \mathcal{T}_0$ be closed terms such that $s\ (\widehat{\lesssim_b})_0\ t$ holds and assume $s \Downarrow \lambda x.s'$, i.e. $\exists k :\ s \xrightarrow{p}{}^{k} \lambda x.s'$. By lemma 4.73, there is also an approximation reduction sequence $s \xrightarrow{\mathcal{N}}{}^{k'}_{\boldsymbol{\lambda}\approx} \lambda x.s'$ to the same abstraction while using only non-closing surface contexts.

Using lemma 4.81 for an induction on the length k' of this sequence then, also $\lambda x.s' \mathrel{\widehat{\lesssim_b}} t$ is shown. Thus by corollary 4.70, there exists a closed $\lambda x.t'$ such that $t \Downarrow \lambda x.t'$ and $s\ [(\widehat{\lesssim_b})_0]_{\boldsymbol{\lambda}\approx}\ t$ holds. □

Having shown $(\widehat{\lesssim_b})_0$ a simulation, by coinduction it is now an easy consequence that the inclusion $(\widehat{\lesssim_b})_0 \subseteq \lesssim_b$ holds. Together with the admissibility of ${\lesssim_b}^o$, this enables the application of theorem 2.16 in order to establish one of our main results.

Main Theorem 4.83. *The similarity ${\lesssim_b}^o$ is a precongruence.*

As noted before, this is the essential precondition for showing that similarity complies with contextual preorder, which will be addressed in a later section.

5 Approximating $\boldsymbol{\lambda_{ND}}$ by $\boldsymbol{\lambda_{\approx}}$-expressions

So far, we have defined a bisimulation in the $\boldsymbol{\lambda}_{\approx}$-calculus and proven it a congruence. This implies that bisimulation is sound w.r.t. the contextual equivalence in the $\boldsymbol{\lambda}_{\approx}$-calculus. But since our aims were a technique for showing contextual equivalences in the $\boldsymbol{\lambda}_{ND}$-calculus, we have an obligation to show that this is indeed the same as the contextual equivalence in the $\boldsymbol{\lambda}_{\approx}$-calculus.

Hence in this section we define an approximation of $\boldsymbol{\lambda}_{ND}$-terms by sets of $\boldsymbol{\lambda}_{\approx}$-terms so that the contextual congruence will be retained, i.e. essentially the convergent behaviour of terms. We therefore understand the notion of normal-order reduction in $\mathcal{T}(\boldsymbol{\lambda}_{ND})$ as extended to terms from $\mathcal{T}(\boldsymbol{\lambda}_{\approx})$ in the obvious way, i.e. regarding $\circledcirc$ as a constant which has no normal-order reduction.

Definition 5.1 (Approximation Set). *Let $s \in \mathcal{T}(\boldsymbol{\lambda}_{ND})$ be a $\boldsymbol{\lambda}_{ND}$-term. Then $\wr s \wr \subseteq \mathcal{T}(\boldsymbol{\lambda}_{\approx})$, its* approximation set*, is defined as the following set of $\boldsymbol{\lambda}_{\approx}$-terms:*

$$\wr s \wr = \{\lambda x.t \in \mathcal{T}(\boldsymbol{\lambda}_{\approx}) \mid s \xrightarrow{p}{}^{*} \lambda x.t\}$$

Theorem 5.2 (Approximation Theorem). *For all terms $s \in \mathcal{T}$ the following holds: $s \Downarrow_{ND}$ if and only if its approximation set $\wr s \wr$ is non-empty.*

This means the following two implications have to be shown.

1. If there is a normal order reduction sequence starting with s and ending in a WHNF, then there is also an approximation reduction from s to an abstraction.
2. If there is an approximation sequence starting with s and ending in an abstraction, there is also a normal order reduction reduction from s to a WHNF.

5.1 Transforming $\xrightarrow{p}$- into $\xrightarrow{n}_{\boldsymbol{\lambda}_{ND}}$-reduction sequences

We begin with the latter for which commuting diagrams for $\xrightarrow{p}$-reductions are required. We will have to show that for every reduction sequence

$$s_0 \xrightarrow{p,\ a} s_1 \xrightarrow{n,\ b_1}_{\boldsymbol{\lambda}_{ND}} \cdots \xrightarrow{n,\ b_k}_{\boldsymbol{\lambda}_{ND}} s_{k+1} \tag{5.1}$$

ending in a weak head normal form s_{k+1} there is also a reduction sequence

$$s_0 \xrightarrow{n,\ b'_0}_{\boldsymbol{\lambda}_{ND}} s'_1 \xrightarrow{n,\ b'_1}_{\boldsymbol{\lambda}_{ND}} \cdots \xrightarrow{n,\ b'_{m-1}}_{\boldsymbol{\lambda}_{ND}} s'_m \xrightarrow{p,\ a'}{}^{0\vee 1} s'_{m+1} \tag{5.2}$$

where $s'_{m+1} \equiv s_{k+1}$ and s'_m is already a WHNF. Roughly, the proof is by induction on the length k of the first reduction sequence above. The following lemma prunes the number of cases which therefore have to be considered.

Lemma 5.3. *Let S be a surface context which is not a reduction context. Then for all terms $s_0, s_1, s_2 \in \mathcal{T}$, every surface reduction $\xrightarrow{S,\ a}_{\boldsymbol{\lambda}_{\approx}}$ and every normal order reduction $\xrightarrow{n,\ b}_{\boldsymbol{\lambda}_{ND}}$ the following holds:*

$$s_0 \xrightarrow{S,\ a}_{\boldsymbol{\lambda}_{\approx}} s_1 \xrightarrow{n,\ b}_{\boldsymbol{\lambda}_{ND}} s_2 \implies \exists s'_2 : s_0 \xrightarrow{n,\ b}_{\boldsymbol{\lambda}_{ND}} s'_2 \xrightarrow{S,\ a}_{\boldsymbol{\lambda}_{\approx}} s_2$$

$$s_1 \xleftarrow{S,\ a}_{\boldsymbol{\lambda}_{\approx}} s_0 \xrightarrow{n,\ b}_{\boldsymbol{\lambda}_{ND}} s_2 \implies \exists s_3 :\ s_1 \xrightarrow{n,\ b}_{\boldsymbol{\lambda}_{ND}} s_3 \xleftarrow{S,\ a}_{\boldsymbol{\lambda}_{\approx}} s_2$$

Proof. Let R be the reduction context in which $\xrightarrow{n,\ b}_{\boldsymbol{\lambda}_{ND}}$ takes place, i.e. the normal order reduction is $\xrightarrow{R,\ n,\ b}_{\boldsymbol{\lambda}_{ND}}$ in fact. Since S is *not* a reduction context, an examination of the structure of surface and reduction contexts shows that S and R are disjoint. Thus an argument as the one in lemma 2.27 shows the claim. □

Using this lemma, it is easy to show that for every reduction sequence of the form (5.1) there is a corresponding sequence of the form (5.2) whenever the surface context is not a reduction context. So it now suffices to show that for every reduction sequence

$$s_0 \xrightarrow{\mathcal{R},\ a}_{\boldsymbol{\lambda}_{\approx}} s_1 \xrightarrow{n,\ b_1}_{\boldsymbol{\lambda}_{ND}} \cdots \xrightarrow{n,\ b_k}_{\boldsymbol{\lambda}_{ND}} s_{k+1}$$

ending in a weak head normal form s_{k+1} there is also a reduction sequence

$$s_0 \xrightarrow{n,\ b'_0}_{\boldsymbol{\lambda}_{ND}} s'_1 \xrightarrow{n,\ b'_1}_{\boldsymbol{\lambda}_{ND}} \cdots \xrightarrow{n,\ b'_{m-1}}_{\boldsymbol{\lambda}_{ND}} s'_m \xrightarrow{\mathcal{R},\ a'}{}^{0\vee 1}_{\boldsymbol{\lambda}_{\approx}} s'_{m+1}$$

where $s'_{m+1} \equiv s_{k+1}$ and s'_m is already a WHNF. The base case for $k = 0$ of the induction is then covered by the following lemma.

Lemma 5.4. *Let $s, t \in \mathcal{T}$ be arbitrary terms such that $s \xrightarrow{\mathcal{R},\ a}_{\boldsymbol{\lambda}_{\approx}} t$ holds and t is a WHNF. Then either s is a WHNF too, $s \xrightarrow{n}_{\boldsymbol{\lambda}_{ND}} t$ directly, or there is a normal order reduction $s \xrightarrow{n,\ cp}_{\boldsymbol{\lambda}_{ND}} t'$ to a WHNF t' such that $t' \xrightarrow{\mathcal{S},\ cpa}_{\boldsymbol{\lambda}_{\approx}} t$.*

Proof. Let $s \xrightarrow{\mathcal{R},\ a}_{\boldsymbol{\lambda}_{\approx}} t$. Since for a $\xrightarrow{i\mathcal{R},\ a}_{\boldsymbol{\lambda}_{ND}}$-reduction s already is in WHNF by lemma 3.15, we assume that t is a WHNF but s is not. If the reduction $s \xrightarrow{\mathcal{R},\ a}_{\boldsymbol{\lambda}_{\approx}} t$ already is a normal order reduction nothing has to be shown. Hence we only have to distinguish the following cases for a:

- If $a = \mathit{stop}$, we may assume $s \equiv R[s']$. So $t \equiv R[\circledcirc]$ cannot be a WHNF, hence $\xrightarrow{\mathit{stop}}$ is not possible.
- The only possibility for $a = \mathit{cpa}$ is

  $$s \equiv L^*_R[\mathtt{let}\ x = \lambda z.q\ \mathtt{in}\ {L^*_R}'[x]] \xrightarrow{\mathcal{R},\ cpa} L^*_R[({L^*_R}'[x])[\lambda z.q/x]] \equiv t$$

  for which there is also a $\xrightarrow{n,\ cp}_{\boldsymbol{\lambda}_{ND}}$-reduction, namely

  $$s \equiv L^*_R[\mathtt{let}\ x = \lambda z.q\ \mathtt{in}\ {L^*_R}'[x]] \xrightarrow{n,\ cp} L^*_R[\mathtt{let}\ x = \lambda z.q\ \mathtt{in}\ {L^*_R}'[\lambda z.q]] \equiv t'$$

  and since $L^*_R[({L^*_R}'[x])[\lambda z.q/x]] \equiv L^*_R[({L^*_R}'[\lambda z.q])[\lambda z.q/x]]$ we clearly have

  $$t' \equiv L^*_R[\mathtt{let}\ x = \lambda z.q\ \mathtt{in}\ {L^*_R}'[\lambda z.q]] \xrightarrow{\mathcal{R},\ cpa} L^*_R[({L^*_R}'[\lambda z.q])[\lambda z.q/x]] \equiv t$$

  Thus the proposition is shown. □

For the induction step, complete sets of commuting diagrams for $\xrightarrow{\mathcal{R},\ cpa}$ and $\xrightarrow{\mathcal{R},\ stop}$ will be established. This is sufficient because firstly, by lemma 5.3 only reduction contexts have to be taken into account, as noted before. Secondly, the common reduction rules of both calculi, $\boldsymbol{\lambda}_{ND}$ and $\boldsymbol{\lambda}_{\approx}$, could be disregarded according to lemma 3.11, if their application is limited to take place inside reduction contexts. The following lemma states this more precisely.

Lemma 5.5. *Let $s_0, s_1, s_2 \in \mathcal{T}$ be terms and (a) a rule of the $\boldsymbol{\lambda}_{\approx}$-calculus. If $s_0 \xrightarrow{i,\ \mathcal{R},\ a}_{\boldsymbol{\lambda}_{ND}} s_1 \xrightarrow{n} s_2$ then there is a term s_1' such that $s_0 \xrightarrow{n} s_1' \xrightarrow{\mathcal{R},\ a} s_2$ holds.*

Proof. By lemma 3.11 there is no reduction $\xrightarrow{\mathcal{R},\ a}$ with the required property.

Since we have treated all those rules which are shared by $\boldsymbol{\lambda}_{ND}$ and $\boldsymbol{\lambda}_{\approx}$ we may now turn our attention to the ones which are present in $\boldsymbol{\lambda}_{\approx}$ but not in $\boldsymbol{\lambda}_{ND}$.

5.1.1 (cpa) commutes with normal-order reductions. For a reduction by rule (cpa) inside a reduction context $R \in \mathcal{R}$ assume

$$R[\texttt{let}\ x = s\ \texttt{in}\ t] \xrightarrow{R,\ cpa} R[t[s/x]]$$

with s being $\circledcirc$ or an abstraction. The normal-order redex may be independent of where the substitution takes place. Or, a subsequent normal-order reduction may also use the term substituted for x, so a preceding $\xrightarrow{n,\ cp}$ is necessary. Note that in this case s cannot be $\circledcirc$, if $\xrightarrow{n,\ lbeta}$ is the normal-order reduction in question. If $t \equiv \texttt{let}\ y = x\ \texttt{in}\ t'$ itself is the $\xrightarrow{n,\ cp}$-redex, then $s \not\equiv \circledcirc$ either and two successive $\xrightarrow{n,\ cp}$-reductions have to be performed.

$$\xrightarrow{R,\ cpa} \cdot \xrightarrow{n,\ a} \rightsquigarrow \xrightarrow{n,\ a} \cdot \xrightarrow{R,\ cpa}$$

$$\xrightarrow{R,\ cpa} \cdot \xrightarrow{n,\ lbeta} \rightsquigarrow \xrightarrow{n,\ cp} \cdot \xrightarrow{n,\ lbeta} \cdot \xrightarrow{R,\ cpa}$$

$$\xrightarrow{R,\ cpa} \cdot \xrightarrow{n,\ cp} \rightsquigarrow \xrightarrow{n,\ cp} \cdot \xrightarrow{n,\ cp} \cdot \xrightarrow{R,\ cpa}$$

It is noteworthy that here the reduction context for the (cpa)-reduction remains the same. This is also the case, if $t[s/x]$ is an abstraction and in the head position for a $\xrightarrow{lbeta}$-normal-order reduction, where we have to connect a $\xrightarrow{lapp}$-reduction before.

$$\xrightarrow{R,\ cpa} \cdot \xrightarrow{n,\ lbeta} \rightsquigarrow \xrightarrow{n,\ lapp} \cdot \xrightarrow{n,\ lbeta} \cdot \xrightarrow{R,\ cpa}$$

If the abstraction for a $\xrightarrow{lbeta}$-normal-order reduction is created by the preceding $\xrightarrow{cpa}$, a normal-order $\xrightarrow{cp}$ has to be inserted after $\xrightarrow{lapp}$:

$$\xrightarrow{R,\ cpa} \cdot \xrightarrow{n,\ lbeta} \rightsquigarrow \xrightarrow{n,\ lapp} \cdot \xrightarrow{n,\ cp} \cdot \xrightarrow{n,\ lbeta} \cdot \xrightarrow{R,\ cpa}$$

Now we consider $R \equiv L_R^*[\texttt{let } y = [\,] \texttt{ in } R'[x]]$ which may only be the case when $\texttt{let } x = s \texttt{ in } t$ lies inside the normal-order redex. Then possibly a $\xrightarrow{llet}$-reduction has to be prepended, modifying the reduction context where (cpa) takes place.

$$\xrightarrow{L_R^*[\texttt{let } y=[\,] \texttt{ in } R'[x]],\ cpa} \cdot \xrightarrow{n,\ a} \rightsquigarrow \xrightarrow{n,\ llet} \cdot \xrightarrow{n,\ a} \cdot \xrightarrow{L_R^*,\ cpa}$$

Because the effect of the reduction rule (cpa) also consists of a garbage collection, it cannot be simulated by any normal-order reduction. Hence a (cpa)-reduction might not disappear. Since we exhausted all cases, we summarise our results in the following lemma.

Lemma 5.6. *A complete set of commuting diagrams for* $\xrightarrow{\mathcal{R},\ cpa}$ *w.r.t.* $\xrightarrow{n}$ *is:*

$$\xrightarrow{\mathcal{R},\ cpa} \cdot \xrightarrow{n,\ a} \rightsquigarrow \xrightarrow{n,\ a} \cdot \xrightarrow{\mathcal{R},\ cpa}$$

$$\xrightarrow{\mathcal{R},\ cpa} \cdot \xrightarrow{n,\ a} \rightsquigarrow \xrightarrow{n,\ cp} \cdot \xrightarrow{n,\ a} \cdot \xrightarrow{\mathcal{R},\ cpa} \quad \text{if } a \in \{lbeta, cp\}$$

$$\xrightarrow{\mathcal{R},\ cpa} \cdot \xrightarrow{n,\ lbeta} \rightsquigarrow \xrightarrow{n,\ lapp} \cdot \xrightarrow{n,\ cp}{}^{0\vee 1} \cdot \xrightarrow{n,\ lbeta} \cdot \xrightarrow{\mathcal{R},\ cpa}$$

$$\xrightarrow{\mathcal{R},\ cpa} \cdot \xrightarrow{n,\ a} \rightsquigarrow \xrightarrow{n,\ llet} \cdot \xrightarrow{n,\ a} \cdot \xrightarrow{\mathcal{R},\ cpa}$$

All these transformation diagrams in the above lemma share one basic pattern, namely that they do not duplicate the (cpa)-reduction. This property plays a central role in induction proofs, i.e. it leads to termination for the composition of such diagrams. This follows from the strict decrease of a reduction sequence under the multi-set ordering w.r.t. a multi-set, which contains for every (cpa)-reduction the number of normal-order reductions following this (cpa)-reduction.

Now that we have the prerequisites to move a (cpa)-reduction from the front of a normal-order reduction sequence to its tail, we will proceed with the remaining rule (stop).

5.1.2 (stop) commutes with normal-order reductions.

If the reduction rule (stop) is applied within a reduction context, there is no subsequent normal-order reduction.

Lemma 5.7. *Let* $s, t \in \mathcal{T}$ *be terms and* $R \in \mathcal{R}$ *a reduction context such that* $s \xrightarrow{R,\ stop} t$. *Then* t *has no normal-order reduction.*

Proof. Assuming a reduction sequence of the form $\xrightarrow{R_i,\ stop} \cdot \xrightarrow{n,\ R_n,\ a}$ we have to distinguish only a few cases. Clearly, (stop) must not take place "above" the normal-order redex, i.e. $R_n \equiv R_i[C]$ for some further context $C \in \mathcal{R}$. So only $R_i \equiv R_n[[\,]\, e]$ and $R_i \equiv R_n[\texttt{let } x = [\,] \texttt{ in } e]$ are possible reduction contexts for the preceding (stop)-reduction. But $R_n[\circledcirc\, e]$ and $R_n[\texttt{let } x = \circledcirc \texttt{ in } e]$ both have no normal-order reduction, either. □

5.1.3 Commutation of $\xrightarrow{p}$-reductions w.r.t. $\xrightarrow{n}_{\lambda_{ND}}$-reductions. So far, we have seen that only (cpa)- and (stop)-reductions inside reduction contexts are worth to be considered. Before we proceed further towards the proof of the Approximation Theorem we first put the preceding parts together by showing that for every reduction sequence $s \xrightarrow{p} \cdot \xrightarrow{n}{}^{+} t$ such that t is a WHNF, there is either a pure normal-order reduction sequence $s \xrightarrow{n}{}^{+} t$ or a reduction sequence of the form $s \xrightarrow{n}{}^{+} \cdot \xrightarrow{p} \cdot \xrightarrow{n}{}^{*} t$ to the same WHNF. This is accomplished in detail by the following lemma.

Lemma 5.8. *Let $s_0, s_1, s_2, s_3 \in \mathcal{T}$ be terms such that $s_0 \xrightarrow{p} s_1 \xrightarrow{n} s_2 \xrightarrow{n}{}^{k} s_3$ and s_3 is a WHNF. Then there is a $s_1' \in \mathcal{T}$ such that $s_0 \xrightarrow{n}{}^{1\vee 2} s_1' \xrightarrow{p} s_2 \xrightarrow{n}{}^{k} s_3$ holds.*

Proof. We assume terms $s_0, s_1, s_2, s_3 \in \mathcal{T}$ such that $s_0 \xrightarrow{p} s_1 \xrightarrow{n} s_2 \xrightarrow{n}{}^{k} s_3$ and s_3 is a WHNF and analyse the following cases for the $\xrightarrow{p}$-reduction:

- If $s_0 \xrightarrow{S,}_{\boldsymbol{\lambda}_\approx} s_1 \xrightarrow{n} s_2$ with a surface context $S \in \mathcal{S} \setminus \mathcal{R}$ which is not a reduction context, we have a term $s_1' \in \mathcal{T}$ from lemma 5.3 such that $s_0 \xrightarrow{n} s_1' \xrightarrow{S,}_{\boldsymbol{\lambda}_\approx} s_2$ holds. Since we can prolong this to the reduction sequence $s_0 \xrightarrow{n} s_1' \xrightarrow{S,}_{\boldsymbol{\lambda}_\approx} s_2 \xrightarrow{n}{}^{k} s_3$ ending in the WHNF s_3, the proposition holds and for the following cases, w.l.o.g. we may assume the reduction to take place inside a reduction context.
- The case $s_0 \xrightarrow{\mathcal{R},\ stop} s_1$ is impossible since it would contradict lemma 5.7.
- If $s_0 \xrightarrow{\mathcal{R},\ cpa} s_1 \xrightarrow{n} s_2$ then one of the diagrams in lemma 5.6 must be applicable since this set is complete and s_2 by $s_2 \xrightarrow{n}{}^{k} s_3$ reduces to a WHNF. So we have a reduction sequence $s_0 \xrightarrow{n}{}^{1\vee 2} s_1' \xrightarrow{\mathcal{R},\ cpa} s_2$ which can be prolonged to $s_0 \xrightarrow{n}{}^{1\vee 2} s_1' \xrightarrow{\mathcal{R},\ cpa} s_2 \xrightarrow{n}{}^{k} s_3$ yielding the WHNF s_3. □

Now it is an easy induction to show that every $\xrightarrow{p}$-reduction may be moved from the front of a normal-order reduction sequence to its tail.

Lemma 5.9. *Let $s_0, s_1, s_2 \in \mathcal{T}$ be terms such that $s_0 \xrightarrow{p} s_1 \xrightarrow{n}{}^{*} s_2$ and s_2 is a WHNF. Then there is also a reduction sequence of the form $s_0 \xrightarrow{n}{}^{*} s_2' \xrightarrow{p}{}^{0\vee 1} s_2$ with s_2' being already a WHNF.*

Proof. For a proof by induction on the length k of the normal order reduction sequence assume terms s_i with $0 \leq i \leq k+1$ such that $s_0 \xrightarrow{p} s_1$, $s_i \xrightarrow{n} s_{i+1}$ for $i > 0$ and s_{k+1} is a WHNF.

- If $k = 0$ lemma 5.4 already shows the claim.
- For $k > 0$ assume the statement to be true for normal order reduction sequences of length at most $k-1$ and split up the given sequence as follows:

$$s_0 \xrightarrow{p} s_1 \xrightarrow{n} s_2 \xrightarrow{n}{}^{k-1} s_{k+1}$$

Then by lemma 5.8, we either already have $s_0 \xrightarrow{n}{}^{+} s_2$, for which nothing has to be shown, or obtain a term s_1' such that

$$s_0 \xrightarrow{n}{}^{*} s_1' \xrightarrow{p} s_2 \xrightarrow{n}{}^{k-1} s_{k+1}$$

Now the induction hypothesis may be applied to the remaining shorter sequence $s_1' \xrightarrow{p} s_2 \xrightarrow{n}{}^{k-1} s_{k+1}$, thus the claim holds. □

It seems necessary to point out again that the simplicity of the argument in the induction step of the previous lemma is only possible because none of the commuting diagrams for $\xrightarrow{\mathcal{S},\ cpa}$ multiplicates the number of (cpa)-reductions.

5.2 Transforming $\xrightarrow{n}_{\boldsymbol{\lambda}_{ND}}$- into $\xrightarrow{p}$-reduction sequences

We now turn to the first implication which has to be shown for the proof of the Approximation Theorem, i.e. that for every term $s \in \mathcal{T}$ there is a reduction $s \xrightarrow{p}{}^{*} \lambda x.r$ whenever s has a normal order reduction sequence $s \xrightarrow{n}{}^{*}_{\boldsymbol{\lambda}_{ND}} t$ such that t is a WHNF. Hence the following explanations stand in contrast to section 5.1 in that complete sets of forking instead of commuting diagrams have to be used.

A major requirement for transforming normal-order reduction sequences is the ability to eliminate $\xrightarrow{n,\ llet}$-reductions. The following lemma deals with this.

Lemma 5.10. *Let $s, s' \in \mathcal{T}$ be terms such that $s \xrightarrow{n,\ llet} s'$ is a top-level reduction. Then $s' \xrightarrow{p}{}^{*} \lambda x.r$ implies $s \xrightarrow{p}{}^{*} \lambda x.r$, i.e. s also has a surface approximation reduction sequence to the same abstraction.*

Proof. Assume $s, s' \in \mathcal{T}$ which match the given preconditions, i.e. these terms relate as follows:

$$s \equiv \texttt{let}\ x = (\texttt{let}\ y = t_y\ \texttt{in}\ t_x)\ \texttt{in}\ R[x] \xrightarrow{n,\ llet} \texttt{let}\ y = t_y\ \texttt{in}\ (\texttt{let}\ x = t_x\ \texttt{in}\ R[x]) \equiv s' \xrightarrow{p}{}^{*} \lambda x.r$$

By lemma 4.20, we may perform reductions in the outermost `let`-environment first, videlicet in the surface context $\texttt{let}\ y = \mathcal{S}\ \texttt{in}\ \texttt{let}\ x = t_x\ \texttt{in}\ R[x]$. Hence from $s' \xrightarrow{p}{}^{*} \lambda x.r$ we obtain a reduction sequence

$$\texttt{let}\ y = t_y\ \texttt{in}\ (\texttt{let}\ x = t_x\ \texttt{in}\ R[x]) \xrightarrow{\texttt{let}\ y=\mathcal{S}\ \texttt{in let}\ x=t_x\ \texttt{in}\ R[x]}{}^{*}_{\boldsymbol{\lambda}_{\approx}} \texttt{let}\ y = t_y'\ \texttt{in}\ (\texttt{let}\ x = t_x\ \texttt{in}\ R[x]) \xrightarrow{cpa} (\texttt{let}\ x = t_x\ \texttt{in}\ R'[x])[t_y'/y] \xrightarrow{p}{}^{*} \lambda x.r \quad (5.3)$$

with t_y' being $\circledcirc$ or an abstraction now. Note, that for every surface context S also $\texttt{let}\ x = (\texttt{let}\ y = S\ \texttt{in}\ t_x')\ \texttt{in}\ R'[x]$ is a surface context, so we may transfer the above reduction sequences to s, thereby eliminating the $\xrightarrow{n,\ llet}$-reduction.

We therefore apply to $\texttt{let}\ x = (\texttt{let}\ y = t_y\ \texttt{in}\ t_x)\ \texttt{in}\ R[x]$ exactly the same reductions as in equation (5.3), but within surface contexts of the form $\texttt{let}\ x = (\texttt{let}\ y = \mathcal{S}\ \texttt{in}\ t'_x)\ \texttt{in}\ R'[x]$, resulting in

$$\begin{aligned}\texttt{let}\ x = (\texttt{let}\ y = t_y\ \texttt{in}\ t_x)\ \texttt{in}\ R[x] &\xrightarrow{\texttt{let}\ x=(\texttt{let}\ y=\mathcal{S}\ \texttt{in}\ t_x)\ \texttt{in}\ R[x]}{}^{*}_{\boldsymbol{\lambda}\approx} \\ \texttt{let}\ x = (\texttt{let}\ y = t'_y\ \texttt{in}\ t_x)\ \texttt{in}\ R[x] &\xrightarrow{\texttt{let}\ x=[\ ]\ \texttt{in}\ R[x],\ cpa} \\ &\qquad\qquad \texttt{let}\ x = t_x[t'_y/y]\ \texttt{in}\ R[x]\end{aligned}$$

which is syntactically identical to $(\texttt{let}\ x = t_x\ \texttt{in}\ R[x])[t'_y/y]$, thus the claim. □

5.3 Proof of the Approximation Theorem

Now we may establish the whole proof of the main theorem.

Proof (Theorem 5.2). For the "if"-part assume an arbitrary but fixed reduction sequence $s \xrightarrow{p}{}^{m} \lambda x.r$ of length m. We will show that there is also a normal order reduction $s \xrightarrow{n}{}^{*}_{\boldsymbol{\lambda}_{ND}} t$ to a WHNF t by induction on m:

- If $m = 1$ then by lemma 5.4 the proposition holds.
- For the induction step we assume that the claim is valid for a reduction sequence of length $< m$. Now suppose a reduction sequence $s \xrightarrow{\mathcal{S}}{}^{n}_{\boldsymbol{\lambda}\approx} \lambda x.r$ of length m which can be split as follows:

  $$s \xrightarrow{p} s_1 \xrightarrow{p}{}^{m-1} \lambda x.r$$

  By the induction hypothesis[3] there is a normal order reduction sequence $s_1 \xrightarrow{n}{}^{*} t$ where t is a WHNF, i.e. we have the following situation

  $$s \xrightarrow{p} s_1 \xrightarrow{n}{}^{*} t$$

  Now by lemma 5.9 we have $s \xrightarrow{n}{}^{*} t' \xrightarrow{p} t$ with t' already a WHNF. □

For the "only if"-part we will construct a reduction $s \xrightarrow{p}{}^{*} \lambda x.r$, so assume an arbitrary but fixed normal-order reduction sequence $s \xrightarrow{n}{}^{m}_{\boldsymbol{\lambda}_{ND}} t$ of length m such that t is a WHNF.

- From $m = 1$ we have $s \xrightarrow{n} t$ where t is a WHNF and s is not (otherwise there would be no normal-order reduction).
  Now consider the case where this reduction is also a possible $\rightarrow_{\boldsymbol{\lambda}\approx}$-reduction. If $t \equiv \lambda x.r$ nothing has to be shown, so assume $t \equiv L^+_R[\lambda x.r]$ with $L^+_R \in \mathcal{E}$ be a non-empty environment. Then we may apply $\xrightarrow{stop}$ to all the bound terms in E followed by a sequence of $\xrightarrow{cpa}$-reductions which yields $\lambda x.r[\circledcirc/x_i]$ if x_i are the variables bound by E.

[3] Therefore normal-order has to be declared for terms where $\circledcirc$ was introduced.

The normal-order reduction $s \xrightarrow{n} t$ cannot be $\xrightarrow{n,\ llet}$, since then s would already be in WHNF, so $\xrightarrow{n,\ cp}$ is the only remaining possibility for a normal-order reduction which is *not* a $\rightarrow_{\boldsymbol{\lambda}\approx}$-reduction. But this could only be the case for $\texttt{let}\ y = \lambda x.r\ \texttt{in}\ y \xrightarrow{n,\ cp} \texttt{let}\ y = \lambda x.r\ \texttt{in}\ \lambda x.r$ which can be simulated by an application of the (cpa)-rule.

– Assume the proposition holds for normal-order reduction sequences of length $k < m$. Then a sequence $s \xrightarrow{n}{}^{m} t$ may be split up as follows

$$s \xrightarrow{n} s_1 \xrightarrow{n}{}^{m-1} t$$

with t being a WHNF. By the induction hypothesis, $s_1 \xrightarrow{p}{}^{*} \lambda x.r$, hence

$$s \xrightarrow{n} s_1 \xrightarrow{p}{}^{*} \lambda x.r$$

so we examine the possibilities for $s \xrightarrow{n} s_1$:

- If $s \xrightarrow{n} s_1$ is also a $\rightarrow_{\boldsymbol{\lambda}\approx}$-reduction then $s \xrightarrow{p}{}^{*} \lambda x.r$.
- For $L_R^*[\texttt{let}\ x = \lambda y.s\ \texttt{in}\ R[x]] \xrightarrow{n,\ cp} L_R^*[\texttt{let}\ x = \lambda y.s\ \texttt{in}\ R[\lambda y.s]]$ we may apply $\xrightarrow{cpa}$ to $L_R^*[\texttt{let}\ x = \lambda y.s\ \texttt{in}\ R[\lambda y.s]]$ yielding

  $$L_R^*[\texttt{let}\ x = \lambda y.s\ \texttt{in}\ R[\lambda y.s]] \xrightarrow{cpa} L_R^*[(R[\lambda y.s])[\lambda y.s/x]]$$

  for which, from $L_R^*[\texttt{let}\ x = \lambda y.s\ \texttt{in}\ R[\lambda y.s]] \xrightarrow{p}{}^{*} \lambda z.q$ by the induction hypothesis, we obtain an approximation reduction to an abstraction $\lambda z.q'$ such that $\lambda z.q' \xrightarrow{i,\ stop}{}^{*} \lambda z.q$ by lemma 4.11 holds, i.e.

  $$L_R^*[(R[\lambda y.s])[\lambda y.s/x]] \xrightarrow{p}{}^{*} \lambda z.q' \xrightarrow{i,\ stop}{}^{*} \lambda z.q$$

  But we could also reduce the initial term $L_R^*[\texttt{let}\ x = \lambda y.s\ \texttt{in}\ R[x]]$ directly with $\xrightarrow{cpa}$ which results in syntactically the same term

  $$L_R^*[\texttt{let}\ x = \lambda y.s\ \texttt{in}\ R[x]] \xrightarrow{cpa} L_R^*[(R[x])[\lambda y.s/x]]$$

  since $L_R^*[(R[\lambda y.s])[\lambda y.s/x]] \equiv L_R^*[(R[x])[\lambda y.s/x]]$ and thus the claim.
- In the case of

  $$s \equiv L_R^*[\texttt{let}\ x = (\texttt{let}\ y = t_y\ \texttt{in}\ t_x)\ \texttt{in}\ R[x]] \xrightarrow{n,\ llet} L_R^*[\texttt{let}\ y = t_y\ \texttt{in}\ (\texttt{let}\ x = t_x\ \texttt{in}\ R[x])] \equiv s_1$$

  we obtain from theorem 4.23 for s_1 the following reduction sequence

  $$L_R^*[\texttt{let}\ y = t_y\ \texttt{in}\ (\texttt{let}\ x = t_x\ \texttt{in}\ R[x])] \xrightarrow{olf}{}^{*} \texttt{let}\ y = t'_y\ \texttt{in}\ (\texttt{let}\ x = t'_x\ \texttt{in}\ R'[x]) \xrightarrow{p}{}^{*} \lambda x.r \quad (5.4)$$

where $\xrightarrow{olf}{}^*$ proceeds completely inside the surface context L^*_R. Hence these reductions could be taken over for s giving

$$s \equiv L^*_R[\texttt{let } x = (\texttt{let } y = t_y \texttt{ in } t_x) \texttt{ in } R[x]] \xrightarrow{olf}{}^* \texttt{let } x = (\texttt{let } y = t'_y \texttt{ in } t'_x) \texttt{ in } R'[x]$$

Since $\xrightarrow{olf}{}^*$ leads to terms, i.e. $\texttt{let } x = (\texttt{let } y = t'_y \texttt{ in } t'_x) \texttt{ in } R'[x]$ and $\texttt{let } y = t'_y \texttt{ in } (\texttt{let } x = t'_x \texttt{ in } R'[x])$ respectively, that are equal up to an (llet)-reduction which in fact is the top-level normal-order reduction

$$\texttt{let } x = (\texttt{let } y = t'_y \texttt{ in } t'_x) \texttt{ in } R'[x] \xrightarrow{n,\ llet} \texttt{let } y = t'_y \texttt{ in } (\texttt{let } x = t'_x \texttt{ in } R'[x])$$

we may apply lemma 5.10, hence

$$\texttt{let } x = (\texttt{let } y = t'_y \texttt{ in } t'_x) \texttt{ in } R'[x] \xrightarrow{p}{}^* \lambda x.r$$

and thus the claim holds by

$$s \equiv L^*_R[\texttt{let } x = (\texttt{let } y = t_y \texttt{ in } t_x) \texttt{ in } R[x]] \xrightarrow{olf}{}^* \texttt{let } x = (\texttt{let } y = t'_y \texttt{ in } t'_x) \texttt{ in } R'[x] \xrightarrow{p}{}^* \lambda x.r$$

□

However, the actual goal was to show the correspondence of the contextual congruences in $\boldsymbol{\lambda}_{ND}$ and $\boldsymbol{\lambda}_{\approx}$, which the following theorem is good for.

Theorem 5.11. *Let $s, t \in \mathcal{T}(\boldsymbol{\lambda}_{ND})$ be terms in the $\boldsymbol{\lambda}_{ND}$-calculus. Then we have $s \simeq_{\boldsymbol{\lambda}_{ND},\ c} t$ if and only if $s \simeq_{\boldsymbol{\lambda}_{\approx},\ c} t$ holds.*

Proof. The claim follows from $\mathcal{T}(\boldsymbol{\lambda}_{ND}) \subseteq \mathcal{T}(\boldsymbol{\lambda}_{\approx})$ by lemma 2.32 in conjunction with the Approximation Theorem. □

6 Contextual (Pre-) Congruence

Since by theorem 5.11 from the preceeding section, the contextual preorder in $\boldsymbol{\lambda}_{ND}$ matches the one in $\boldsymbol{\lambda}_{\approx}$, for the rest of the paper we will drop the distinction between the two.

Lemma 6.1. *For all terms $s, t \in \mathcal{T}$ the following holds:*

$$s \lesssim_b{}^o\ t \implies s \lesssim_c t$$

Proof. We assume $s \lesssim_b{}^o\ t$ and an arbitrary but fixed context $C \in \mathcal{C}$. Since, by the Main Theorem, $\lesssim_b{}^o$ is a precongruence, we also have $C[s] \lesssim_b{}^o\ C[t]$ from which $C[s] \Downarrow \implies C[t] \Downarrow$ follows. □

However, the inclusion $(\lesssim_c)_0 \subseteq \lesssim_b$ does not hold in general:

Proposition 6.2. *There exist closed terms $s, t \in \mathcal{T}_0$ such that $s\ (\lesssim_c)_0\ t$ holds but $s\ \lesssim_b\ t$ does not.*

Establishing this proposition requires some preparation.

Lemma 6.3 (Surface Context Lemma). *Let $s \in \mathcal{T}$ be a term and $T \subseteq \mathcal{T}$ a countable set of terms satisfying the following: For every surface context $S \in \mathcal{S}$ with $S[s] \Downarrow$ there is a term $t \in T$ such that $S[t] \Downarrow$ holds.*

Then this property is also valid for general contexts, i.e. for every context $C \in \mathcal{C}$ with $C[s] \Downarrow$ there exists $t \in T$ such that $C[t] \Downarrow$ is true.

Proof. Given a term s such that $\forall S \in \mathcal{S} :\ S[s] \Downarrow \implies \exists t \in T :\ S[t] \Downarrow$ holds. For all terms $r \in \mathcal{T}$ we have $r \Downarrow_{ND} \iff r \Downarrow_{\boldsymbol{\lambda}\approx}$ by the Approximation Theorem. Hence the above implication is equivalent to

$$\forall S \in \mathcal{S} :\ S[s] \Downarrow_{ND} \implies \exists t \in T :\ S[t] \Downarrow_{ND}$$

and as every reduction context is also a surface context, lemma 3.16 applies. □

Definition 6.4. *Let $s, t \in \mathcal{T}$ be terms and $i \in \mathbb{N}$ a natural number. Then $s^i\, t$, the i-*fold application *of s to t, is inductively defined as follows:*

$$s^0\, t \equiv t$$
$$s^{i+1}\, t \equiv s\,(s^i\, t)$$

Lemma 6.5. *Let $f \equiv \lambda x.s \in \mathcal{T}_0$ designate a closed abstraction. Then for every abstraction the term $(\lambda y.f\,(y\,y))\,(\lambda z.f\,(z\,z))$ converges to, there is a natural number $i \in \mathbb{N}$ such that $f^i \circledcirc$ reduces to the same abstraction.*

Proof. By induction on the length n of the converging approximation sequence for the term $(\lambda y.f\,(y\,y))\,(\lambda z.f\,(z\,z))$. We therefore distinguish on the first reduction of this sequence:

- Clearly $(\lambda y.f\,(y\,y))\,(\lambda z.f\,(z\,z)) \xrightarrow{\mathcal{S},\ stop} \circledcirc$ may be ruled out, since $\circledcirc$ has no approximation reduction to an abstraction.
- By lemma 4.13, the case $(\lambda y.f\,(y\,y))\,(\lambda z.f\,(z\,z)) \xrightarrow{\mathcal{S},\ stop} \circledcirc\,(\lambda z.f\,(z\,z))$ behaves identically to the previous one.
- The reduction $(\lambda y.f\,(y\,y))\,(\lambda z.f\,(z\,z)) \xrightarrow{\mathcal{S},\ stop} (\lambda y.f\,(y\,y))\circledcirc$ may only be followed by (lbeta) to reach an abstraction — otherwise one of the previous cases would arise. This sequence $\xrightarrow{\mathcal{S},\ stop} \cdot \xrightarrow{\mathcal{S},\ lbeta}$ obviously commutes, i.e. leads to the same abstraction as $\xrightarrow{\mathcal{S},\ lbeta} \cdot \xrightarrow{\mathcal{S},\ stop}$ that together form the induction base.
  The common result of both sequences is $\texttt{let}\ y = \circledcirc\ \texttt{in}\ f\,(y\,y)$ whose reduction may continue with rule (cpa) without changing the length of the sequence by lemma 4.20. This yields $f\,(\circledcirc\,\circledcirc)$ and establishes the claim, since $f^1 \circledcirc$ has the same approximation reduction to an abstraction by lemma 4.13.

– The induction step is given when the use of (stop) is avoided at this early stage. Then only (lbeta) is possible at first:

$$(\lambda y.f\,(y\,y))\,(\lambda z.f\,(z\,z)) \xrightarrow{[\,],\ lbeta} \texttt{let}\ y = (\lambda z.f\,(z\,z))\ \texttt{in}\ f\,(y\,y)$$

By lemma 4.20, we now may perform (cpa) without changing the length of the sequence. Recalling $f \equiv \lambda x.s$, the possible non-(stop)-reductions are all of type (lbeta) and may clearly be commuted. Hence we may assume:

$$\begin{aligned}(\lambda y.f\,(y\,y))\,(\lambda z.f\,(z\,z)) &\xrightarrow{[\,],\ lbeta} \texttt{let}\ y = (\lambda z.f\,(z\,z))\ \texttt{in}\ f\,(y\,y)\\ &\xrightarrow{[\,],\ cpa} f\,((\lambda z'.f\,(z'\,z'))\,(\lambda z''.f\,(z''\,z'')))\\ &\xrightarrow{[\,],\ lbeta} \texttt{let}\ x = ((\lambda z'.f\,(z'\,z'))\,(\lambda z''.f\,(z''\,z'')))\ \texttt{in}\ s\end{aligned}$$

By lemma 4.20 again, we may assume that reduction first proceeds inside the context $\texttt{let}\ x = [\,]\ \texttt{in}\ s$ without changing the length of the sequence at all. But then, since $(\lambda z'.f\,(z'\,z'))\,(\lambda z''.f\,(z''\,z''))$ is syntactically equal to the original term, we may apply the induction hypothesis to it.
Hence there is a natural number i such that $f^i\,\circledcirc$ reduces to the same abstraction as $(\lambda z'.f\,(z'\,z'))\,(\lambda z''.f\,(z''\,z''))$ converges to. Thus the claim is shown since $f^{i+1}\,\circledcirc \xrightarrow{lbeta} \texttt{let}\ x = (f^i\,\circledcirc)\ \texttt{in}\ s$ holds. □

Lemma 6.6. *Let $f \equiv \lambda y.p \in \mathcal{T}_0$ be a closed abstraction and $S \in \mathcal{S}$ a surface context. Then $S[\mathbf{Y}\,f] \Downarrow \lambda z.q$ implies that there is a natural number $i \in \mathbb{N}$ such that also $S[f^i\,\circledcirc]$ has a reduction to $\lambda z.q$, i.e. to the same abstraction.*

Proof. Assuming $S[\mathbf{Y}\,f] \Downarrow \lambda z.q$ lemma 4.18 is applicable, since $\mathbf{Y}$ and f both are closed. Hence we have $\lambda x.s$ such that $\mathbf{Y}\,f \Downarrow \lambda x.s$ and $S[\lambda x.s] \Downarrow \lambda z.q$ hold. Since for $\mathbf{Y}\,f \Downarrow \lambda x.s$ all sensible reduction sequences have to start with $\mathbf{Y}\,f \xrightarrow{\mathcal{S}}{}^*_{\boldsymbol{\lambda}\approx}$ $(\lambda y.f\,(y\,y))\,(\lambda z.f\,(z\,z))$ lemma 6.5 may be used. Therefore, there is a natural number i such that $f^i\,\circledcirc \xrightarrow{\mathcal{S}}{}^*_{\boldsymbol{\lambda}\approx} \lambda x.s$ too. This approximation reduction may be performed within the surface context S, thus $S[f^i\,\circledcirc] \xrightarrow{\mathcal{S}}{}^*_{\boldsymbol{\lambda}\approx} S[\lambda x.s] \xrightarrow{\mathcal{S}}{}^*_{\boldsymbol{\lambda}\approx} \lambda z.q$ proves the claim. □

Lemma 6.7. *Let $f \equiv \lambda x.s \in \mathcal{T}_0$ be a closed abstraction and $C \in \mathcal{C}$ an arbitrary context. Then the implication $C[\mathbf{Y}\,f] \Downarrow \implies \exists i \in \mathbb{N} :\ C[f^i\,\circledcirc] \Downarrow$ is valid.*

Proof. Assume $f \equiv \lambda x.s \in \mathcal{T}_0$ to be a closed abstraction. Then, by lemma 6.6, the implication $\forall S \in \mathcal{S} :\ S[\mathbf{Y}\,f] \Downarrow \implies \exists i \in \mathbb{N} :\ S[f^i\,\circledcirc] \Downarrow$ holds. Therewith, the preconditions for lemma 6.3 are met. □

Corollary 6.8. *Let $f \equiv \lambda x.s \in \mathcal{T}_0$ be a closed abstraction. Then for all contexts $C \in \mathcal{C}$ with $C[\lambda z.(\mathbf{Y}\,f)] \Downarrow$ there is a $\lambda z.(f^i\,\circledcirc)$ such $C[\lambda z.(f^i\,\circledcirc)] \Downarrow$ holds.*

Proof. Let C be an arbitrary context such that $C[\lambda z.(\mathbf{Y}\,f)]$ converges. For the context $D \equiv C[\lambda z.[\,]]$ we obtain $D[\mathbf{Y}\,f] \Downarrow$ since $C[\lambda z.(\mathbf{Y}\,f)] \equiv D[\mathbf{Y}\,f]$. Hence, by lemma 6.7, there exists a $f^i\,\circledcirc$ such that $D[f^i\,\circledcirc] \Downarrow$ which in fact is $C[\lambda z.(f^i\,\circledcirc)] \Downarrow$ as desired. □

Notation. Whenever f denotes some closed abstraction, let G_f stand for the closed term $G_f \equiv (\mathbf{Y}\,(\lambda g.\lambda x.\mathtt{pick}\ (\lambda z.x)\ (g\,(f\,x))))\circledcirc$ in the following.

Corollary 6.9. *Let $f \in \mathcal{T}_0$ be a closed abstraction. Then for $\mathtt{ans}(G_f)$, the answer set, $\mathtt{ans}(G_f) = \{\lambda z.(f^i\circledcirc) \mid i \in \mathbb{N}\}$ holds.*

Proof. By induction on the number i of (nd, right)-reductions in a converging approximation reduction $G_f \xrightarrow{\mathcal{S}}{}^{*}_{\boldsymbol{\lambda}\approx} \lambda z.q$ we will show that $q \equiv f^i\circledcirc$ holds. We therefore unfold the beginning of such a sequence, where we disregard (stop)-reductions, since they will not contribute anything for an abstraction:

$$
\begin{aligned}
&G_f \equiv (\mathbf{Y}\,(\lambda g.\lambda x.\mathtt{pick}\ (\lambda z.x)\ (g\,(f\,x))))\circledcirc\\
&\xrightarrow{lbeta} (\mathtt{let}\ h = (\lambda g.\lambda x.\mathtt{pick}\ (\lambda z.x)\ (g\,(f\,x)))\ \mathtt{in}\ (\lambda y.h\,(y\,y))\,(\lambda y'.h\,(y'\,y')))\circledcirc\\
&\xrightarrow{cpa} ((\lambda y.(\lambda g.\lambda x.\mathtt{pick}\ (\lambda z.x)\ (g\,(f\,x)))\,(y\,y))\,(\lambda y'.(\lambda g.\ldots)\,(y'\,y')))\circledcirc\\
&\xrightarrow{lbeta} \ldots\\
&\xrightarrow{cpa} ((\lambda g.\lambda x.\mathtt{pick}\ (\lambda z.x)\ (g\,(f\,x)))\,((\lambda y'.(\lambda g.\ldots)\,(y'\,y'))\ldots))\circledcirc\\
&\xrightarrow{\mathcal{S}}{}^{*}_{\boldsymbol{\lambda}\approx} \mathtt{pick}\ (\lambda z.\circledcirc)\ w
\end{aligned}
$$

The induction base is given by $\lambda z.\circledcirc$ while the term w is of a form so that the induction hypothesis may be applied. □

Lemma 6.10. *Let f denote a closed abstraction. Then $\lambda z.(\mathbf{Y}\,f) \lesssim_c G_f$ is true.*

Proof. Assuming $f \equiv \lambda x.s \in \mathcal{T}_0$ and a context C satisfying $C[\lambda z.(\mathbf{Y}\,f)]\Downarrow$, we have to show that $C[G_f]\Downarrow$ holds, too. By corollary 6.8 there is a number i such that $C[\lambda z.f^i\circledcirc]$ converges. Furthermore, by corollary 6.9, the reduction $G_f \xrightarrow{\mathcal{S}}{}^{*}_{\boldsymbol{\lambda}\approx} \lambda z.(f^i\circledcirc)$ exists, hence $\lambda z.(f^i\circledcirc) \lesssim_b G_f$ by lemma 4.27. From this we have $C[\lambda z.(f^i\circledcirc)] \lesssim_b C[G_f]$ since $\lesssim_b$ is a precongruence, and thus, by the definition of $\lesssim_b$, the claim holds. □

Corollary 6.11. *The inequation $\lambda z.(\mathbf{Y}\,\mathbf{K})\ (\lesssim_c)_0\ G_{\mathbf{K}}$ is true.*

Lemma 6.12. *The inequation $\lambda z.(\mathbf{Y}\,\mathbf{K}) \lesssim_b G_{\mathbf{K}}$ is false.*

Proof. Assume $\lambda z.(\mathbf{Y}\,\mathbf{K}) \lesssim_b G_{\mathbf{K}}$ for a proof by contradiction. Since $\lambda z.(\mathbf{Y}\,\mathbf{K})$ already is an abstraction, there has to be a closed abstraction $\lambda z.q \in \mathcal{T}_0$ such that $G_{\mathbf{K}}\Downarrow \lambda z.q$ and $\lambda z.(\mathbf{Y}\,\mathbf{K}) \lesssim_b \lambda z.q$ hold. By corollary 6.9, q has to be of the form $q \equiv \lambda z.(\mathbf{K}^i\circledcirc)$. Fixing some i we obtain $(\lambda z.(\mathbf{Y}\,\mathbf{K}))\circledcirc\ldots\circledcirc\Downarrow$ for $i+2$ many applications to $\circledcirc$, but $(\lambda z.\mathbf{K}^i\circledcirc)\circledcirc\ldots\circledcirc$, with $\circledcirc\ldots\circledcirc$ representing the same argument sequence, does not converge. □

Proof (Proposition 6.2). By corollary 6.11 and lemma 6.12. □

7 Conclusion and Future Work

We have seen how the framework of Howe may be extended to cope with sharing. For a non-deterministic call-by-need calculus we have developed a bisimulation and proven it to be equivalent to contextual equivalence. Here, two points emerged to be of great importance. First, we have seen that testing terms by just reducing them to weak head normal form and applying these WHNF's to arbitrary arguments is not appropriate. Instead, the terms to be tested have rather be equipped with all the information about which choices have to be shared and which may be copied. We accomplished this by performing evaluation inside surface contexts up to every arbitrary depth, in which also choices in `let`-environments may be forced. Since we non-deterministically collect all these possible outcomes, the bisimulation then has as much potential to discriminate terms as the contextual equivalence. Moreover, as the proof of lemma 4.38 and the examples 4.37 and 4.45 and illustrate, with bisimulation we have a method at hand for which proofs often require only finitely many steps whereas for contextual equivalence infinitely many contexts have to be considered.

The other aspect concerns which kind of terms may be copied. As we have seen, the precongruence candidate or, strictly speaking, the extension of the bisimilarity to open terms had to be adapted such that only $\circledcirc$ and abstractions are considered. This might point out a general approach for the proof of the fundamental substitution lemma, i.e. [How89, Lemma 1] and [How96, Lemma 3.2] respectively, or lemma 4.63 and 4.64 in our case, to go through.

On the basis of these explanations, we feel confident that the technique demonstrated in this paper is powerful enough for the treatment of a language extending the $\boldsymbol{\lambda}_{ND}$-calculus with a `case` and data constructors. However, introducing a `letrec`-construct, i.e. recursive bindings as e.g. in [MSC99a,SS03], seems to be non-trivial, since a straightforward encoding by fixed point combinators may duplicate non-deterministic choices and therefore is not appropriate for a lazy semantics (cf. [Lau93]).

But a further extension of the $\boldsymbol{\lambda}_{ND}$-calculus which seems possible, is to incorporate divergent behaviour in the definition of the contextual equivalence. As the work of [MSC99a,Kut99,SS03] suggests, it is quite reasonable in a non-deterministic calculus to regard possibly infinite reduction sequences.

8 Acknowledgements

I would like to express my gratitude to Manfred Schmidt-Schauß and David Sabel for their constructive comments. I am particularly indebted to Manfred Schmidt-Schauß for his helpful suggestions and careful proof-reading.

References

[Abr90] Samson Abramsky. The lazy lambda calculus. In David A. Turner, editor, *Research Topics in Functional Programming*, University of Texas at Austin

Year of Programming Series, chapter 4, pages 65–116. Addison-Wesley, 1990.

[ACCL91] Martín Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. *Journal of Functional Programming*, 1991.

[AF97] Zena M. Ariola and Matthias Felleisen. The call-by-need lambda calculus. *Journal of Functional Programming*, 7(3):265–301, 1997.

[AFM+95a] 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.

[AFM+95b] Zena M. Ariola, Matthias Felleisen, John Maraist, Martin Odersky, and Philip Wadler. A call-by-need lambda calculus. In *Proceedings of 22nd Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL)*, pages 233–246, 1995.

[Bar84] Hendrik Pieter Barendregt. *The Lambda Calculus, Its Syntax and Semantics*. Elsevier Science Publishers, 1984.

[BKvO98] Marc Bezem, Jan Willem Klop, and Vincent van Oostrom. Diagram techniques for confluence. *Information and Computation*, 141(2):172–204, March 1998.

[Bou94] Gérard Boudol. Lambda-calculi for (strict) parallel functions. 108(1):51–127, January 1994.

[DP92] B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, Cambridge, 1992.

[dP95] Ugo de'Liguoro and Adolfo Piperno. Nondeterministic extensions of untyped λ-calculus. *Information and Computation*, 122(2):149–177, November 1995.

[Gor94] Andrew D. Gordon. *Functional programming and input/output*. Distinguished Dissertations in Computer Science. Cambridge University Press, September 1994.

[Gor99] Andrew D. Gordon. Bisimilarity as a theory of functional programming. *Theoretical Computer Science*, 228(1-2):5–47, October 1999.

[HHS00] C. A. R. Hoare, He Jifeng, and A. Sampaio. Algebraic derivation of an operational semantics. In Gordon Plotkin, Colin Stirling, and Mads Tofte, editors, *Proof, language, and interaction: essays in honour of Robin Milner*, Foundation of Computing, chapter 3, pages 77–98. MIT Press, Cambridge, Massachusetts, 2000.

[How89] Douglas J. Howe. Equality in lazy computation systems. In *Proceedings, Fourth Annual Symposium on Logic in Computer Science*, pages 198–203, Asilomar Conference Center, Pacific Grove, California, 5–8 June 1989. IEEE Computer Society Press.

[How96] Douglas J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2):103–112, 1 February 1996.

[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.

[Kut99] Arne Kutzner. *Ein nichtdeterministischer call-by-need Lambda-Kalkül mit erratic Choice: Operationale Semantik, Programmtransformationen und Anwendungen*. PhD thesis, Johann Wolfgang Goethe-Universität, Frankfurt, October 1999.

[KvOvR93] Jan Willem Klop, Vincent van Oostrom, and Femke van Raamsdonk. Combinatory reduction systems: introduction and survey. *Theoretical Computer Science*, 121(1–2):279–308, 1993.

[Lau93] John Launchbury. A natural semantics for lazy evaluation. In *Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 144–154. ACM Press, 1993.

[LP00] Soren Lassen and Corin Pitcher. Similarity and bisimilarity for countable non-determinism and higher-order functions. In Andrew Gordon, Andrew Pitts, and Carolyn Talcott, editors, *Electronic Notes in Theoretical Computer Science*, volume 10. Elsevier, 2000.

[Mil71] Robin Milner. An algebraic definition of simulation between programs. In D. C. Cooper, editor, *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*, pages 481–489, London, September 1971. William Kaufmann.

[Mil77] Robin Milner. Fully Abstract Models of Typed lambda-Calculi. *Theoretical Computer Science*, 4(1):1–22, 1977.

[MN98] Richard Mayr and Tobias Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192:3–29, 1998.

[Mor68] J.H. Morris. *Lambda-Calculus Models of Programming Languages.* PhD thesis, MIT, 1968.

[MOW98] John Maraist, Martin Odersky, and Philip Wadler. The call-by-need lambda calculus. *Journal of Functional Programming*, 8(3), May 1998.

[MSC99a] A. K. Moran, D. Sands, and M. Carlsson. Erratic Fudgets: A semantic theory for an embedded coordination language. In *the Third International Conference on Coordination Languages and Models; COODINATION'99*, number 1594 in Lecture Notes in Computer Science, pages 85–102. Springer-Verlag, April 1999. Extended available: [MSC99b].

[MSC99b] A. K. Moran, D. Sands, and M. Carlsson. Erratic Fudgets: A semantic theory for an embedded coordination language (extended version). Extended version of [MSC99a], February 1999.

[Ong93] C.-H. Luke Ong. Non-determinism in a functional setting. In *Logic in Computer Science*, pages 275–286, 1993.

[Par81] David Park. Concurrency and automata on infinite sequences. In Peter Deussen, editor, *Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer, 1981.

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

[San] Davide Sangiorgi. The lazy lambda calculus in a concurrency scenario. pages 120–153, 15 .

[San91] D. Sands. Operational theories of improvement in functional languages (extended abstract). In *Proceedings of the Fourth Glasgow Workshop on Functional Programming*, Workshops in Computing Series, pages 298–311, Skye, August 1991. Springer-Verlag.

[San97] David Sands. From SOS rules to proof principles. Technical report, Chalmers University of Technology and Göteborg University, 1997.

[SS92] Harald Søndergard and Peter Sestoft. Non-determinism in Functional Languages. *The Computer Journal*, 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 am Main, September 2003.