Extending Abramsky's Lazy Lambda Calculus: (Non)-Conservativity of Embeddings

Manfred Schmidt-Schauss[1] and Elena Machkasova[2] and David Sabel[1]

[1] Dept. Informatik und Mathematik, Inst. Informatik, J.W. Goethe-University, PoBox 11 19 32, D-60054 Frankfurt, Germany,
{schauss,sabel}@ki.informatik.uni-frankfurt.de
[2] Division of Science and Mathematics,
University of Minnesota, Morris, MN 56267-2134, U.S.A
elenam@morris.umn.edu

Technical Report Frank-51

Research group for Artificial Intelligence and Software Technology
Institut für Informatik,
Fachbereich Informatik und Mathematik,
Johann Wolfgang Goethe-Universität,
Postfach 11 19 32, D-60054 Frankfurt, Germany

August 22, 2013[3]

Abstract. Our motivation is the question whether the lazy lambda calculus, a pure lambda calculus with the leftmost outermost rewriting strategy, considered under observational semantics, or extensions thereof, are an adequate model for semantic equivalences in real-world purely functional programming languages, in particular for a pure core language of Haskell. We explore several extensions of the lazy lambda calculus: addition of a seq-operator, addition of data constructors and case-expressions, and their combination, focusing on conservativity of these extensions. In addition to untyped calculi, we study their monomorphically and polymorphically typed versions. For most of the extensions we obtain non-conservativity which we prove by providing counterexamples. However, we prove conservativity of the extension by data constructors and case in the monomorphically typed scenario.

1 Introduction

We are interested in reasoning about the semantics of lazy functional programming languages such as Haskell [Pey03], in particular in semantical equivalences of expressions and, as a more general issue, in correctness of program translations and transformations. As a notion of expression equivalence in a calculus, we employ *contextual equivalence* which identifies expressions iff they cannot be distinguished when observing convergence to WHNFs in any surrounding context. Contextual equivalence is coarser than the (syntactical) conversion equality, and provides a more useful language model due to its maximal set of equivalences.

However, complexity of a language makes analyses and reasoning hard, so it is advantageous to find conceptually simpler sublanguages which also permit reasoning about equivalences in the superlanguage. As a starting point we may use the pure core language, say L^{α}_{Hcore}, of Haskell [PS98], which is a Hindley-Milner polymorphically typed call-by-need lambda calculus extended by data constructors, case-expressions, `seq` for strict evaluation and `letrec` to model recursive bindings and sharing. The semantics of such extended lambda calculi have been analyzed in several papers [Ses97,MOW98,MS99,SSSS08,SSSM12].

However, even this language has a rich syntax and thus one may ask whether there are simpler and/or smaller languages which can be used to reason about (parts of) Haskell. The issue of transferring the equivalence question is as follows: given two expressions s_1, s_2 in a calculus L, in which cases is it possible to decide the semantic equivalence $s_1 \sim s_2$ by transferring the equivalence question for s_1, s_2 into a smaller or conceptually simpler language L_{simple}, using the proof methods in L_{simple}? There are three (standard) types of transfer steps: (i) from a typed language L^{τ} into its untyped language L (which may be larger). Since we use contextual equivalence, in general $s_1 \sim_L s_2$ implies $s_1 \sim_{L^{\tau}} s_2$ for equally typeable expressions

[3] This document is a revised version of an earlier version originally published on the web in April 2013.

s_1, s_2, and thus this is a valid transfer, however, some equivalences may be lost. (ii) from a language L into a sublanguage L_{sub} by the removal of a syntactic construction possibility. Since now all expressions of L_{sub} are also L-expressions, the desired implication $s_1 \sim_L s_2 \implies s_1 \sim_{L^\tau} s_2$ exactly corresponds to conservativity of the inclusion w.r.t. equivalence. (iii) transferring the question to an isomorphic language L'.

We consider four calculi in this paper: Abramsky's lazy lambda calculus AL and its extensions $\mathsf{AL}_{\mathsf{seq}}, \mathsf{AL}_{\mathsf{cc}}, \mathsf{AL}_{\mathsf{cc,seq}}$ with `seq`, with `case` and constructors, and the combination of the two extensions, resp. We also consider variants of these calculi with monomorphic (τ-superscript) and polymorphic (α-superscript) types. We analyze whether natural embeddings between the calculi are conservative w.r.t. contextual equivalence in the calculi.
Our results can be depicted as follows, where *Yes*/*No* indicates a conservative (non-conservative, resp.) embedding, and *Open* indicates that the question is still unresolved.

$$\begin{tikzcd} & \mathsf{AL}_{\mathsf{cc,seq}} & \\ \mathsf{AL}_{\mathsf{cc}} \arrow[ur, "No"] & & \mathsf{AL}_{\mathsf{seq}} \arrow[ul, "No"'] \\ & \mathsf{AL} \arrow[ul, "No"] \arrow[ur, "No"'] \arrow[uu, "No"] & \end{tikzcd} \quad \begin{tikzcd} & \mathsf{AL}^\tau_{\mathsf{cc,seq}} & \\ \mathsf{AL}^\tau_{\mathsf{cc}} \arrow[ur, "No"] & & \mathsf{AL}^\tau_{\mathsf{seq}} \arrow[ul, "Yes"'] \\ & \mathsf{AL}^\tau \arrow[ul, "Yes"] \arrow[ur, "No"'] \arrow[uu, "No"] & \end{tikzcd} \quad \begin{tikzcd} & \mathsf{AL}^\alpha_{\mathsf{cc,seq}} & \\ \mathsf{AL}^\alpha_{\mathsf{cc}} \arrow[ur, "No"] & & \mathsf{AL}^\alpha_{\mathsf{seq}} \arrow[ul, "Open"'] \\ & \mathsf{AL}^\alpha \arrow[ul, "Open"] \arrow[ur, "No"'] \arrow[uu, "No"] & \end{tikzcd}$$

A common pattern is that the removal of `seq` makes the embeddings non-conservative.
A powerful commonly used proof technique in all the calculi under consideration is based on Howe's method [How89,How96], which shows that contextual equivalence coincides with *applicative bisimilarity* which equates expressions if they cannot be distinguished by first evaluating them, then applying their results to arguments, and then using this experiment co-inductively. Our improvement, which is valid since the languages are deterministic, is a so-called *AP_i-context lemma*, which means that expressions are equivalent iff their termination behavior is identical when applying them in all possible ways to finitely many arbitrary arguments.

Our results are of help for equivalence reasoning in L^α_{Hcore} considering implication chains for the justification of equivalences. The first one starts with transferring to the untyped core-language L_{Hcore}, then removing the syntactic construct `letrec` (and changing call-by-need to call-by-name), justified in [SSSM10,SSSM12], arriving at $\mathsf{AL}_{\mathsf{cc,seq}}$. Then our results and counterexamples for the four untyped calculi come into play, where the conclusion is that further transfer steps appear impossible, in particular that AL [Abr90] cannot be justified as equivalence checking calculus via this implication chain. The second implication chain takes another potential route: the first step is monomorphising the core language, then removing the `letrec`, adding `Fix`, and again changing the reduction strategy to call-by-name, arriving at the calculus $\mathsf{AL}^\tau_{\mathsf{cc,seq}}$. We believe that both implications of equivalence are correct, but a formal proof is future work. Then, for the calculi $\mathsf{AL}^\tau_{\mathsf{cc}}$, $\mathsf{AL}^\tau_{\mathsf{seq}}$, AL^τ, we got negative as well as positive results. A further step could then be omitting the monomorphic types as well, which gives a valid implication chain from $\mathsf{AL}^\tau_{\mathsf{cc,seq}}$ to $\mathsf{AL}^\tau_{\mathsf{seq}}$ and to $\mathsf{AL}_{\mathsf{seq}}$, but again there is no justification for AL and AL^τ as equivalence checking calculi for L^α_{Hcore}. Thus our results show that calculus for the transfer is $\mathsf{AL}_{\mathsf{cc,seq}}$, and under the correctness assumptions above, also $\mathsf{AL}^\tau_{\mathsf{cc,seq}}$, $\mathsf{AL}^\tau_{\mathsf{seq}}$, and $\mathsf{AL}_{\mathsf{seq}}$. Focusing on the direct relation between the minimal calculi compared with L^α_{Hcore}, and taking into account our counterexamples in the paper, $\mathsf{AL}^\tau_{\mathsf{cc}}$ and $\mathsf{AL}_{\mathsf{cc}}$ are ruled out by examples s_7, s_8. However, it is still possible that AL or AL^τ can be used as equivalence checking calculi L^α_{Hcore} (although there are very few nontrivial equivalences there), which is strongly related to the open problem of whether there exist Böhm-like trees for AL (see Problem 18 in [TLC10]).

Related Work. Our approach follows the general setup laid out e.g. in [Fel91,SSNSS08] which consider the questions of relative expressivity between programming languages. , and define the notions such as conservativity of extensions, and adequacy and full-abstraction of translations from one language into another one. However, [Fel91] uses a notion of a conservative extension different from what we use throughout the paper: We require that all equivalences of the smaller language also hold in the extended language, while the notion of [Fel91] only requires that convergence of expressions of the smaller language coincides with convergence in the extended language. In difference to [Fel91], we use applicative bisimilarity and the AP_i-context lemma as a proof technique, and explore different calculi extensions. The closest work to ours is [RS94] that shows, in particular, that the extension of a monomorphically typed PCF with sum and product types and with Girard/Reynolds polymorphic types is conservative. They also show that extending PCF with a "convergence tester" by second-order polymorphic types is conservative. However, they do not (dis-)prove conservativity of adding the convergence tester to PCF and also do not consider an untyped case, or the pure lambda calculus. Moreover, the PCF has built-in integers with simple operations and a built-in test for 0. We do not know if these features themselves are a conservative extension of Abramsky's lazy lambda calculus.

Adding `seq` to call-by-need/call-by-name functional languages is investigated in several papers (e.g. [Fel91,HH10,JV06]). For the lazy lambda calculus and its extension by `seq` an example in [Fel91] can be adapted to show non-conservativity (see Theorem 4.3). It is well-known that in full Haskell `seq` makes a difference: The usual free theorems [Wad89] break under the addition of `seq` [JV06], and the monad laws do not hold for the `IO`-monad if the first argument of `seq` is allowed to be of an `IO`-type [SSS11]. [SSSM12,SSSS08] provide a counterexample showing non-conservativity of adding `seq` to the lazy lambda calculus with data-constructors and case expressions. However several questions remained open, e.g. whether the extension by `seq` in typed variants of the lazy lambda calculus is conservative.

Research on calculi extensions with case and constructors also including studies of untyped calculi is [AMR06,dV89,Stø06]. In [AMR06] the addition of case and constructors to a basic calculus is explored. However, that calculus significantly differs from our ones in several points, e.g. it permits full η-reduction. [dV89] and [Stø06] study an extension of a lambda calculus with surjective pairs. However, these works are incomparable to our approach since they use an axiomatic approach to equality instead of a rewriting and observational one.

Structure of the paper. In Sect. 2 we introduce a common notion for program calculi together with the notion of contextual equivalence. In Sect. 3 we briefly introduce the lazy lambda calculus AL and its three extensions $\mathsf{AL}_{\mathsf{cc}}, \mathsf{AL}_{\mathsf{seq}}, \mathsf{AL}_{\mathsf{cc,seq}}$. Conservativity of embeddings between the untyped calculi is refuted in Sect. 4. In Sect. 5 the monomorphically typed variants of the calculi are investigated. Counter-examples are provided for the addition of `seq`, and conservativity is proven for the extensions by case and constructors. Sect. 6 presents the analysis of polymorphically typed calculi. We conclude in Sect. 7. For readability some proofs are in a technical appendix.

2 Preliminaries

We define our notion of a program calculus in an abstract way:

Definition 2.1. *A* typed deterministic program calculus *(TDPC) is a tuple $(\mathcal{E}, \mathcal{C}, \rightarrow_D, \mathcal{A}, \mathcal{T})$ where $\mathcal{E}$ is the (nonempty) set of* expressions*, such that every $s \in \mathcal{E}$ has a type $T \in \mathcal{T}$. We write $\mathcal{E}_T$ for the expressions of type T, and assume $\mathcal{E}_T \neq \emptyset$. We also assume that $\mathcal{E}$ can be divided into* closed *and* open *expressions, where $\mathcal{E}^c$ denotes the set of closed expressions. We use s, t, r, a, b, d to denote expressions and x, y, z, u to denote variables. $\mathcal{C}$ is the set of* contexts*, such that every $C \in \mathcal{C}$ is a function $C : \mathcal{E}_T \rightarrow \mathcal{E}_{T'}$ where $T, T' \in \mathcal{T}$. With $\mathcal{C}_{T,T'}$ we denote the contexts that are functions from $\mathcal{E}_T$ to $\mathcal{E}_{T'}$. We assume that $\mathcal{C}$ contains the identity function for every type $T \in \mathcal{T}$, and that $\mathcal{C}$ is closed under composition, i.e. iff $C_1 \in \mathcal{C}_{T_2,T_3}$ and $C_2 \in \mathcal{C}_{T_1,T_2}$ then also $(C_1 \circ C_2) \in \mathcal{C}_{T_1,T_3}$. We denote the application of contexts C to an expression $s \in \mathcal{E}$ by $C[s]$. The* standard reduction relation *$\rightarrow_D \subseteq (\mathcal{E} \times \mathcal{E})$ must be: (i) deterministic: $s_1 \rightarrow_D s_2$ and $s_1 \rightarrow_D s_3$ implies $s_2 = s_3$, where $=$ is syntactical equivalence (which usually also identifies α-equivalent expressions); (ii) type preserving: $s_1 \rightarrow_D s_2$ implies that s_1 and s_2 are of the same type; (iii) closedness-preserving: if s_1 is closed and $s_1 \rightarrow_D s_2$, then s_2 is closed. The set $\mathcal{A} \subseteq \mathcal{E}$ are the* answers *of the calculus, which are usually irreducible values or specific kinds of normal forms. We use v to range over answers.*

An untyped calculus can also be presented as a typed one, by adding a single type called "expression". However, we simply write $(\mathcal{E}, \mathcal{C}, \rightarrow_D, \mathcal{A})$ for such a calculus.

We denote the transitive-reflexive closure of $\rightarrow_D$ by $\xrightarrow{*}_D$, and $\xrightarrow{n}_D$ with $n \in \mathbb{N}_0$ means n reductions. We define the notions of convergence, contextual approximation, and contextual equivalence in a general way. Expressions are contextually equal if they have the same termination behavior in any surrounding context. This makes contextual equivalence a strong equality, since the contexts of the language have a high discrimination power. For instance, it is not necessary to add additional tests, such as checking whether evaluation of both expressions terminates with the same values, since different values can be distinguished by contexts.

Definition 2.2. *Let $D = (\mathcal{E}, \mathcal{C}, \rightarrow, \mathcal{A}, \mathcal{T})$ be a TDPC. An expression $s \in \mathcal{E}$* converges *if there exists $v \in \mathcal{A}$ such that $s \xrightarrow{*}_D v$. We then write $s{\downarrow_D}v$, or just $s{\downarrow_D}$ if the value v is not of interest. If $s{\downarrow_D}$ does not hold, then we say s* diverges *and write $s{\Uparrow_D}$.* Contextual preorder *$\leq_D$ and* contextual equivalence *$\sim_D$ are defined by:*

$$\text{For } s_1, s_2 \in \mathcal{E}_T\text{: } s_1 \leq_D s_2 \text{ iff } \forall T' \in \mathcal{T}, C \in \mathcal{C}_{T,T'} : C[s_1]{\downarrow_D} \implies C[s_2]{\downarrow_D}$$
$$\text{For } s_1, s_2 \in \mathcal{E}_T\text{: } s_1 \sim_D s_2 \text{ iff } s_1 \leq_D s_2 \text{ and } s_2 \leq_D s_1$$

A program transformation *ξ is a binary relation on D-expressions, such that for all $s_1\ \xi\ s_2$ the expressions s_1 and s_2 are of the same type. ξ is* correct *if for all expressions $s_1\ \xi\ s_2$ the equivalence $s_1 \sim_D s_2$ holds.*

(β) $((\lambda x.s)\ t) \to s[t/x]$
(seq) $(\texttt{seq}\ v\ t)\ \to\ t$ if v is an answer
(case) $\texttt{case}_{K_i}\ (c_{K_i,j}\ \overrightarrow{s})\ (p_1\ \texttt{->}\ t_1)\ldots((c_{K_i,j}\ \overrightarrow{y})\ \texttt{->}\ t_j)\ldots(p_{|K_i|}\ \texttt{->}\ t_{|K_i|})) \to t_j[\overrightarrow{s}/\overrightarrow{y}]$
(fix) $(\texttt{Fix}\ s)\ \to\ s\ (\texttt{Fix}\ s)$

Fig. 1. Call-by-name reduction rules

By straightforward arguments one can prove that contextual preorder is a precongruence, and contextual equivalence is a congruence.

Definition 2.3. *Let $D = (\mathcal{E}, \mathcal{C}, \to_D, \mathcal{A}, \mathcal{T})$ and $D' = (\mathcal{E}', \mathcal{C}', \to_{D'}, \mathcal{A}', \mathcal{T}')$ be TDPCs. A* translation *$\zeta : D \to D'$ consists of mappings $\zeta : \mathcal{E} \to \mathcal{E}'$, $\zeta : \mathcal{C} \to \mathcal{C}'$, such that ζ maps the identity function $\mathcal{C}$ to the identity function in $\mathcal{C}'$, and $\zeta(s)$ is closed iff s is closed.*

- *ζ is* convergence equivalent (ce) *if $s{\downarrow_D} \iff \zeta(s){\downarrow_{D'}}$ for all $s \in \mathcal{E}$.*
- *ζ is* compositional up to observation (cuo)*, if for all $C \in \mathcal{C}$ and all $s \in \mathcal{E}$ such that $C[s]$ is typed: $\zeta(C[s]){\downarrow_{D'}}$ iff $\zeta(C)[\zeta(s)]{\downarrow_{D'}}$.*
- *ζ is* observationally correct (oc) *if it is (ce) and (cuo).*
- *ζ is* adequate *if for all expressions s, t: $\zeta(s) \leq_{D'} \zeta(t) \implies s \leq_D t$.*
- *ζ is* fully abstract *if for all expressions s, t: $\zeta(s) \leq_{D'} \zeta(t) \iff s \leq_D t$.*
- *ζ is an isomorphism if ζ is fully abstract and acts as a bijection on the equivalence classes from $\mathcal{E}/\sim_D$ to $\mathcal{E}'/\sim_{D'}$.*

We say D' is an extension *of D iff $\mathcal{T} \subseteq \mathcal{T}'$, $\mathcal{E}_T \subseteq \mathcal{E}'_T$ for any type $T \in \mathcal{T}$, $\mathcal{C}_{T,T'} \subseteq \mathcal{C}'_{T,T'}$ for all types $T, T' \in \mathcal{T}$, $\mathcal{A} = \mathcal{A}' \cap \mathcal{E}$ and $\to_D \subseteq \to_{D'}$ s.t. for all $e_1 \in \mathcal{E}$ with $e_1 \to_{D'} e_2$ always $e_2 \in \mathcal{E}$ (and thus $e_1 \to_D e_2$). Given D and an extension D', the* natural embedding *of D into D' is the identity translation of $\mathcal{E}_T$ into $\mathcal{E}'_T$ and $\mathcal{C}_{T,T'}$ into $\mathcal{C}'_{T,T'}$ for all types $T, T' \in \mathcal{T}$. A natural embedding is* conservative *iff it is a fully abstract translation.*

Note that a natural embedding is always convergence equivalent and compositional, which implies that it is always adequate (see [SSNSS08]).

3 Untyped Lazy Lambda Calculi and Their Properties

In this section we briefly introduce four variants of the lazy lambda calculus [Abr90] as instances of untyped TDPCs: the pure calculus AL, its extension by `seq`, called $\mathsf{AL}_{\mathsf{seq}}$, its extension by data constructors and `case`, called $\mathsf{AL}_{\mathsf{cc}}$, and finally its extension by `seq` as well as data constructors and `case`, called $\mathsf{AL}_{\mathsf{cc,seq}}$.

Definition 3.1 (Lazy Lambda Calculus AL). AL *is the (untyped)* lazy lambda calculus *[Abr90]. We define the components of* AL *according to Definition 2.1.*

*Expressions $\mathcal{E}$ are the set of expressions of the usual (untyped) lambda calculus, defined by the grammar $r, s, t \in \mathcal{L}_{\mathsf{AL}} ::= x \mid (s\ t) \mid \lambda x.s$. We identify α-equivalent expressions as syntactically equal according to Definition 2.1. The only reduction rule is β-*reduction *(see Fig. 1). An* AL-context *is defined as an expression in which one subexpression is replaced by the context hole $[\cdot]$.* AL-reduction contexts *R are defined by the grammar $R := [\cdot] \mid (R\ s)$, and the standard reduction in the sense of Definition 2.1 is the normal order reduction $\to_{\mathsf{AL}}$ which applies beta-reduction in a reduction context, i.e. $R[(\lambda x.s)\ t] \to_{\mathsf{AL}} R[s[t/x]]$. The answers $\mathcal{A}$ are all (also open) abstractions, which are also called* weak head normal forms (WHNF)*.*

Definition 3.2 ($\mathsf{AL}_{\mathsf{seq}}$). $\mathsf{AL}_{\mathsf{seq}}$ *is the lazy lambda calculus extended by* `seq`*, i.e. expressions are defined by $r, s, t \in \mathcal{L}_{\mathsf{AL}_{\mathsf{seq}}} ::= x \mid (s\ t) \mid \lambda x.s \mid \texttt{seq}\ s\ t$. Answers are all abstractions (WHNFs).* $\mathsf{AL}_{\mathsf{seq}}$*-reduction contexts R are defined by the grammar $R := [\cdot] \mid (R\ s) \mid \texttt{seq}\ R\ t$, and a normal order reduction is $R[s] \to_{\mathsf{AL}_{\mathsf{seq}}} R[t]$, whenever $s \xrightarrow{\beta} t$ or $s \xrightarrow{seq} t$ (see Fig. 1).*

Definition 3.3 ($\mathsf{AL}_{\mathsf{cc}}$). $\mathsf{AL}_{\mathsf{cc}}$ *extends* AL *by* `case` *and data constructors. There is a finite nonempty set of type constructors $K_1, \ldots, K_n$, where for every K_i there are pairwise disjoint finite nonempty sets of data constructors $\{c_{K_i,1}, \ldots c_{K_i,|K_i|}\}$. Every constructor has a fixed* arity *(a non-negative integer) denoted by $ar(K_i)$ or $ar(c_{K_i,j})$, resp. Examples are a type constructor Bool (of arity 0) with data constructors* `True` *and* `False` *(both of arity 0), as well as lists with a type constructor List (of arity 1) and data constructors* `Nil` *(of arity 0) and* `Cons` *(of arity 2). For the* constructor application *$(c_{K_i,j}\ s_1 \ldots s_{ar(c_{K_i,j})})$, we use $(c_{K_i,j}\ \overrightarrow{s})$ as an* *abbreviation, and write* $t[\overrightarrow{s}/\overrightarrow{x}]$ *for the parallel substitution* $t[s_1/x_1,\ldots,s_{ar(c_{K_i,j})}/x_{ar(c_{K_i,j})}]$. *The grammar* $r,s,t \in \mathcal{L}_{\mathsf{AL_{cc}}} ::= x \mid (s\ t) \mid \lambda x.s \mid (c_{K_i,j}\ \overrightarrow{s}) \mid (\mathtt{case}_{K_i}\ s\ (c_{K_i,1}\ \overrightarrow{x}\ \texttt{->}\ s_{i,1})\ldots(c_{K_i,|K_i|}\ \overrightarrow{x}\ \texttt{->}\ s_{i,|K_i|}))$ *defines expressions of* $\mathsf{AL_{cc}}$. *Note that constructor applications are allowed only to occur fully saturated. Note also that there are case expressions for every type constructor* K_i *and that in a case expression for type constructor* K_i *there must be exactly one case-alternative for every constructor belonging to type constructor* K_i. *We use an abbreviation* $\mathtt{case}_K$ *s alts if the alternatives of the* `case` *do not matter. The* $\mathsf{AL_{cc}}$*-reduction contexts* R *are defined as* $R := [\cdot] \mid (R\ s) \mid \mathtt{case}_{K_i}\ R\ alts$. *A normal order reduction is* $R[s] \to_{\mathsf{AL_{cc}}} R[t]$, *where* $s \xrightarrow{\beta} t$ *or* $s \xrightarrow{case} t$ *(see Fig. 1, where* p_i *mean patterns* $(c_{K_k,i}\ \overrightarrow{x})$ *in* `case`*-expressions). Answers in* $\mathsf{AL_{cc}}$ *are* $\lambda x.s$ *and* $(c_{K_i}\ \overrightarrow{s})$, *also called WHNFs.*

Note that in $\mathsf{AL_{cc}}$ there are closed stuck terms which are normal-order irreducible, but not WHNFs. Such expressions are "ill-typed", i.e. they are of the form $R[(c_{K_j,k}\overrightarrow{s})\ t]$, $R[\mathtt{case}_{K_i}\ \lambda x.s\ alts]$, or $R[\mathtt{case}_{K_i}\ c_{K_j,k}\overrightarrow{s}\ alts]$, where $K_i \neq K_j$.

Definition 3.4 ($\mathsf{AL_{cc,seq}}$)**.** *The calculus* $\mathsf{AL_{cc,seq}}$ *combines the syntax and reduction rules of* $\mathsf{AL_{seq}}$ *and* $\mathsf{AL_{cc}}$ *with the obvious notion of normal order reduction* $\to_{\mathsf{AL_{cc,seq}}}$ *applying (β), (seq), and (case) (see Fig. 1) in reduction contexts.*

We will write $\lambda x_1,x_2,\ldots,x_n.t$ instead of $\lambda x_1.\lambda x_2.\ldots\lambda x_n.t$. We use the following abbreviations for specific closed lambda expressions:

$$id = \lambda x.x \qquad \omega = \lambda x.(x\ x) \qquad \Omega = (\omega\ \omega)$$
$$Y = \lambda f.\big((\lambda x.f\ (x\ x))\ (\lambda x.f\ (x\ x))\big) \qquad \top = (Y\ (\lambda x,y.x))$$

It is not too hard to show that all closed diverging expressions are contextually equal. Thus we will use the symbol $\bot$ to denote a representative of the equivalence class of closed diverging expressions, e.g. one such expression is Ω.

Remark 3.5. Note that contextual equivalence in all our calculi always distinguishes different values. For instance, different constructors can always be distinguished by choosing `case`-expressions as contexts such that one constructor is mapped to a value while the other one is mapped to Ω. Different abstractions are distinguished by applying them to arguments. Different variables x,y are always contextually different: The context $C := (\lambda x,y.[\cdot])\ id\ \Omega$ distinguishes them, since $C[x]$ converges, while $C[y]$ diverges.

We now show correctness of program transformations. The *simplifications* for the calculi $\mathsf{AL_{seq}}, \mathsf{AL_{cc}}, \mathsf{AL_{cc,seq}}$ are defined in Fig. 2, s.t. each simplification is defined in all calculi where the constructs exist. In the appendix (Theorem B.1 and Lemma B.3) we show:

Theorem 3.6. *For* $D \in \{\mathsf{AL}, \mathsf{AL_{seq}}, \mathsf{AL_{cc}}, \mathsf{AL_{cc,seq}}\}$ *the reductions of the corresponding calculus (Fig. 1) and the simplifications (Fig. 2) are correct program transformations, regardless of the context they are applied in.*

Contextual equivalence of open expressions can be proven by closing them using additional lambda binders.

Lemma 3.7. *For* $D \in \{\mathsf{AL}, \mathsf{AL_{seq}}, \mathsf{AL_{cc}}, \mathsf{AL_{cc,seq}}\}$ *and* D*-expressions* s,t *with* $FV(s) \cup FV(t) \subseteq \{x_1,\ldots,x_n\}$: $s \sim_D t \iff \lambda x_1,\ldots,x_n.s \sim_D \lambda x_1,\ldots,x_n.t$.

Proof. The proof is given in the appendix, Lemma B.2).

Correctness of β-reduction implies that a restricted use of η-expansion is correct:

Proposition 3.8. *For every* $D \in \{\mathsf{AL}, \mathsf{AL_{seq}}, \mathsf{AL_{cc}}, \mathsf{AL_{cc,seq}}\}$ *the transformation* η *is correct for all abstractions, i.e.* $s \sim_D \lambda z.s\ z$, *if* s *is an abstraction.*

Definition 3.9. *We use* B_k^m *as an abbreviation for a "bot-alternative" of the* k^{th} *data constructor of type constructor* K_m *i.e.* $B_k^m := (c_{K_m,k}\ \overrightarrow{x}\ \texttt{->}\ \bot)$. *Let* v *be any closed abstraction (for* $\mathsf{AL}, \mathsf{AL_{seq}}$*) or be any closed abstraction or constructor application* $(c_{K_m,j}\ \overrightarrow{s})$ *(for in* $\mathsf{AL_{cc}}, \mathsf{AL_{cc,seq}}$*), respectively.*

Approximation contexts AP_i *(*$i \in \mathbb{N}_0$*) are defined for* $\mathsf{AL}, \mathsf{AL_{seq}}, \mathsf{AL_{cc}}, \mathsf{AL_{cc,seq}}$ *as follows:*

For AL, $\mathsf{AL_{seq}}$: $AP_0 ::= [\cdot] \qquad AP_{i+1} ::= (AP_i\ v) \mid (AP_i\ \bot)$

For $\mathsf{AL_{cc}}$, $\mathsf{AL_{cc,seq}}$: $AP_0 ::= [\cdot] \qquad AP_{i+1} ::= (AP_i\ v) \mid (AP_i\ \bot)$
$\mid \mathtt{case}_{K_m}\ AP_i\ B_1^m \ldots B_{j-1}^m\ (c_{K_m,j}\ \overrightarrow{x}\ \texttt{->}\ x_k)\ B_{j+1}^m \ldots B_n^m$
$\mid \mathtt{case}_{K_m}\ AP_i\ B_1^m \ldots B_{j-1}^m\ (c_{K_m,j}\ \overrightarrow{x}\ \texttt{->}\ \mathtt{True})\ B_{j+1}^m \ldots B_n^m$

(caseapp) $((\mathtt{case}_K\ t_0\ (p_1\ \texttt{->}\ t_1)\dots(p_n\ \texttt{->}\ t_n))\ r)$
$\rightarrow (\mathtt{case}_K\ t_0\ (p_1\ \texttt{->}\ (t_1\ r))\dots(p_n\ \texttt{->}\ (t_n\ r)))$
(casecase) $(\mathtt{case}_K\ (\mathtt{case}_{K'}\ t_0\ (p_1\ \texttt{->}\ t_1)\dots(p_n\ \texttt{->}\ t_n))\ (q_1\ \texttt{->}\ r_1)\dots(q_m\ \texttt{->}\ r_m))$
$\rightarrow (\mathtt{case}_{K'}\ t_0\ (p_1\ \texttt{->}\ (\mathtt{case}_K\ t_1\ (q_1\ \texttt{->}\ r_1)\dots(q_m\ \texttt{->}\ r_m)))$
$\dots$
$(p_n\ \texttt{->}\ (\mathtt{case}_K\ t_n (q_1\ \texttt{->}\ r_1)\dots(q_m\ \texttt{->}\ r_m))))$
(seqseq) $(\mathtt{seq}\ (\mathtt{seq}\ s_1\ s_2)\ s_3) \rightarrow (\mathtt{seq}\ s_1\ (\mathtt{seq}\ s_2\ s_3))$
(seqapp) $((\mathtt{seq}\ s_1\ s_2)\ s_3) \rightarrow (\mathtt{seq}\ s_1\ (s_2\ s_3))$
(seqcase) $(\mathtt{seq}\ (\mathtt{case}_K\ t_0\ (p_1\ \texttt{->}\ t_1)\dots(p_n\ \texttt{->}\ t_n))\ r)$
$\rightarrow (\mathtt{case}_K\ t_0\ (p_1\ \texttt{->}\ (\mathtt{seq}\ t_1\ r))\dots(p_n\ \texttt{->}\ (\mathtt{seq}\ t_n\ r)))$
(caseseq) $(\mathtt{case}_K\ (\mathtt{seq}\ s_1\ s_2)\ \mathit{alts}) \rightarrow (\mathtt{seq}\ s_1\ (\mathtt{case}_K\ s_2\ \mathit{alts}))$

Fig. 2. case- and seq-simplifications

$s_1 := \lambda x.x\ (\lambda y.x\ \top\ \bot\ y)\ \top$ $s_2 := \lambda x.x\ (x\ \top\ \bot)\ \top$
$t_1(s) := \lambda x.x\ (x\ s)$ $t_2(s) := \lambda x.x\ \lambda z.x\ s\ z$
where s is an expression with $FV(s) \subseteq \{x\}$
$s_3 := \lambda x, y.x\ (y\ (y\ (x\ id)))$ $s_4 := \lambda x, y.x\ (y\ \lambda z.y\ (x\ id)\ z)$
$s_5 := \lambda x, y.(x\ (x\ y))\ (x\ (x\ y))$ $s_6 := \lambda x, y.(x\ (x\ y))\ (x\ \lambda z.x\ y\ z))$
$s_7 := \lambda x.\mathtt{case}_{Bool}\ (x\ \bot)\ (\mathtt{True}\ \texttt{->}\ \mathtt{True})\ (\mathtt{False}\ \texttt{->}\ \bot)$
$s_8 := \lambda x.\mathtt{case}_{Bool}\ (x\ \lambda y.\bot)\ (\mathtt{True}\ \texttt{->}\ \mathtt{True})\ (\mathtt{False}\ \texttt{->}\ \bot)$

Fig. 3. The untyped counterexample expressions

The following result known as a context lemma is proven in the appendix (Theorem B.9).

Theorem 3.10 (**AP_i-Context-Lemma**)**.** *For $D \in \{\mathsf{AL}, \mathsf{AL}_{\mathsf{seq}}, \mathsf{AL}_{\mathsf{cc}}, \mathsf{AL}_{\mathsf{cc,seq}}\}$ and closed D-expressions s, t holds:*
$s \leq_D t$ iff for all i and all approximation contexts AP_i: $AP_i[s]{\downarrow_D} \implies AP_i[t]{\downarrow_D}$

We provide a criterion to prove contextual equivalence of expressions, which is used in later sections. Its proof can be found in the appendix (Theorem B.11).

Theorem 3.11. *For $D \in \{\mathsf{AL}, \mathsf{AL}_{\mathsf{seq}}, \mathsf{AL}_{\mathsf{cc}}, \mathsf{AL}_{\mathsf{cc,seq}}\}$ closed D-expressions s and t are contextually equivalent if there exists $i \in \mathbb{N}_0$ such that*

1. *$AP_j[s]{\downarrow_D} \iff AP_j[t]{\downarrow_D}$ for all $0 \leq j < i$ and all AP_j-contexts.*
2. *$AP_i[s] \sim_D AP_i[t]$ for all AP_i-contexts.*

For all four calculi *applicative contexts* are defined by $A ::= [\cdot] \mid (A\ s)$. The following proposition allows systematic case-distinctions for expressions (the proof can be found in the appendix, Proposition B.5).

Proposition 3.12. *Let $D \in \{\mathsf{AL}, \mathsf{AL}_{\mathsf{seq}}, \mathsf{AL}_{\mathsf{cc}}, \mathsf{AL}_{\mathsf{cc,seq}}\}$. For every D-expression s one of the following equations holds: 1. $s \sim_D \bot$; 2. $s \sim_D v$ where v is an answer; 3. $s \sim_D A[x]$ where x is a free variable and A is an applicative D-context; 4. $s \sim_D \mathtt{seq}\ A[x]\ t'$ where x is a free variable and A is an applicative D-context, for $D \in \{\mathsf{AL}_{\mathsf{seq}}, \mathsf{AL}_{\mathsf{cc,seq}}\}$; or 5. $s \sim_D \mathtt{case}_K\ A[x]\ alts$ where x is a free variable and A is an applicative D-context, for $D \in \{\mathsf{AL}_{\mathsf{cc}}, \mathsf{AL}_{\mathsf{cc,seq}}\}$.*

4 Relations between the Untyped Calculi

This section shows the non-conservativity of embeddings of the four untyped lazy calculi. These negative results show that the syntactically less expressive calculi are not sufficiently expressive and are thus unstable under extensions. Expressions used in our counterexamples are defined in Fig. 3. We also prove equations necessary for the examples:

Lemma 4.1. *For all expressions s: $\top \sim_{\mathsf{AL}} \lambda x.\top$ and $\top\ s \sim_{\mathsf{AL}} \top$.*

Proof. $\top\ s \sim_{\mathsf{AL}} \top$ follows from correctness of β (Theorem 3.6), since $\top\ s \xrightarrow{\beta,*} \lambda x.\lambda z.(\lambda x.\lambda z.(x\ x))\ (\lambda x.\lambda z.(x\ x)) \xleftarrow{\beta,*} \top$. For $\top \sim_{\mathsf{AL}} \lambda x.\top$ we use Theorem 3.11 (for $i = 1$): $\top{\downarrow_D}, \lambda x.\top{\downarrow_D}$, and $(\lambda x.\top)\ r \xrightarrow{\beta} \top \sim_D (\top\ r)$ for any r.

Theorem 4.2. *The following equalities hold for the expressions in Fig. 3: 1. If $s[id/x] \not\sim_{\mathsf{AL}} \bot$ then $t_1(s) \sim_{\mathsf{AL}} t_2(s)$. 2. $s_1 \sim_{\mathsf{AL}} s_2$. 3. $s_3 \sim_{\mathsf{AL}} s_4$. 4. $s_5 \sim_{\mathsf{AL_{seq}}} s_6$. 5. $s_7 \sim_{\mathsf{AL_{cc}}} s_8$.*

Proof. 1. *We use Theorem 3.11 (for $i = 1$). For the empty context we have $t_1(s)\downarrow_{\mathsf{AL}}$ and $t_2(s)\downarrow_{\mathsf{AL}}$. Now we consider the case $(t_1\ b)$ and $(t_2\ b)$ where b is a closed abstraction or $\bot$. We make a case distinction on the argument b according to Proposition 3.12. By easy computations $(t_1\ b) \sim_{\mathsf{AL}} (t_2\ b)$ if $b = \bot$, $b = \lambda x.\bot$, or $b = \lambda x_1.\lambda x_2.t$. For $b := \lambda x.x$, two β-reductions show that $t_1\ \lambda x.x \sim_{\mathsf{AL}} s[id/x]$, and that $t_2\ \lambda x.x \sim_{\mathsf{AL}} \lambda z.s[id/x]\ z$. Since $s[id/x] \not\sim_{\mathsf{AL}} \bot$, it is equivalent to an abstraction, and Proposition 3.8 shows contextual equivalence of the two expressions. Now let $b := \lambda u.u\ t_1\ \dots\ t_n$ with $n \geq 1$. If $(b\ s[b/x]) \not\sim_{\mathsf{AL}} \bot$, then there exists a closed abstraction $\lambda w.s'$ such that $(\lambda w.s') \sim_{\mathsf{AL}} (b\ s[b/x])$. By Proposition 3.8 we can transform: $(t_1\ b) \sim_{\mathsf{AL}} b\ (b\ s[b/x]) \sim_{\mathsf{AL}} b\ \lambda w.s' \sim_{\mathsf{AL}}\ b\ \lambda z.(\lambda w.s')\ z \sim_{\mathsf{AL}} b\ \lambda z.(b\ s[b/x]\ z) \sim_{\mathsf{AL}} t_2\ b$. In the case $(b\ s[b/x]) \sim_{\mathsf{AL}} \bot$, evaluation of $(\lambda u.u\ t_1 \dots t_n)\ \bot$ and $(\lambda u.u\ t_1 \dots t_n)\ (\lambda y.\bot)$ results in $\bot$.*

2. *We use Theorem 3.11 (for $i = 1$). Since $s_1\downarrow_{\mathsf{AL}}$ and $s_2\downarrow_{\mathsf{AL}}$, we only consider the cases $(s_1\ b)$ and $(s_2\ b)$ where b is a closed abstraction or $\bot$. We use Proposition 3.12 for a case distinction on b. It is easy to verify that $s_1\ b \sim_{\mathsf{AL}} s_2\ b$ for $b = \bot$, $b = \lambda z.\bot$, and $b = \lambda z.z$. For $b := \lambda z.(z\ u_1 \dots u_n)$ where $n \geq 1$, we have $(s_1\ b) \sim_{\mathsf{AL}} b\ (\lambda y.\top)\ \top$ and $(s_2\ b) \sim_{\mathsf{AL}} b\ \top\ \top$, and by Lemma 4.1 also $(s_1\ b) \sim_{\mathsf{AL}} (s_2\ b)$.*
3. *We use Theorem 3.11 (for $i = 2$). Since $s_j\downarrow_{\mathsf{AL}}$ and $(s_j\ b)\downarrow_{\mathsf{AL}}$ for $j = 3, 4$ we need to consider the cases $(s_3\ b\ d)$ and $(s_4\ b\ d)$ where b, d are closed abstractions or $\bot$. We use Proposition 3.12 for case distinction on d. If $d = \bot$, or $d = \lambda x.\bot$, then $s_3\ b\ d \sim_{\mathsf{AL}} s_4\ b\ d$. If $d := \lambda x.x$, then item 1 shows that $\lambda x.x\ (x\ id) \sim_{\mathsf{AL}} \lambda x.x\ \lambda z.(x\ id)\ z$. Correctness of β implies that $b\ (b\ id) \sim_{\mathsf{AL}} b\ \lambda z.(b\ id)\ z$, and thus $s_3\ b\ d \sim_{\mathsf{AL}} b\ (b\ id) \sim_{\mathsf{AL}} b\ \lambda z.(b\ id)\ z \sim_{\mathsf{AL}} s_4\ b\ d$. If $d := \lambda x_1.\lambda x_2.t$, then $s_3\ b\ d \sim_{\mathsf{AL}} b\ (d\ (\lambda x_2.t[(b\ id)/x_1])) \xrightarrow{\eta} b\ (d\ \lambda.z\ (\lambda x_2.t[(b\ id)/x_1])\ z) \sim_{\mathsf{AL}} s_4\ b\ d$, where η is correct by Proposition 3.8. If $d := \lambda u.u\ t_1\ \dots\ t_n$ with $n \geq 1$ and $(d\ (b\ id)) \not\sim_{\mathsf{AL}} \bot$, it is equivalent to an abstraction, and η is correct, hence equivalence holds in this case. Otherwise, if $(d\ (b\ id)) \sim_{\mathsf{AL}} \bot$, then $(b\ (d\ \bot) \sim_{\mathsf{AL}} (b\ (d\ \lambda x.\bot))$ since $(d\ \bot) \sim_{\mathsf{AL}} \bot \sim_{\mathsf{AL}} (d\ \lambda x.\bot)$.*
4. *We use Theorem 3.11 (for $i = 2$). We have $s_j\downarrow_{\mathsf{AL_{seq}}}$ and $(s_j\ b)\downarrow_{\mathsf{AL_{seq}}}$ for any b for $j = 5, 6$. Now we consider the cases $(s_5\ b\ d)$ and $(s_6\ b\ d)$ where b, d are closed abstractions or $\bot$. We make a case distinction on b using Proposition 3.12. The cases $b = \bot$, $b = \lambda x.\bot$, and $b = \lambda u.u$ are easy to verify. If $b = \lambda u, v.b'$ then the subexpression $(b\ d)$ is contextually equivalent to $\lambda v.b'[d/u]$ Thus, η-expansion for $(b\ d)$ is correct which shows $s_5\ b\ d\ \sim_{\mathsf{AL_{seq}}} s_6\ b\ d$. For the other case we distinguish whether $(b\ d) \sim_{\mathsf{AL_{seq}}} \bot$ holds. If $(b\ d) \not\sim_{\mathsf{AL_{seq}}} \bot$ then η is correct, which shows that $s_5\ b\ d\ \sim_{\mathsf{AL_{seq}}} s_6\ b\ d$. If $(b\ d) \sim_{\mathsf{AL_{seq}}} \bot$, then we have to check more cases: If $b = \lambda u.\mathtt{seq}\ (u\ t_1 \dots)\ r$ or $b = \lambda u.\mathtt{seq}\ u\ r$, then $(b\ (b\ d)) \sim_{\mathsf{AL_{seq}}} \bot$, and $s_5\ b\ d$ is equivalent to $s_6\ b\ d$. If $b = \lambda u.u\ t_1 \dots$, then $(b\ (b\ d))$ becomes $\bot$ in both expressions, which shows $(s_5\ b\ d) \sim_{\mathsf{AL_{seq}}} \bot \sim_{\mathsf{AL_{seq}}} (s_6\ b\ d)$.*
5. *We use Theorem 3.11. Since $s_7\downarrow_{\mathsf{AL_{cc}}}$, $s_8\downarrow_{\mathsf{AL_{cc}}}$, and $\mathtt{case}\ s_7\ \dots \sim_{\mathsf{AL_{cc}}} \bot \sim_{\mathsf{AL_{cc}}}\ \mathtt{case}\ s_8\ \dots$, it is sufficient to show $(s_7\ b)\ \sim_{\mathsf{AL_{cc}}} (s_8\ b)$ where b is a closed abstraction, a constructor application, or $\bot$. If $b = \bot$ then the equivalence holds. Otherwise, we inspect the cases of a normal-order reduction for $b\ y$ for some free variable y: If $b\ y\downarrow_{\mathsf{AL_{cc}}}\mathtt{True}$ (or $b\ y\downarrow_{\mathsf{AL_{cc}}}\mathtt{False}$, resp.) then $(b\ \bot)$ and $(b\ \lambda x.\bot)$ also converge with $\mathtt{True}$ (or $\mathtt{False}$, resp.), which shows that $(s_7\ b) \sim_{\mathsf{AL_{cc}}} (s_8\ b)$. If evaluation of $b\ y$ stops with $R[y]$ for some reduction context R, then $(s_7\ b)$ evaluates to $\mathtt{case}_{Bool}\ R[\bot]$ alts which is equivalent to $\bot$, and $(s_8\ b)$ evaluates to $\mathtt{case}_{Bool}\ R[\lambda x.\bot]$ alts. We consider cases of R: If $R = [\cdot]$ then $(s_8\ b) \sim_{\mathsf{AL_{cc}}} \bot$. If $R = R'[[\cdot]\ r]$ then $(s_8\ b)$ evaluates to $\mathtt{case}_{Bool}\ R'[\bot]$ alts which is equivalent to $\bot$. Finally, if $R = R'[\mathtt{case}_K\ [\cdot]\ alts']$ then $(s_8\ b) \sim_{\mathsf{AL_{cc}}} \bot$. If $(b\ y)$ converges with $c_{K_i,j}\ t_1\ \dots\ t_n$ for some constructor $c_{K_i,k}$ not of type Bool then $(b\ \bot)$ converges to $c_{K_i,j}\ t'_1\ \dots\ t'_n$ and $(b\ \lambda x.\bot)$ converges to $c_{K_i,j}\ t''_1\ \dots\ t''_n$. However, in this case $(s_7\ b)\ \sim_{\mathsf{AL_{cc}}} \bot \sim_{\mathsf{AL_{cc}}} (s_8\ b)$.* □

Now we obtain non-conservativity for all embeddings between the four calculi as follows:

Theorem 4.3. *The natural embeddings of AL in $\mathsf{AL_{seq}}$, AL in $\mathsf{AL_{cc,seq}}$, AL in $\mathsf{AL_{cc}}$, $\mathsf{AL_{seq}}$ in $\mathsf{AL_{cc,seq}}$, and $\mathsf{AL_{cc}}$ in $\mathsf{AL_{cc,seq}}$ are not conservative.*

Proof. **AL in $\mathsf{AL_{seq}}$ and AL in $\mathsf{AL_{cc,seq}}$:** The proof uses the expressions s_1, s_2 which are adapted from the example of [Fel91, Proposition 3.15]. Theorem 4.2, item 2 shows that $s_1 \sim_{\mathsf{AL}} s_2$. The context $C := ([\cdot]\ \lambda z.\mathtt{seq}\ z\ id)$ distinguishes s_1, s_2 in $\mathsf{AL_{seq}}, \mathsf{AL_{cc,seq}}$, since $C[s_1]\downarrow_D$ while $C[s_2]\Uparrow_D$ for $D \in \{\mathsf{AL_{seq}}, \mathsf{AL_{cc,seq}}\}$.

Another counterexample uses the expressions $t_1(s), t_2(s)$ with $s = (x\ ((x\ id)\ (x\ id)))$: Since $s[id/x] \sim_{\mathsf{AL}} id$, Theorem 4.2, item 1 shows $t_1(s) \sim_{\mathsf{AL}} t_2(s)$. However, the context $C := ([\cdot]\ \lambda y.\ \mathtt{seq}\ y\ \omega)$ distinguishes $t_1(s)$ and $t_2(s)$ in $\mathsf{AL_{seq}}$ and $\mathsf{AL_{cc,seq}}$.

AL in $\mathsf{AL_{cc}}$: From Theorem 4.2 we have $s_3 \sim_{\mathsf{AL}} s_4$. In $\mathsf{AL_{cc}}$ the context $C := [\cdot]\ (\lambda u.u\ \mathtt{True})\ (\lambda u.\mathtt{case}_{Bool}\ u\ (\mathtt{True} \mathbin{\text{->}} \mathtt{False})\ (\mathtt{False} \mathbin{\text{->}} id))$ distinguishes s_3 and s_4, since $C[s_3] \sim_{\mathsf{AL_{cc}}} \mathtt{True}$ and $C[s_4]\Uparrow_{\mathsf{AL_{cc}}}$.

$\mathsf{AL}_{\mathsf{seq}}$ in $\mathsf{AL}_{\mathsf{cc,seq}}$: Theorem 4.2 shows $s_5 \sim_{\mathsf{AL}_{\mathsf{seq}}} s_6$. In $\mathsf{AL}_{\mathsf{cc,seq}}$ the context $C := ([\cdot]\ b\ \mathtt{True})$ with $b := \lambda u.\mathtt{case}_{Bool}\ u\ (\mathtt{True} \mathtt{->} \mathtt{False})(\mathtt{False} \mathtt{->} id)$ distinguishes s_5 and s_6, since $C[s_5] \xrightarrow{*}_{\mathsf{AL}_{\mathsf{cc,seq}}} id$, but $C[s_6]\Uparrow_{\mathsf{AL}_{\mathsf{cc,seq}}}$.

$\mathsf{AL}_{\mathsf{cc}}$ in $\mathsf{AL}_{\mathsf{cc,seq}}$: A counterexample for conservativity of embedding $\mathsf{AL}_{\mathsf{cc}}$ into $\mathsf{AL}_{\mathsf{cc,seq}}$ was given in [SSSM12] which can be translated into the notations of this paper as follows: The equation $s_7 \sim_{\mathsf{AL}_{\mathsf{cc}}} s_8$ holds (Theorem 4.2), but for the context $C := [\cdot]\ \lambda u.\mathtt{seq}\ u\ \mathtt{True}$ we have $C[s_8]\downarrow_{\mathsf{AL}_{\mathsf{cc,seq}}}\mathtt{True}$ while $C[s_7]\Uparrow_{\mathsf{AL}_{\mathsf{cc,seq}}}$. $\square$

5 Monomorphically Typed Calculi and Embeddings

We now analyze embeddings among the four calculi under monomorphic typing, and therefore we add a monomorphic type system to the calculi. The counterexamples in Sect. 4 cannot be transferred to the typed calculi except for the counterexample showing non-conservativity of embedding $\mathsf{AL}_{\mathsf{cc}}$ into $\mathsf{AL}_{\mathsf{cc,seq}}$.

Since AL with a monomorphic type system is the simply typed lambda calculus (which is too inexpressive since every expression converges) we extend all the calculi by a fixpoint combinator `Fix` as a constant to implement recursion, and by a constant `Bot` to denote a diverging expression[4]. The resulting calculi are called AL^τ, $\mathsf{AL}^\tau_{\mathsf{seq}}$, $\mathsf{AL}^\tau_{\mathsf{cc}}$, $\mathsf{AL}^\tau_{\mathsf{cc,seq}}$.

The syntax for types is $T ::= o \mid T \to T \mid K(T_1,\ldots,T_{arK})$, where o is the base type, and K is a type constructor. The syntax for expressions is as in the base calculi, but extended by `Fix` as a family of constants of all types of the form $(T \to T) \to T$, and the constant `Bot` as a family of constants of all types. Variables have a built-in type, i.e. in an expression every variable is annotated with a monomorphic type, e.g. $\lambda x^{o\to o}.x^{o\to o}$ is an identity on functions of type $o \to o$. However, we rarely write these annotations explicitly. The type of constructors is structured as in a polymorphic calculus: The family of constructors for one constructor c_{K_i} has a (polymorphic) type schema of the form $T_1 \to \ldots \to T_n \to (K_i\ T'_1\ \ldots\ T'_{ar(K_i)})$, where every type-variable of $T_1 \to \ldots \to T_n$ is contained in $(K_i\ T'_1\ \ldots\ T'_{ar(K_i)})$, and every monomorphic type of constructor c_{K_i} is an instance of this type. The types of `case` and `seq` are the monomorphic instances of the usual polymorphic types as in Haskell. We omit the standard typing rules. However, we write $s :: T$ which means that s can be typed by a (monomorphic) type T. The reduction rules are in Fig. 1, and normal order reduction $\to_D$ for $D \in \{\mathsf{AL}^\tau, \mathsf{AL}^\tau_{\mathsf{seq}}, \mathsf{AL}^\tau_{\mathsf{cc}}, \mathsf{AL}^\tau_{\mathsf{cc,seq}}\}$ applies the reduction rules in reduction contexts (defined as before). It is easy to verify that normal order reduction is deterministic, type-, and closedness-preserving. The following progress lemma holds: for every closed expression t, either $t \xrightarrow{*}_D t_0$, where t_0 is a value, or t has an infinite reduction sequence, or $t \xrightarrow{*}_D R[\mathtt{Bot}]$, where R is a reduction context. In particular, the typing implies that case-expressions $(\mathtt{case}_K\ (c\ \ldots)\ alts)$ are always reducible by a case-reduction.

Answers are defined as abstractions, constructor applications, and the constant `Fix`. Contextual equivalence $\sim_D$ is defined according to Definition 2.2.

We also reuse the approximation contexts, but restrict them to well-typed contexts. The AP_i-context lemma (Theorem 3.10) also holds for the typed calculi, where only equally typed expressions and well-typed contexts are taken into account.

Theorem 5.1. *For $D \in \{\mathsf{AL}^\tau, \mathsf{AL}^\tau_{\mathsf{seq}}, \mathsf{AL}^\tau_{\mathsf{cc}}, \mathsf{AL}^\tau_{\mathsf{cc,seq}}\}$ and closed, equally typed D-expressions s,t holds: $s \leq_D t$ iff for all i and all approximation contexts AP_i, such that $AP_i[s]$ and $AP_i[t]$ are well-typed: $AP_i[s]\downarrow_D \implies AP_i[t]\downarrow_D$.*

To lift the correctness results for program transformations into the typed calculi, we define a translation δ.

Definition 5.2. *Let $\delta : \mathsf{AL}^\tau_{\mathsf{cc,seq}} \to \mathsf{AL}_{\mathsf{cc,seq}}$ be the translation of an $\mathsf{AL}^\tau_{\mathsf{cc,seq}}$-expression that first removes all types and then leaves all syntactical constructs as they are except for the cases $\delta(\mathtt{Bot}) := \Omega$ and $\delta(\mathtt{Fix}) := \lambda f.(\lambda x.f\ (x\ x))\ (\lambda x.f\ (x\ x))$.*

In the appendix (Corollary C.5) we prove adequacy of δ, which implies that reduction rules and simplifications are correct program transformations in the typed calculi (Lemma C.10).

Proposition 5.3. *For equally typed $\mathsf{AL}^\tau_{\mathsf{cc,seq}}$-expressions s,t it holds: $\delta(s) \sim_{\mathsf{AL}_{\mathsf{cc,seq}}} \delta(t)$ implies $s \sim_{\mathsf{AL}^\tau_{\mathsf{cc,seq}}} t$. The same holds for $\mathsf{AL}^\tau, \mathsf{AL}^\tau_{\mathsf{seq}}$, and $\mathsf{AL}^\tau_{\mathsf{cc}}$ w.r.t. their untyped variants.*

Theorem 5.4. *All reduction rules and simplifications in Figs. 1, 2, and 4 are correct program transformations in $\mathsf{AL}^\tau, \mathsf{AL}^\tau_{\mathsf{cc}}, \mathsf{AL}^\tau_{\mathsf{seq}}$, and $\mathsf{AL}^\tau_{\mathsf{cc,seq}}$.*

[4] `Bot` can also be encoded using `Fix`, but for convenient representation we include the constant.

(botapp) $(\texttt{Bot}\ s) \to \texttt{Bot}$ (botseq) $(\texttt{seq Bot}\ s) \to \texttt{Bot}$
(botcase) $(\texttt{case}_K\ \texttt{Bot}\ alts) \to \texttt{Bot}$

Fig. 4. The bot-simplifications

We now show non-conservativity of embedding AL^τ in $\mathsf{AL}^\tau_{\mathsf{seq}}$ as well as of $\mathsf{AL}^\tau_{\mathsf{cc}}$ in $\mathsf{AL}^\tau_{\mathsf{cc,seq}}$, i.e. the addition of `seq` is not conservative. For the other embeddings, AL^τ in $\mathsf{AL}^\tau_{\mathsf{cc}}$ and $\mathsf{AL}^\tau_{\mathsf{seq}}$ in $\mathsf{AL}^\tau_{\mathsf{cc,seq}}$, we show conservativity. This is consistent with typability: the counterexample for AL in $\mathsf{AL}_{\mathsf{cc}}$ requires an untyped context, and the counterexample for $\mathsf{AL}_{\mathsf{seq}}$ in $\mathsf{AL}_{\mathsf{cc,seq}}$ has a self-application of an expression, which is nontypable.

5.1 Adding Seq is Not Conservative

We consider calculi AL^τ and $\mathsf{AL}^\tau_{\mathsf{seq}}$.

There is only one equivalence class w.r.t. contextual equivalence for closed expressions of type o: it is $\texttt{Bot}^o :: o$. For type $o \to o$, there are only two equivalence classes with representatives $\texttt{Bot}^{o\to o}$ and $\lambda x^o.\texttt{Bot}^o$. Note that the expression $\lambda x^o.x^o$ is equivalent to $\lambda x^o.\texttt{Bot}^o$, since there are no values of type o.

Our counterexample to conservativity are the following expressions s_9, s_{10} of type $((o \to o) \to (o \to o) \to (o \to o)) \to (o \to o) \to (o \to o) \to (o \to o) \to (o \to o)$:

$$s_9 := \lambda f, x, y, z.f\ (f\ x\ y)\ (f\ y\ z) \qquad s_{10} := \lambda f, x, y, z.f\ (f\ x\ x)\ (f\ z\ z)$$

Theorem 5.5. *The embedding of* AL^τ *into* $\mathsf{AL}^\tau_{\mathsf{seq}}$ *and into* $\mathsf{AL}^\tau_{\mathsf{cc,seq}}$ *is not conservative*

Proof. We use Theorem 3.11 (with $i = 1$) which also holds in the typed calculi (Theorem C.3) and show $s_9 \sim_{\mathsf{AL}^\tau} s_{10}$. Since $s_9{\downarrow}_{\mathsf{AL}^\tau}$ and $s_{10}{\downarrow}_{\mathsf{AL}^\tau}$, we need to show $s'_9 := (s_9\ b) \sim_{\mathsf{AL}^\tau} s'_{10} := (s_{10}\ b)$, where b is a closed expression of type $(o \to o) \to (o \to o) \to (o \to o)$. We check the different cases for b. Due to its type b must be equivalent to one of $\texttt{Bot}$, $\lambda w.\texttt{Bot}$, $\lambda u, w.\texttt{Bot}$, $\lambda w_1, w_2, w_3.\texttt{Bot}$, $\lambda x, y.x$, and $\lambda x, y.y$. For the first three cases it holds: $s'_9 \sim_{\mathsf{AL}^\tau} \lambda x, y, z.\texttt{Bot} \sim_{\mathsf{AL}^\tau} s'_{10}$. If $b = \lambda w_1, w_2, w_3.\texttt{Bot}$ then $s'_9 \sim_{\mathsf{AL}^\tau} \lambda x, y, z, w_3.\texttt{Bot} \sim_{\mathsf{AL}^\tau} s'_{10}$. If $b = \lambda x, y.x$ then $s'_9 \sim_{\mathsf{AL}^\tau} \lambda x, y, z.x \sim_{\mathsf{AL}^\tau} s'_{10}$. If $b = \lambda x, y.y$ then $s'_9 \sim_{\mathsf{AL}^\tau} \lambda x, y, z.z \sim_{\mathsf{AL}^\tau} s'_{10}$. Nonconservativity now follows from the context $C = ([\cdot]\ (\lambda x, y.\texttt{seq}\ x\ y)\ (\lambda x.\texttt{Bot})\ \texttt{Bot}\ (\lambda x.\texttt{Bot}))$: The expressions $C[s_9]$, $C[s_{10}]$, are typable in $\mathsf{AL}^\tau_{\mathsf{seq}}, \mathsf{AL}^\tau_{\mathsf{cc,seq}}$ and $C[s_9] \sim_D \texttt{Bot}$, but $C[s_{10}] \sim_D (\lambda x.\texttt{Bot})$ for $D \in \{\mathsf{AL}^\tau_{\mathsf{seq}}, \mathsf{AL}^\tau_{\mathsf{cc,seq}}\}$.

We reuse the counterexample in the untyped case represented by expressions s_7 and s_8, where $\perp$ is replaced by `Bot`. The example becomes

$$s_{11} := \lambda x.\texttt{case}_{Bool}\ (x\ \texttt{Bot})\ (\texttt{True -> True})\ (\texttt{False -> Bot})$$
$$s_{12} := \lambda x.\texttt{case}_{Bool}\ (x\ (\lambda y.\texttt{Bot}))\ (\texttt{True -> True})\ (\texttt{False -> Bot})$$

where s_{11}, s_{12} are typed as $((T \to Bool) \to Bool)$ for any type T. The two expressions are equivalent in $\mathsf{AL}^\tau_{\mathsf{cc}}$: They are typed, and $\delta(s_{11}) \sim_{\mathsf{AL}_{\mathsf{cc}}} \delta(s_{12})$ (see Theorem 4.2, item 5). Thus Proposition 5.3 is applicable. However, $s_{11} \not\sim_{\mathsf{AL}^\tau_{\mathsf{cc,seq}}} s_{12}$, since $s_{12}\ b$ evaluates to `True`, while $s_{11}\ b$ diverges, where $b = \lambda u.\texttt{seq}\ u\ \texttt{True}$.

Theorem 5.6. *The embedding of* $\mathsf{AL}^\tau_{\mathsf{cc}}$ *into* $\mathsf{AL}^\tau_{\mathsf{cc,seq}}$ *is not conservative.*

5.2 Adding Case and Constructors is Conservative

We show that adding case and constructors to the monomorphically typed calculi is conservative. We give a detailed proof for embedding $\mathsf{AL}^\tau_{\mathsf{seq}}$ into $\mathsf{AL}^\tau_{\mathsf{cc,seq}}$. The proof for embedding AL^τ into $\mathsf{AL}^\tau_{\mathsf{cc}}$ is analogous by omitting unnecessary cases. We show that for $\mathsf{AL}^\tau_{\mathsf{seq}}$-expressions s, t the embedding is fully abstract, i.e. $s \le_{\mathsf{AL}^\tau_{\mathsf{seq}}} t \iff s \le_{\mathsf{AL}^\tau_{\mathsf{cc,seq}}} t$. The hard part is $s \le_{\mathsf{AL}^\tau_{\mathsf{seq}}} t \implies s \le_{\mathsf{AL}^\tau_{\mathsf{cc,seq}}} t$. Lemma 3.7 holds in the typed calculi as well, and thus it suffices to consider closed s, t. The AP_i-context lemma (Theorem 5.1) can be used, where the arguments are closed.

The main argument concerns the following situation: There are closed equally typed $\mathsf{AL}^\tau_{\mathsf{seq}}$-expressions s, t, such that $s \le_{\mathsf{AL}^\tau_{\mathsf{seq}}} t$, but we assume that $s \le_{\mathsf{AL}^\tau_{\mathsf{cc,seq}}} t$ does not hold. Since s, t must have a type without constructed types and since the AP_i-context lemma holds, there is an $n \ge 0$, and $v_i, i = 1, \ldots, n$, that are `Bot` or $\mathsf{AL}^\tau_{\mathsf{cc,seq}}$-values, and where all v_i are of an $\mathsf{AL}^\tau_{\mathsf{seq}}$-type, such that $s\ v_1 \ldots v_n{\downarrow}_{\mathsf{AL}^\tau_{\mathsf{cc,seq}}}$, but $t\ v_1 \ldots v_n{\Uparrow}_{\mathsf{AL}^\tau_{\mathsf{cc,seq}}}$. The goal is to show that there are $\mathsf{AL}^\tau_{\mathsf{seq}}$-expressions v'_i that are `Bot` or $\mathsf{AL}^\tau_{\mathsf{seq}}$-values, such that $s\ v'_1 \ldots v'_n{\downarrow}_{\mathsf{AL}^\tau_{\mathsf{seq}}}$, and $t\ v'_1 \ldots v'_n{\Uparrow}_{\mathsf{AL}^\tau_{\mathsf{seq}}}$ which refutes $s \le_{\mathsf{AL}^\tau_{\mathsf{seq}}} t$ and thus leads to a contradiction. It is sufficient to show that for every $\mathsf{AL}^{\tau}_{\mathsf{cc,seq}}$-value v and context C with $C[v]{\downarrow}_{\mathsf{AL}^{\tau}_{\mathsf{cc,seq}}}$, there is an $\mathsf{AL}^{\tau}_{\mathsf{seq}}$-value v', with $v' \leq_{\mathsf{AL}^{\tau}_{\mathsf{cc,seq}}} v$, such that $C[v']{\downarrow}_{\mathsf{AL}^{\tau}_{\mathsf{cc,seq}}}$.

In order to construct the proof we define simplification transformations in our monomorphically typed calculi, whenever the appropriate constructs exist in the calculus.

Definition 5.7. *The* simplification rules *(caseapp), (casecase), (seqseq), (seqapp), (seqcase), (caseseq), (botapp), (botcase), and (botseq) are defined in Figs. 2 and 4, where we use the typed variants. For $D \in \{\mathsf{AL}^{\tau}_{\mathsf{cc,seq}}, \mathsf{AL}^{\tau}_{\mathsf{cc}}\}$ let $\xrightarrow{Dx}$ denote the reduction using normal order reductions and simplification rules in a reduction context, where in case of a conflict the topmost redex is reduced. If $s \xrightarrow{Dx,*} v$ for some D-answer v, then we denote this as $s{\downarrow}_{Dx}$.*

Let $\xrightarrow{bcsfC}$ denote the reduction in any context by (β), (case), (seq), and (fix).

The simplifications are correct in the calculi under consideration (Lemma C.10) and they do not change the normal order reduction length (Lemma D.4):

Lemma 5.8. *In the calculi $\mathsf{AL}^{\tau}_{\mathsf{cc}}$ and $\mathsf{AL}^{\tau}_{\mathsf{cc,seq}}$: The simplification rules preserve the length of (converging) normal order reductions, i.e. let d be a simplification rule and $D \in \{\mathsf{AL}^{\tau}_{\mathsf{cc}}, \mathsf{AL}^{\tau}_{\mathsf{cc,seq}}\}$: if $s \xrightarrow{d} s'$ then $s \xrightarrow{n}_D v$, where v is a D-WHNF, if and only if $s' \xrightarrow{n}_D v'$, where v' is a D-WHNF.*

Lemma 5.9. *For $D \in \{\mathsf{AL}^{\tau}_{\mathsf{cc,seq}}, \mathsf{AL}^{\tau}_{\mathsf{cc}}\}$ we have ${\downarrow}_D = {\downarrow}_{Dx}$.*

Proof. Since the simplification rules are correct in $\mathsf{AL}^{\tau}_{\mathsf{cc,seq}}$, $\mathsf{AL}^{\tau}_{\mathsf{cc}}$, $s{\downarrow}_{Dx}$ implies that $s{\downarrow}_D$. Now assume that $s{\downarrow}_D$. We use induction on the number of (β), (case), (seq), (fix)-reductions of s to a WHNF. If s is a WHNF, then it is irreducible w.r.t. $\xrightarrow{Dx}$. If s has a normal order reduction of length $n > 0$ to a WHNF, then consider a $\xrightarrow{Dx}$-reduction sequence $s \xrightarrow{Dx,*} s_0$, where s_0 is a D-WHNF. Lemma 5.8 and termination of the simplifications (Lemma D.3) show that there are s', s'', such that $s \xrightarrow{Dx,*} s' \to_D s''$, where $s \xrightarrow{Dx,*} s'$ consists only of simplification rules. Lemma 5.8 shows that the normal order reduction length of s'' to a WHNF is smaller than n. Now we can apply the induction hypothesis.

Definition 5.10. *The following approximation procedure computes for every D-expression t (for $D \in \{\mathsf{AL}^{\tau}_{\mathsf{cc,seq}}, \mathsf{AL}^{\tau}_{\mathsf{cc}}\}$) and every depth i an approximating expression $approx(t,i) \leq_D t$. First a pre-approximation is computed where $preapprox(t,0) := \mathtt{Bot}$. If there is an infinite $\xrightarrow{Dx}$-reduction sequence starting with t, then $preapprox(t,i) := \mathtt{Bot}$ for all for $i > 0$. Otherwise, let $t \xrightarrow{Dx,*} t'$ where t' is irreducible for $\xrightarrow{Dx}$. Let M be the multicontext derived from t' where every subexpression at depth one is a hole, such that $t' = M(t_1,\ldots,t_k)$, and t_j, for $1 \leq j \leq k$, are subexpressions at depth 1. Let $t'_j = preapprox(t_j, i-1)$ for $j = 1,\ldots,k$, and define the result as $preapprox(t,i) := M(t'_1,\ldots,t'_k)$.*

Finally, $approx(t,i)$ is computed from $preapprox(t,i)$ by computing its normal form under the bot-simplifications in Fig. 4.

E.g. for $t = \mathtt{seq}\ (\mathtt{seq}\ x\ id)\ id$ first $t \xrightarrow{Dx,*} (\mathtt{seq}\ x\ (\mathtt{seq}\ id\ id))$. Replacing the subexpressions at depth 1 by $\mathtt{Bot}$ results in $preapprox(t,1) = (\mathtt{seq}\ \mathtt{Bot}\ \mathtt{Bot})$ which reduces to $approx(t,1) = \mathtt{Bot}$. Similarly, $preapprox(t,2) = approx(t,2) = (\mathtt{seq}\ x\ \lambda z.\mathtt{Bot})$.

Lemma 5.11. *For $D \in \{\mathsf{AL}^{\tau}_{\mathsf{cc,seq}}, \mathsf{AL}^{\tau}_{\mathsf{cc}}\}$: $approx(t,i) \leq_D t$.*

Proof. By Lemma 5.9, since all reductions are correct, and since cutting by $\mathtt{Bot}$ makes expressions smaller w.r.t. $\leq_D$.

We show a variant of the so-called subterm property for approximations:

Lemma 5.12. *The approximations $approx(t,i)$ are of the same type as t and irreducible w.r.t. the simplification rules and $\xrightarrow{bcsfC}$-irreducible. If t is an $\mathsf{AL}^{\tau}_{\mathsf{cc,seq}}$-expression of $\mathsf{AL}^{\tau}_{\mathsf{seq}}$-type, then $approx(t,i)$ is an $\mathsf{AL}^{\tau}_{\mathsf{seq}}$-expression. If t is an $\mathsf{AL}^{\tau}_{\mathsf{cc}}$-expression of AL^{τ}-type, then $approx(t,i)$ is an AL^{τ}-expression.*

Proof. The expressions $approx(t,i)$ have the same type as t. Only bot-simplifications may be possible, and these can only enable other bot-simplifications and thus, every $approx(t,i)$ is irreducible w.r.t. the simplification rules. It remains to show that $a := approx(t,i)$ must be an $\mathsf{AL}^{\tau}_{\mathsf{seq}}$-expression ($\mathsf{AL}^{\tau}$-expression, resp.). W.l.o.g. we consider the case with `seq`-expressions. It is sufficient to use the facts that *approx*(t,i) is irreducible w.r.t. the simplification rules and $\xrightarrow{bcsfC}$,and of $\mathsf{AL}^{\tau}_{\mathsf{seq}}$-type.

Suppose that there is a subexpression in $a = approx(t,i)$ of non-$\mathsf{AL}^{\tau}_{\mathsf{seq}}$-type. We select the subexpressions of non-$\mathsf{AL}^{\tau}_{\mathsf{seq}}$-type that are not contained in another subexpression of non-$\mathsf{AL}^{\tau}_{\mathsf{seq}}$-type; let s denote the one of a maximal non-$\mathsf{AL}^{\tau}_{\mathsf{seq}}$-type among these subexpressions. Since a is closed, we obtain that s cannot be a variable, since then either there is a superterm of s that is an abstraction of non-$\mathsf{AL}^{\tau}_{\mathsf{seq}}$-type, or a case-expression of non-$\mathsf{AL}^{\tau}_{\mathsf{seq}}$-type. Since a is of $\mathsf{AL}^{\tau}_{\mathsf{seq}}$-type, and s is maximal, there must be an immediate superterm s' of s which is of $\mathsf{AL}^{\tau}_{\mathsf{seq}}$-type. We look for the structure of s'. Due to the maximality conditions, s' cannot be an abstraction, an application of the form $(s_0\ s)$, a constructor application, a seq-expression of the form $(\mathtt{seq}\ s_0\ s)$, or a case-alternative, since then it would also have a non-$\mathsf{AL}^{\tau}_{\mathsf{seq}}$-type. It may be an application $(s\ s_2)$, a seq-expression $(\mathtt{seq}\ s\ s_2)$, or a case expression $\mathtt{case}\ s\ alts$.

First assume that s' is an application, then let s_0 be the leftmost and topmost non-application in s, i.e. $s' = (s_0\ r_1 \dots r_n)$, and $s = (s_0\ r_1 \dots r_{n-1})$, $n \geq 1$, where s_0 is not an application. The expression s_0 must be of non-$\mathsf{AL}^{\tau}_{\mathsf{seq}}$-type. Then s_0 cannot be `Bot`, an abstraction, `Fix`, a `case`-expression, or a `seq`-expression, since otherwise the subterm $s_0\ r_1$ would be reducible by (botapp), (β), (fix), (caseapp), or (seqapp). s_0 cannot be a constructor application either, due to types. Hence s' is not an application.

If s' is a case expression $\mathtt{case}_K\ s\ alts$, then s cannot be `Bot`, a `case`-expression, a `seq`-expression, or a constructor-application, since otherwise s would be reducible by (botcase), (casecase), (case), or (caseseq). Due to typing s cannot be an abstraction or `Fix`, and finally s' cannot be an application using the arguments above. Hence s' is not a case-expression.

If s' is a `seq`-expression $\mathtt{seq}\ s\ s_2$, then s cannot be `Bot`, an abstraction, `Fix`, a constructor application, a `case`-expression, or a `seq`-expression, since then s' would be reducible by (botseq), (seq), (seqcase), or (seqseq). s cannot be an application either, as argued above. Hence s' cannot be `seq`-expression.
In summary, such a subexpression does not exist, i.e. *approx*(t,i) is an $\mathsf{AL}^{\tau}_{\mathsf{seq}}$-expression.

In the following we use $s|_p$ for the subterm of s at position p, and $s[\cdot]_p$ for the expression s where the subterm at position p is replaced by a context hole.

Definition 5.13. *For an $\mathsf{AL}^{\tau}_{\mathsf{cc,seq}}$-expression ($\mathsf{AL}^{\tau}_{\mathsf{cc}}$-expression, resp.) s, a position p, and a subexpression s' such that $s|_p = s'$ the* non-R-depth *of s' at p is the number of prefixes p' of p s.t. $s[\cdot]_{p'}$ is not a reduction context.*

Lemma 5.14. *For $D \in \{\mathsf{AL}^{\tau}_{\mathsf{cc,seq}}, \mathsf{AL}^{\tau}_{\mathsf{cc}}\}$, a D-expression t, a D-context C with $C[t]{\downarrow_D}$ there is some i and an approximation $approx(t,i)$ with $C[approx(t,i)]{\downarrow_D}$.*

Proof. Let $C[t] \xrightarrow{n}_D t_0$, where t_0 is a D-WHNF. Then compute $t' := approx(t, n+1)$. The construction of $approx(t, n+1)$ includes (β)-, (case)-, (seq)- and (fix)-reductions and simplification rules. Let A be the set of all the simplification rules. We have $C[t] \xrightarrow{bcsfC \cup A,*} C[t'']$, where t' is t'' with subexpressions replaced by `Bot`. Since reductions and simplifications are correct, we have $C[t'']{\downarrow_D}$, and in particular, the number of normal order reductions of $C[t'']$ to a D-WHNF is $n' \leq n$ (proven in the appendix, Lemmas D.4 and D.5).

The normal order reduction for $C[t']$ makes the same reduction steps as the normal order reduction of $C[t'']$ since the `Bot`-expressions placed by the approximation are in the beginning at the non-R-depth $n+1$, and remain at non-R-depth $\geq n+1-j$ after j normal order reductions. Finally, they will be at non-R-depth of at least 1, hence the final D-WHNF may have `Bot`s only at non-R-depth of at least 1, and so it is a WHNF. Thus $C[approx(t,n)]{\downarrow_D}$.

Theorem 5.15. *The embeddings of AL^{τ} in $\mathsf{AL}^{\tau}_{\mathsf{cc}}$ and of $\mathsf{AL}^{\tau}_{\mathsf{seq}}$ in $\mathsf{AL}^{\tau}_{\mathsf{cc,seq}}$ are conservative.*

Proof. We prove this for the embedding of $\mathsf{AL}^{\tau}_{\mathsf{seq}}$ in $\mathsf{AL}^{\tau}_{\mathsf{cc,seq}}$. The other case is similar. Let s, t be $\mathsf{AL}^{\tau}_{\mathsf{seq}}$-expressions with $s \leq_{\mathsf{AL}^{\tau}_{\mathsf{seq}}} t$. We have to show that $s \leq_{\mathsf{AL}^{\tau}_{\mathsf{cc,seq}}} t$. Assume this is false. Since the AP_i-context lemma holds (Theorem 5.1) the assumption implies that there is an $n \geq 0$ and closed $\mathsf{AL}^{\tau}_{\mathsf{cc,seq}}$-expressions $b_1, \dots, b_n$ of $\mathsf{AL}^{\tau}_{\mathsf{seq}}$-type which are answers or `Bot`, such that $(s\ b_1 \dots b_n){\downarrow_{\mathsf{AL}^{\tau}_{\mathsf{cc,seq}}}}$ but $(t\ b_1 \dots b_n){\Uparrow_{\mathsf{AL}^{\tau}_{\mathsf{cc,seq}}}}$. According to Lemma 5.14, we have successively constructed the approximations b'_i of b_i of a depth depending on the length of the normal order reduction of $(s\ b_1 \dots b_n)$, such that $(s\ b'_1 \dots b'_n){\downarrow_{\mathsf{AL}^{\tau}_{\mathsf{cc,seq}}}}$ but $(t\ b'_1 \dots b'_n){\Uparrow_{\mathsf{AL}^{\tau}_{\mathsf{cc,seq}}}}$, also using Lemma 5.11. Lemma 5.12 shows that the approximations are in the smaller calculus $\mathsf{AL}^{\tau}_{\mathsf{seq}}$, and thus also $(s\ b'_1 \dots b'_n){\downarrow_{\mathsf{AL}^{\tau}_{\mathsf{seq}}}}$ but $(t\ b'_1 \dots b'_n){\Uparrow_{\mathsf{AL}^{\tau}_{\mathsf{seq}}}}$, which contradicts $s \leq_{\mathsf{AL}^{\tau}_{\mathsf{seq}}} t$.

The same reasoning can be used to show the following result (of practical interest) for $D \in \{\mathsf{AL}^{\tau}_{\mathsf{cc}}, \mathsf{AL}^{\tau}_{\mathsf{cc,seq}}\}$: Assume that the set of type and data constructors is a fixed set in D, and that D'

is an extension of D such that only new type and data constructors are added. Then D' is a conservative extension of D, since we can use the approximation technique from this section to approximate D'-values by D-values and then apply the AP_i-context lemma.

6 Polymorphically Typed Calculi

We consider polymorphically typed variants $\mathsf{AL}^\alpha, \mathsf{AL}^\alpha_{\mathsf{seq}}, \mathsf{AL}^\alpha_{\mathsf{cc}}, \mathsf{AL}^\alpha_{\mathsf{cc,seq}}$ of the four calculi. We will show non-conservativity of embedding AL^α in $\mathsf{AL}^\alpha_{\mathsf{seq}}$ and $\mathsf{AL}^\alpha_{\mathsf{cc}}$ in $\mathsf{AL}^\alpha_{\mathsf{cc,seq}}$, but leave open the question of (non-)conservativity of embedding AL^α in $\mathsf{AL}^\alpha_{\mathsf{cc}}$ and $\mathsf{AL}^\alpha_{\mathsf{seq}}$ in $\mathsf{AL}^\alpha_{\mathsf{cc,seq}}$.

The expression syntax is the untyped one. The syntax for polymorphic types $\overline{T}$ is $\overline{T} ::= V \mid \overline{T}_1 \to \overline{T}_2 \mid (K\ \overline{T}_1 \dots \overline{T}_{ar(K)})$ where V is a type variable. The constructors have predefined Hindley-Milner polymorphic types according to the usual standards. Only expressions that are Hindley-Milner polymorphically typed are permitted. Normal order reduction is defined only on monomorphic type-instances of expressions, which is a deviation from Definition 2.1.

Definition 6.1. *For $D \in \{\mathsf{AL}^\alpha, \mathsf{AL}^\alpha_{\mathsf{seq}}, \mathsf{AL}^\alpha_{\mathsf{cc}}, \mathsf{AL}^\alpha_{\mathsf{cc,seq}}\}$ and for $s, t \in D$ of equal polymorphic type: $s \leq_D t$ iff $\rho(s) \leq_{D'} \rho(t)$ for all monomorphic type instantiations ρ, where D' is the corresponding monomorphically typed calculus. Contextual equivalence is defined by $s \sim_D t$ iff $s \leq_D t \wedge t \leq_D s$.*

Since s_{11}, s_{12} (Sect. 5.1) are of polymorphic type $(a \to \mathtt{Bool}) \to \mathtt{Bool}$, the same arguments as for the proof of Theorem 5.6 can be applied, hence:

Theorem 6.2. *The natural embedding of $\mathsf{AL}^\alpha_{\mathsf{cc}}$ into $\mathsf{AL}^\alpha_{\mathsf{cc,seq}}$ is not conservative.*

We observe that encoding cases and constructors in s_{11}, s_{12} in Subsection 5.1 does not produce a counterexample:

$$s'_{11} := \lambda x.x\ \mathtt{Bot}\ id$$
$$s'_{12} := \lambda x.x\ (\lambda y.\mathtt{Bot})\ id$$

These two expressions are not equivalent in AL^α, since application to $\lambda u, v.u$ extract the first argument, and thus the expressions behave differently.
A successful example is the following. Let s_{13}, s_{14} of the polymorphic type $((\alpha \to \alpha) \to (\alpha \to \alpha) \to (\alpha \to \alpha)) \to (\alpha \to \alpha))$ be defined as: $s_{13} := \lambda x.x\ id\ (x\ \mathtt{Bot}\ id)$ and $s_{14} := \lambda x.x\ id\ (x\ (\lambda y.\mathtt{Bot})\ id)$.

Lemma 6.3. *For AL^τ-expressions $t = M[\mathtt{Bot}, \dots, \mathtt{Bot}]$, $t' = M[\lambda x.\mathtt{Bot}, \dots, \lambda x.\mathtt{Bot}]$, and $t\Uparrow_{\mathsf{AL}^\tau}$, $t'\downarrow_{\mathsf{AL}^\tau}$ it holds that $M[x_1, \dots, x_n] \xrightarrow{*}_{\mathsf{AL}^\tau} x_i$ for some i.*

Proof. This follows by observing a normal order reduction of t, t' and comparing the first use of $\mathtt{Bot}$, or $\lambda x.\mathtt{Bot}$, respectively. There must be a use of this argument, since otherwise the observations are identical. If it is ever used in a function position in a beta-reduction, then both expressions diverge. Hence, the only possibility is that they are returned.

Theorem 6.4. *The embedding of AL^α into $\mathsf{AL}^\alpha_{\mathsf{seq}}$ is not conservative. The embedding of AL^α into $\mathsf{AL}^\alpha_{\mathsf{cc,seq}}$ is also not conservative.*

Proof. Since $(\rho(s_{13})\ (\lambda u, v.\mathtt{seq}\ u\ v))\Uparrow_{\mathsf{AL}^\tau}$, but $(\rho(s_{14})\ (\lambda u, v.\mathtt{seq}\ u\ v))\downarrow_{\mathsf{AL}^\tau}$ for $\rho = \{\alpha \mapsto o\}$, we have $s_{13} \not\sim_{\mathsf{AL}^\alpha_{\mathsf{seq}}} s_{14}$ as well as $s_{13} \not\sim_{\mathsf{AL}^\alpha_{\mathsf{cc,seq}}} s_{14}$. It remains to show that $s_{13} \sim_{\mathsf{AL}^\alpha} s_{14}$ holds, i.e. that $\rho(s_{13}) \sim_{\mathsf{AL}^\tau} \rho(s_{14})$ for any monomorphic type instantiation ρ of the type $((a \to a) \to (a \to a) \to (a \to a)) \to (a \to a))$. We use the AP_i-context lemma (Theorem 5.1) and assume that there is an n, a closed AL^τ-expression s, and closed arguments $b_1, \dots, b_n$, such that $\rho(s_{13})\ s\ b_1 \dots b_n$ is typed in AL^τ, and $\rho(s_{13})\ s\ b_1 \dots b_n\Uparrow_{\mathsf{AL}^\tau}$, $\rho(s_{14})\ s\ b_1 \dots b_n\downarrow_{\mathsf{AL}^\tau}$. By Lemma 6.3, the only possibility is that the $\mathtt{Bot}$, and $\lambda x.\mathtt{Bot}$-positions are extracted. By the type preservation, and since the type of $\rho(s_{13})\ s$ is the type of the $\mathtt{Bot}$-position, it is impossible that $n > 0$, since then the type of the result is smaller than the type of the $\mathtt{Bot}$-position. Hence $s\ id\ (s\ y\ id) \xrightarrow{*}_{\mathsf{AL}^\tau} y$. But since the y occurs in the expression $(s\ y\ id)$, we also have $(s\ y\ id) \xrightarrow{*}_{\mathsf{AL}^\tau} y$. This implies that $(s\ id\ y) \xrightarrow{*}_{\mathsf{AL}^\tau} y$. But then the normal order reduction of $s\ x_1\ x_2$ cannot apply either of its arguments x_1, x_2, and hence must be a projection to one of the arguments, which is impossible, since it must project to both arguments. We conclude that $\rho(s_{13})$ and $\rho(s_{14})$ cannot be distinguished in all approximation contexts, and the reasoning does not depend on ρ. Hence $s_{13} \sim_{\mathsf{AL}^\alpha} s_{14}$.

The expressions s_{13}, s_{14} could also be used to show non-conservativity of embedding AL^{τ} into $\mathsf{AL}^{\tau}_{\mathsf{seq}}$. Hence there are also examples at higher types as witnesses for Theorem 5.6.

Whether adding case and constructors is conservative or not in the polymorphic case, for AL^{α} as well as for $\mathsf{AL}^{\alpha}_{\mathsf{seq}}$ remains an open problem. Also the appendix with the pair example should only be part of the technical report?

Remark 6.5. It is an open question whether the embeddings of AL^{α} into $\mathsf{AL}^{\alpha}_{\mathsf{cc}}$ and $\mathsf{AL}^{\alpha}_{\mathsf{seq}}$ into $\mathsf{AL}^{\alpha}_{\mathsf{cc,seq}}$ are conservative. Using bisimulation for a proof that $s \sim_{\mathsf{AL}^{\alpha}} t$ implies that $s \sim_{\mathsf{AL}^{\alpha}_{\mathsf{cc}}} t$ has to try $\mathsf{AL}^{\alpha}_{\mathsf{cc}}$-arguments b_i of s, t. In contrast to the monomorphically typed conservativity proof, also b_i with a non-AL^{α}-type have to be checked, hence an approximation does not work. Another potential method would be a translation of b_i using Church encoding of the cases and constructors. This, however, is not possible in the Hindley-Milner polymorphic typing system, since, for example, the translation will enforce equal types of the component of a pair, and thus does not allow to program the copy-function of pairs [SSNSS08,MJJS09]. An extended example is in the appendix (Sect. **??**).

Forgetting Types. Now we look for the translations defined as "forgetting" the types, and ask for adequacy and full abstraction, which plays now the role of conservativity. For the monomorphically typed calculi the answer is obvious: these translations are not fully abstract. For example $\lambda x^o.x^o$ is equivalent to $\lambda x^o.\bot^o$, which refutes full abstractness in all cases. For the polymorphically typed calculi, this question is non-trivial:

Proposition 6.6. *The translations of AL^{α} into AL, $\mathsf{AL}^{\alpha}_{\mathsf{cc}}$ into $\mathsf{AL}_{\mathsf{cc}}$, and $\mathsf{AL}^{\alpha}_{\mathsf{cc,seq}}$ into $\mathsf{AL}_{\mathsf{cc,seq}}$ by simply forgetting the types are adequate but not fully-abstract.*

Proof. For the first case, AL^{α} and AL, we have $s_{13} \sim_{\mathsf{AL}^{\alpha}} s_{14}$, but $(s_{13}\ \lambda u, v.(v\ (\lambda x.u)))\Uparrow$ and $(s_{14}\ \lambda u, v.(v\ (\lambda x.u)))\downarrow$. For the other calculi, $\lambda x.\mathtt{case}_{Bool}\ x\ (\mathtt{True} \texttt{ -> } \mathtt{True})\ (\mathtt{False} \texttt{ -> } \mathtt{False})$ is equivalent to $\lambda x^{\mathtt{Bool}}.x$, but in the untyped case, $([\cdot]\ \lambda z.z)$ distinguishes these expressions.

Full abstractness of forgetting types in $\mathsf{AL}^{\alpha}_{\mathsf{seq}}$ also remains an open question.

7 Conclusion

We have shown that the semantics of the pure lazy lambda calculus changes when `seq`, or `case` and constructors, are added. Under the insight that any semantic investigation for Haskell should include the `seq`-operator, we exhibited calculus extensions that are useful for the analysis of expression equivalences that also hold in a realistic core calculus of lazy functional and typed languages. We left the rigorous analysis of the implication chain for equivalence from $\mathsf{AL}^{\tau}_{\mathsf{cc,seq}}$ to the polymorphic calculus with letrec for future research.

References

Abr90. S. Abramsky. The lazy lambda calculus. In *Research Topics in Functional Programming*, pages 65–116, 1990.

AMR06. A. Arbiser, A. Miquel, and A. Rios. A lambda-calculus with constructors. In *Proc. RTA 2006*, volume 4098 of *LNCS*, pages 181–196, 2006.

DM79. N. Dershowitz and Z. Manna. Proving termination with multiset orderings. *Commun. of the ACM*, 22:465–476, 1979.

dV89. R. C. de Vrijer. Extending the lambda calculus with surjective pairing is conservative. In *Proc. LICS 1989*, pages 204–215, 1989.

Fel91. M. Felleisen. On the expressive power of programming languages. *Sci. Comput. Programming*, 17(1–3):35–75, 1991.

HH10. S. Holdermans and J. Hage. Making "stricterness" more relevant. In *Proc. PEPM 2010*, pages 121–130, 2010.

How89. D. Howe. Equality in lazy computation systems. In *Proc. LICS 1989*, pages 198–203, 1989.

How96. D. Howe. Proving congruence of bisimulation in functional programming languages. *Inform. and Comput.*, 124(2):103–112, 1996.

JV06. P. Johann and J. Voigtländer. The impact of seq on free theorems-based program transformations. *Fund. Inform.*, 69(1–2):63–102, 2006.

KSS98. A. Kutzner and M. Schmidt-Schauß. A nondeterministic call-by-need lambda calculus. In *Proc. ICFP 1998*, pages 324–335, 1998.

MJJS09. M., J., J., and D. Sabel. Adequacy of compositional translations for observational semantics. Frank report 33, Inst. f. Informatik, Goethe-University, Frankfurt, 2009. `http://www.ki.informatik.uni-frankfurt.de/papers/frank/`.

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

MS99. A. K. D. Moran and D. Sands. Improvement in a lazy context: An operational theory for call-by-need. In *Proc. POPL 1999*, pages 43–56, 1999.

Pey03. S. Peyton Jones. *Haskell 98 language and libraries: the revised report.* 2003.

PS98. S. L. Peyton Jones and A. L. M. Santos. A transformation-based optimiser for Haskell. *Sci. Comput. Programming*, 32(1–3):3–47, 1998.

RS94. J. G. Riecke and R. Subrahmanyam. Extensions to type systems can preserve operational equivalences. In *Proc. TACS 1994*, pages 76–95, 1994.

Ses97. P. Sestoft. Deriving a lazy abstract machine. *J. Funct. Programming*, 7(3):231–264, 1997.

SSNSS08. M. Schmidt-Schauß, J. Niehren, J. Schwinghammer, and D. Sabel. Adequacy of compositional translations for observational semantics. In *5th IFIP TCS 2008*, volume 273, pages 521–535, 2008.

SSS11. D. Sabel and M. Schmidt-Schauß. A contextual semantics for Concurrent Haskell with futures. In *Proc. PPDP 2011*, pages 101–112, 2011.

SSSM10. M. Schmidt-Schauß, D. Sabel, and E. Machkasova. Simulation in the call-by-need lambda-calculus with letrec. In *Proc. RTA 2010*, volume 6 of *LIPIcs*, pages 295–310, 2010.

SSSM12. M. Schmidt-Schauß, D. Sabel, and E. Machkasova. Simulation in the call-by-need lambda-calculus with letrec, case, constructors, and seq. Frank report 49, Inst. f. Informatik, Goethe-University, Frankfurt, 2012.

SSSS08. M. Schmidt-Schauß, M. Schütz, and D. Sabel. Safety of Nöcker's strictness analysis. *J. Funct. Programming*, 18(04):503–551, 2008.

Stø06. K. Støvring. Extending the extensional lambda calculus with surjective pairing is conservative. *Log. Methods Comput. Sci.*, 2(2), 2006.

TLC10. TLCA. List of open problems, 2010. `http://tlca.di.unito.it/opltlca/`.

Wad89. P. Wadler. Theorems for free! In *Proc. FPCA 1989*, pages 347–359, 1989.

A Bisimilarity and Finite Simulation

We define the notion of bisimilarity in a general way, and then show that bisimilarity coincides with a restricted form of contextual equivalence, provided the corresponding TDPC is convergence-admissible (see Definition A.7).

Definition A.1. *For a TDPC $D = (\mathcal{E}, \mathcal{C}, \rightarrow_D, \mathcal{A}, \mathcal{T})$ a* set of experiments *$\mathcal{Q} \subseteq \mathcal{C}$ is a set of contexts which preserves closedness, i.e. $Q(s)$ is a closed expression for every $Q \in \mathcal{Q}$ and $s \in \mathcal{E}^c$. We write $Q \in \mathcal{Q}_{T,T'}$ if Q is a context which behaves like a function from $\mathcal{E}_T$ into $\mathcal{E}_{T'}$.*

A binary relation χ on typed expressions is called *type-preserving*, if $s \chi t$ always implies that s and t are of the same type.

Definition A.2 ($\mathcal{Q}$-Similarity). *Let $D = (\mathcal{E}, \mathcal{C}, \rightarrow_D, \mathcal{A}, \mathcal{T})$ be a TDPC and $\mathcal{Q}$ be a set of experiments. The experiment-operator $F_\mathcal{Q}$ on type-preserving binary relations χ on closed D-expressions is defined as follows: The relation $s\, F_\mathcal{Q}(\chi)\, t$ for closed, equally typed expressions s, t of type T holds iff $s{\downarrow_D} v \implies \big(t{\downarrow_D} v' \wedge$ for all types T' and all $Q \in \mathcal{Q}_{T,T'}\colon Q(v)\, \chi\, Q(v')\big)$.*

$\mathcal{Q}$-similarity $\leq_{b,D,\mathcal{Q}}$ *is the greatest fixpoint of the operator $F_\mathcal{Q}$.* $\mathcal{Q}$-bisimilarity $\sim_{b,D,\mathcal{Q}}$ *is defined as $\leq_{b,D,\mathcal{Q}} \cap \geq_{b,D,\mathcal{Q}}$ where $\geq_{b,D,\mathcal{Q}}$ is the inverse of $\leq_{b,D,\mathcal{Q}}$*

Definition A.3. *For a relation χ on closed expressions, let χ^o be the extension to open expression, defined by $s\ \chi^o\ t$ iff for all closing substitutions σ, the relation $\sigma(s)\ \chi\ \sigma(t)$ holds.*

We first show that normal order reductions of a TDPC preserve $\mathcal{Q}$-bisimilarity.

Lemma A.4. *Let $D = (\mathcal{E}, \mathcal{C}, \rightarrow_D, \mathcal{A}, \mathcal{T})$ be a TDPC and $\mathcal{Q}$ be a set of experiments. Then for any expressions $s_1, s_2 \in \mathcal{E}^c$ with $s_1 \rightarrow_D s_2$ we have $s_1 \sim_{b,D,\mathcal{Q}} s_2$.*

Proof. Let $M := \rightarrow_D \cup \{(s,s) \mid s \in \mathcal{E}^c\}$, $\overline{M} := \{(s_2, s_1) \mid (s_1, s_2) \in M\}$. Then the relations M and $\overline{M}$ are $F_\mathcal{Q}$-dense, i.e. $X \subseteq F_\mathcal{Q}(X)$ for $X \in \{M, \overline{M}\}$ holds, since $\rightarrow_D$ is deterministic, type-preserving, and closedness-preserving. Since $\leq_{b,D,\mathcal{Q}} = \bigcup\{X \mid X \subseteq F_\mathcal{Q}(X)\}$, $F_\mathcal{Q}$-density of M and $\overline{M}$ implies that $M \subseteq \sim_{b,D,\mathcal{Q}}$.

Now we define the notion of finite simulation.

Definition A.5 (Finite Simulation). *Let D be a typed deterministic calculus and let $\mathcal{Q}$ be a set of experiments.* Finite simulation $\preceq_{D,\mathcal{Q}}$ *is defined as follows:*

For $s_1, s_2 \in \mathcal{E}^c_{T_n}$ the relation $s_1 \preceq_{D,\mathcal{Q}} s_2$ holds if and only if:
$\forall n \geq 0, T_0 \in \mathcal{T} : \forall Q_i \in \mathcal{Q}_{T_i,T_{i-1}} : Q_1(Q_2(\ldots(Q_n(s_1)))){\downarrow_D} \implies Q_1(Q_2(\ldots(Q_n(s_2)))){\downarrow_D}$

Finite bisimulation $\approx_{D,\mathcal{Q}}$ *is defined as $\preceq_{D,\mathcal{Q}} \cap \succeq_{D,\mathcal{Q}}$.*

If $\approx_{D,\mathcal{Q}}$ coincides with contextual equivalence, a sufficient criterion to show contextual equivalence is the following:

Proposition A.6. *Let $D = (\mathcal{E}, \mathcal{C}, \rightarrow_D, \mathcal{A}, \mathcal{T})$ and $\mathcal{Q}$ be a set of experiments, such that ${\approx_{D,\mathcal{Q}}}^o = \sim_D$. Two closed expressions $s_1, s_2 \in \mathcal{E}^c$ of type $T_i \in \mathcal{T}$ are contextually equivalent if there exists $i \in \mathbb{N}_0$ such that*

1. *$s_1{\downarrow_D} \iff s_2{\downarrow_D}$.*
2. *For all $1 < k \leq i$, and for all $Q_k, \ldots, Q_i$ with $Q_j \in \mathcal{Q}_{T_j,T_{j-1}}$ (for $k \leq j \leq i$): $Q_k(Q_{k+1}(\ldots(Q_i(s_1)\ldots))){\downarrow_D} \iff Q_k(Q_{k+1}(\ldots(Q_i(s_2)\ldots))){\downarrow_D}$.*
3. *For all $Q_1, \ldots, Q_i$ with $Q_j \in \mathcal{Q}_{T_j,T_{j-1}}$ (for $1 \leq j \leq i$): $Q_1(Q_2(\ldots(Q_i(s_1)\ldots))) \sim_D Q_1(Q_2(\ldots(Q_i(s_2)\ldots)))$.*

Proof. $s_1 \approx_{D,\mathcal{Q}} s_2$ holds, since $\sim_D$ is a congruence, $\mathcal{Q} \subseteq \mathcal{C}$, and $\approx_{D,\mathcal{Q}} = \sim_D$.

Definition A.7. *$D = (\mathcal{E}, \mathcal{C}, \rightarrow, \mathcal{A}, \mathcal{T})$ is* convergence-admissible w.r.t. a set of experiments $\mathcal{Q}$ *iff $\forall Q \in \mathcal{Q}_{T,T'}, s \in {\mathcal{E}_T}^c : Q(s){\downarrow_D} v \iff \exists v' : s{\downarrow_D} v' \wedge Q(v'){\downarrow_D} v$*

For the remainder of this section we assume a TDPC $D = (\mathcal{E}, \mathcal{C}, \rightarrow_D, \mathcal{A}, \mathcal{T})$ and a set of experiments $\mathcal{Q}$ to be given.

Lemma A.8. *For all expressions $s_1, s_2 \in \mathcal{E}^c$ of the same type T the following holds: $s_1 \leq_{b,D,\mathcal{Q}} s_2$ iff $s_1{\downarrow_D} v_1 \implies (s_2{\downarrow_D} v_2 \wedge \forall T' \in \mathcal{T}, Q \in \mathcal{Q}_{T,T'} : Q(v_1) \leq_{b,D,\mathcal{Q}} Q(v_2))$.*

Proof. Since $\leq_{b,D,\mathcal{Q}}$ is a fixpoint of $F_{\mathcal{Q}}$, we have $\leq_{b,D,\mathcal{Q}} = F_{\mathcal{Q}}(\leq_{b,D,\mathcal{Q}})$. This equation is equivalent to the claim of the lemma.

We show that $F_{\mathcal{Q}}$ is monotonic and lower-continuous, which allows us to apply Kleene's fixpoint theorem and leads to an alternative characterization of $\leq_{b,D,\mathcal{Q}}$.

Lemma A.9. *$F_{\mathcal{Q}}$ is monotonic w.r.t. set inclusion, i.e. for all type-preserving binary relations χ_1, χ_2 on closed expressions $\chi_1 \subseteq \chi_2 \implies F_{\mathcal{Q}}(\chi_1) \subseteq F_{\mathcal{Q}}(\chi_2)$.*

Proof. Let $\chi_1 \subseteq \chi_2$ and $s_1\, F_{\mathcal{Q}}(\chi_1)\, s_2$ with $s_1, s_2 \in \mathcal{E}_T$. The assumption implies $s_1{\downarrow_D} v_1 \implies (s_2{\downarrow_D} v_2 \wedge \forall T' \in \mathcal{T}, Q \in \mathcal{Q}_{T,T'} : Q(v_1)\, \chi_1\, Q(v_2))$. Now $\chi_1 \subseteq \chi_2$ shows $s_1{\downarrow_D} v_1 \implies (s_2{\downarrow_D} v_2 \wedge \forall T' \in \mathcal{T}, Q \in \mathcal{Q}_{T,T'} : Q(v_1)\, \chi_2\, Q(v_2))$.

For infinite sequences of sets $S_1, S_2 \ldots$, we define the greatest lower bound w.r.t. set-inclusion ordering as $\mathrm{glb}(S_1, S_2, \ldots) = \bigcap_{i=1}^{\infty} S_i$.

Proposition A.10. *$F_{\mathcal{Q}}$ is lower-continuous w.r.t. countably infinite descending chains $Ch = \chi_1 \supseteq \chi_2 \supseteq \ldots$ of type-preserving binary relations on closed expressions, i.e. $\mathrm{glb}(F_{\mathcal{Q}}(Ch)) = F_{\mathcal{Q}}(\mathrm{glb}(Ch))$ where $F_{\mathcal{Q}}(Ch)$ is the infinite descending chain $F_{\mathcal{Q}}(\chi_1) \supseteq F_{\mathcal{Q}}(\chi_2) \supseteq \ldots$.*

Proof. "$\supseteq$": Since $\mathrm{glb}(Ch) = \bigcap_{i=1}^{\infty} \chi_i$, we have for all i: $\mathrm{glb}(Ch) \subseteq \chi_i$. Monotonicity of $F_{\mathcal{Q}}$ shows $F_{\mathcal{Q}}(\mathrm{glb}(Ch)) \subseteq F_{\mathcal{Q}}(\chi_i)$ for all i. This implies $F_{\mathcal{Q}}(\mathrm{glb}(Ch)) \subseteq \bigcap_{i=1}^{\infty} F_{\mathcal{Q}}(\chi_i)$, i.e. $F_{\mathcal{Q}}(\mathrm{glb}(Ch)) \subseteq \mathrm{glb}(F_{\mathcal{Q}}(Ch))$.

"$\subseteq$": Let $(s_1, s_2) \in \mathrm{glb}(F_{\mathcal{Q}}(Ch))$, i.e. for all i: $(s_1, s_2) \in F_{\mathcal{Q}}(\chi_i)$. Unfolding the definition of $F_{\mathcal{Q}}$ gives: $\forall i : s_1{\downarrow_D} v_1 \implies (s_2{\downarrow_D} v_2 \wedge \forall Q \in \mathcal{Q} : Q(v_1)\, \chi_i\, Q(v_2))$. Now we can move the universal quantifier for i inside the formula: $s_1{\downarrow_D} v_1 \implies (s_2{\downarrow_D} v_2 \wedge \forall Q \in \mathcal{Q} : \forall \mathrm{i}\text{: } Q(v_1)\, \chi_i\, Q(v_2))$. This is equivalent to $s_1{\downarrow_D} v_1 \implies (s_2{\downarrow_D} v_2 \wedge \forall Q \in \mathcal{Q} : Q(v_1) \left(\bigcap_{i=1}^{\infty} \chi_i\right) Q(v_2))$ or $s_1{\downarrow_D} v_1 \implies (s_2{\downarrow_D} v_2 \wedge \forall Q \in \mathcal{Q} : (Q(v_1), Q(v_2)) \in \mathrm{glb}(Ch))$ and thus $(s_1, s_2) \in F_{\mathcal{Q}}(\mathrm{glb}(Ch))$.

Since the operator $F_{\mathcal{Q}}$ is monotonic and lower-continuous, we can apply Kleene's fixpoint theorem to derive an inductive description of $\mathcal{Q}$-similarity.

Theorem A.11. *Let $\leq_{b,D,\mathcal{Q},i}$ for $i \in \mathbb{N}_0$ be defined as follows:*

$$\leq_{b,D,\mathcal{Q},0} = \{\mathcal{E}_T^c \times \mathcal{E}_T^c \mid T \in \mathcal{T}\} \quad \text{and} \quad \leq_{b,D,\mathcal{Q},i} = F_{\mathcal{Q}}(\leq_{b,D,\mathcal{Q},i-1}), \text{if } i > 0$$

Then $\leq_{b,D,\mathcal{Q}} = \bigcap_{i=1}^{\infty} \leq_{b,D,\mathcal{Q},i}$

Proof. This follows by Kleene's fixpoint theorem since $F_{\mathcal{Q}}$ is monotonic and lower-continuous, and since $\leq_{b,D,\mathcal{Q},i+1} \subseteq \leq_{b,D,\mathcal{Q},i}$ for all $i \geq 0$.

We show some helpful properties of $\leq_{D,\mathcal{Q}}$:

Lemma A.12. *Let $D = (\mathcal{E}, \mathcal{C}, \rightarrow, \mathcal{A}, \mathcal{T})$ be convergence-admissible w.r.t. $\mathcal{Q}$. Then the following holds for all closed expressions s_1, s_2 of the same type T:*

- *$s_1 \preceq_{D,\mathcal{Q}} s_2 \implies Q(s_1) \preceq_{D,\mathcal{Q}} Q(s_2)$ for all $T' \in \mathcal{T}$ and $Q \in \mathcal{Q}_{T,T'}$*
- *$s_1 \preceq_{D,\mathcal{Q}} s_2, s_1{\downarrow_D} v_1,$ and $s_2{\downarrow_D} v_2 \implies v_1 \preceq_{D,\mathcal{Q}} v_2$*

Proof. The first part is easy to verify. For the second part let $s_1 \preceq_{D,\mathcal{Q}} s_2$, and $s_1{\downarrow_D} v_1$, $s_2{\downarrow_D} v_2$ hold. Assume that $Q_1(\ldots(Q_n(v_1))){\downarrow_D} v_1'$ for some $n \geq 0$ where all $Q_i \in \mathcal{Q}$. Convergence-admissibility implies $Q_1(\ldots(Q_n(s_1))){\downarrow_D} v_1'$. Now $s_1 \preceq_{D,\mathcal{Q}} s_2$ implies $Q_1(\ldots(Q_n(s_2))){\downarrow_D} v_2'$. Applying convergence-admissibility multiple times shows that $s_2{\downarrow_D} w_2$ and $Q_1(\ldots(Q_n(w_2))){\downarrow} v_2'$ for some answer w_2. Since normal order reduction is deterministic, the assumption $s_2{\downarrow_D} v_2$ shows $w_2 = v_2$ and thus $Q_1(\ldots(Q_n(v_2))){\downarrow} v_2'$.

We prove that $\leq_{b,D,\mathcal{Q}}$ respects functions $Q \in \mathcal{Q}$ provided the underlying TDPC is convergence-admissible w.r.t. $\mathcal{Q}$:

Lemma A.13. *Let $D = (\mathcal{E}, \mathcal{C}, \rightarrow, \mathcal{A}, \mathcal{T})$ be convergence-admissible w.r.t. $\mathcal{Q}$. For all $s_1, s_2 \in \mathcal{E}_T^c, Q \in \mathcal{Q}_{T,T'} : s_1 \leq_{b,D,\mathcal{Q}} s_2 \implies Q(s_1) \leq_{b,D,\mathcal{Q}} Q(s_2)$*

Proof. Let $s_1 \leq_{b,D,\mathcal{Q}} s_2$, $Q_0 \in \mathcal{Q}$, and $Q_0(s_1){\downarrow_D} v_1$. By convergence-admissibility $s_1{\downarrow_D} v_1'$ holds and $Q_0(v_1'){\downarrow_D} v_1$. Since $s_1 \leq_{b,D,\mathcal{Q}} s_2$ this implies $s_2{\downarrow_D} v_2'$ and for all $Q \in \mathcal{Q} : Q(v_1') \leq_{b,D,\mathcal{Q}} Q(v_2')$. Hence, from $Q_0(v_1'){\downarrow_D} v_1$ we derive $Q_0(v_2'){\downarrow_D} v_2$. Convergence-admissibility now implies $Q_0(s_2){\downarrow_D} v_2$. It remains to show for all $Q \in \mathcal{Q}$: $Q(v_1) \leq_{b,D,\mathcal{Q}} Q(v_2)$: Since $Q_0(v_1'){\downarrow_D} v_1$ and $Q_0(v_2'){\downarrow_D} v_2$, applying Lemma A.8 to $Q_0(v_1') \leq_{b,D,\mathcal{Q}} Q_0(v_2')$ implies $Q(v_1) \leq_{b,D,\mathcal{Q}} Q(v_2)$ for all $Q \in \mathcal{Q}$.

We show that $\preceq_{D,\mathcal{Q}}$ and $\mathcal{Q}$-similarity coincide for convergence-admissible TDPC:

Theorem A.14. *Let $D = (\mathcal{E}, \mathcal{C}, \to, \mathcal{A}, \mathcal{T})$ be convergence-admissible w.r.t. $\mathcal{Q}$. Then $\preceq_{D,\mathcal{Q}} \;=\; \leq_{b,D,\mathcal{Q}}$.*

Proof. "$\subseteq$": Let $s_1 \preceq_{D,\mathcal{Q}} s_2$. We use Theorem A.11 and show $s_1 \leq_{b,D,\mathcal{Q},i} s_2$ for all i by induction on i. The case $i = 0$ obviously holds. Let $i > 0$ and let $s_1{\downarrow_D} v_1$. Then $s_1 \preceq_{D,\mathcal{Q}} s_2$ implies $s_2{\downarrow_D} v_2$. Thus, it is sufficient to show that $Q(v_1) \leq_{b,D,\mathcal{Q},i-1} Q(v_2)$ for all $Q \in \mathcal{Q}$: As induction hypothesis we use that $s_1 \preceq_{D,\mathcal{Q}} s_2 \implies s_1 \leq_{b,D,\mathcal{Q},i-1} s_2$ holds. Using Lemma A.12 twice and $s_1 \preceq_{D,\mathcal{Q}} s_2$, we have $Q(v_1) \preceq_{D,\mathcal{Q}} Q(v_2)$. The induction hypothesis shows that $Q(v_1) \leq_{b,D,\mathcal{Q},i-1} Q(v_2)$. Thus the definition of $\leq_{b,D,\mathcal{Q},i}$ is satisfied.

"$\supseteq$": Let $s_1 \leq_{b,D,\mathcal{Q}} s_2$. By induction on the number n of observers we show $\forall n, Q_i \in \mathcal{Q} : Q_1(\ldots(Q_n(s_1))){\downarrow_D} \implies Q_1(\ldots(Q_n(s_2))){\downarrow_D}$. The base case follows from $s_1 \leq_{b,D,\mathcal{Q}} s_2$. For the induction step we use the induction hypothesis: $t_1 \leq_{b,D,\mathcal{Q}} t_2 \implies \forall j < n, Q_i \in \mathcal{Q} : Q_1(\ldots(Q_j(t_1))){\downarrow_D} \implies Q_1(\ldots(Q_j(t_2))){\downarrow_D}$ for all t_1, t_2 of the same type. Let $Q_1(\ldots(Q_n(s_1))){\downarrow_D}$. From Lemma A.13 we have $r_1 \leq_{b,D,\mathcal{Q}} r_2$, where $r_i = Q_n(s_i)$. Now the induction hypothesis shows that $Q_1(\ldots(Q_{n-1}(r_1))){\downarrow_D} \implies Q_1(\ldots(Q_{n-1}(r_2))){\downarrow_D}$ and thus $Q_1(\ldots(Q_n(s_2))){\downarrow_D}$.

We now instantiate the experiment set $\mathcal{Q}$ of $\mathcal{Q}$-similarity (Definition A.2) with the usual contexts (application for functions and projection for constructor applications):

Definition A.15. *The set of experiments $\mathfrak{A}(D)$ for $D \in \{\mathsf{AL_{cc}}, \mathsf{AL_{cc,seq}}\}$ is defined as*

$$\begin{aligned}\mathfrak{A}(D) := & \{[\cdot]\ r \mid r \textit{ is a closed } D\textit{-expression}\} \\ & \cup \{\mathtt{case}_{K_i}\ [\cdot]\ \ldots (c_{K_i,j}\ x_1 \ldots x_{ar(c_{K_i,j})})\ \texttt{->}\ x_k \ldots \mid \textit{for all } i,j,k\} \\ & \cup \{\mathtt{case}_{K_i}\ [\cdot]\ \ldots (c_{K_i,j}\ x_1 \ldots x_{ar(c_{K_i,j})})\ \texttt{->}\ \mathtt{True} \ldots \mid \textit{for all } i,j\}.\end{aligned}$$

For $D \in \{\mathsf{AL}, \mathsf{AL_{seq}}\}$ let $\mathfrak{A}(D) := \{([\cdot]\ r) \mid r$ is a closed D-expression$\}$.

Howe's method ([How89,How96]) can be applied to all four calculi showing that contextual preorder and $\mathfrak{A}(D)$-similarity coincide, since these are call-by-name calculi. For the calculi with `case` and constructors the proofs are not completely straightforward, since in these calculi the "irregularity" that $\lambda x.\bot$ is contextually smaller than $c\ t_1\ \ldots\ t_n$ for any constructor c has to be taken into account.

However, we omit the proofs (for AL a proof can be found in [Abr90], for $\mathsf{AL_{cc,seq}}$ a worked out proof can be found in [SSSM12]).

Proposition A.16. *For every $D \in \{\mathsf{AL}, \mathsf{AL_{seq}}, \mathsf{AL_{cc}}, \mathsf{AL_{cc,seq}}\}$: $\leq_D \;=\; \leq^o_{b,D,\mathfrak{A}(D)} \;=\; \preceq^o_{D,\mathfrak{A}(D)}$*

Proof. Coincidence of contextual preorder and $\mathfrak{A}(D)$-similarity can be shown by Howe's method. Inspecting the normal order reduction in all four calculi shows that all the calculi are convergence-admissible which makes Theorem A.14 applicable and thus implies that finite simulation coincides with contextual preorder.

B On the Contextual Equivalence in the Untyped Calculi

B.1 Correctness of Reductions and Simplifications

Theorem B.1. *For $D \in \{\mathsf{AL}, \mathsf{AL_{seq}}, \mathsf{AL_{cc}}, \mathsf{AL_{cc,seq}}\}$ the reductions of the corresponding calculus (β-, seq-, case-reduction) are correct program transformations.*

Proof. Lemma A.4 shows that the reductions on closed expressions preserve $\mathfrak{A}(D)$-bisimilarity. For open expressions it suffices to observe that $s \to_D t$ also implies $\sigma(s) \to_D \sigma(t)$ for any closing substitution σ. This shows $\to_D \subseteq \leq^o_{b,D,\mathfrak{A}(D)}$ and thus Proposition A.16 shows the claim.

Lemma B.2. *For $D \in \{\mathsf{AL}, \mathsf{AL_{seq}}, \mathsf{AL_{cc}}, \mathsf{AL_{cc,seq}}\}$ and open D-expressions s, t with $FV(s) \cup FV(t) \subseteq \{x_1, \ldots, x_n\}$ the equivalence $s \sim_D t \iff \lambda x_1, \ldots, x_n.s \sim_D \lambda x_1, \ldots, x_n.t$ holds.*

Proof. One direction holds, since $\sim_D$ is a congruence. For the other direction, let $\lambda x_1,\ldots,x_n.s \sim_D \lambda x_1,\ldots,x_n.t$ hold. The congruence property implies that $(\lambda x_1,\ldots,x_n.s)\ r_1\ \ldots\ r_n \sim_D (\lambda x_1,\ldots,x_n.t)\ r_1\ \ldots\ r_n$ holds for expressions r_i. Correctness of β-reduction implies $s[r_1/x_1,\ldots,r_n/x_n] \sim_D t[r_1/x_1,\ldots,r_n/x_n]$. Proposition A.16 implies $s[r_1/x_1,\ldots,r_n/x_n] \approx_{\mathfrak{A}(D),\mathcal{Q}} t[r_1/x_1,\ldots,r_n/x_n]$ and thus also $\sigma(s) \approx_{\mathfrak{A}(D),\mathcal{Q}} \sigma(t)$ for any closing substitution σ, and thus $s \approx^o_{\mathfrak{A}(D),\mathcal{Q}} t$. Proposition A.16 now shows $s \sim_D t$.

Lemma B.3. *The reductions of Fig. 2 are correct in* $\{\mathsf{AL}_{\mathsf{seq}},\mathsf{AL}_{\mathsf{cc}},\mathsf{AL}_{\mathsf{cc,seq}}\}$.

Proof. Let $D \in \{\mathsf{AL}_{\mathsf{seq}},\mathsf{AL}_{\mathsf{cc}},\mathsf{AL}_{\mathsf{cc,seq}}\}$ as applicable. For all considered transformations it is easy to observe that if $s_1 \to s_2$ by a transformation, where s_1, s_2 are closed, then either both expressions s_1, s_2 diverge, or their evaluation ends in the same answer. This shows that s_1 and s_2 are bisimilar. Now it is also easy to verify that if the expressions s_1, s_2 are open, then $\sigma(s_1) \to \sigma(s_2)$ for a closing substitution σ is still contained in the transformation. Thus $s_1 \sim^o_{b,D,\mathfrak{A}(D)} s_2$. Finally, Proposition A.16 shows correctness of the transformations.

B.2 The AP_i-Context Lemma

Lemma B.4. *Let* $D \in \{\mathsf{AL},\mathsf{AL}_{\mathsf{seq}},\mathsf{AL}_{\mathsf{cc}},\mathsf{AL}_{\mathsf{cc,seq}}\}$ *and let R be a D-reduction context and s be a D-expression. Then one of the following equivalences hold:*

1. $R[s] \sim_D A[s]$ *for some applicative context* A
2. $R[s] \sim_D \mathtt{seq}\ A[s]\ t$ *for some applicative context A (for* $D \in \{\mathsf{AL}_{\mathsf{seq}},\mathsf{AL}_{\mathsf{cc,seq}}\}$*)*
3. $R[s] \sim_D \mathtt{case}_K\ A[s]\ alts$ *for some applicative context A (for* $D \in \{\mathsf{AL}_{\mathsf{cc}},\mathsf{AL}_{\mathsf{cc,seq}}\}$*)*

Proof. We only consider $\mathsf{AL}_{\mathsf{cc,seq}}$, for the other calculi the proof is analogous. The simplifications terminate (the measure $mcss(\cdot)$ introduced in Lemma D.3 can also be used for the untyped case) and are correct in the respective calculi (Lemma 3.6). Hence, w.l.o.g. the expressions are in normal form w.r.t. the simplifications. We consider the path to the hole of R: If there are only applications, then the lemma holds (item 1). If a $\mathtt{seq}\ [\cdot]\ r$ and applications are on the path, but no $\mathtt{case}\ [\cdot]\ alts$, then the normal form is described in item 2, since other forms are reducible. If a $\mathtt{case}\ [\cdot]\ r$ and applications (or perhaps a $\mathtt{seq}$) are on the path, then the normal form has a single $\mathtt{case}$ at the top, and the $\mathtt{seq}$ cannot occur on the path of the reduced expression, hence the context is described in item 3.

Proposition B.5. *Let* $D \in \{\mathsf{AL},\mathsf{AL}_{\mathsf{seq}},\mathsf{AL}_{\mathsf{cc}},\mathsf{AL}_{\mathsf{cc,seq}}\}$. *For every (perhaps open) D-expression s one of the following is true: $s \sim_D \bot$; $s \sim_D v$ where v is an answer; $s \sim_D A[x]$ where x is a free variable and A is an applicative D-context; $s \sim_D \mathtt{seq}\ A[x]\ t'$ where x is a free variable and A is an applicative D-context, for* $D \in \{\mathsf{AL}_{\mathsf{seq}},\mathsf{AL}_{\mathsf{cc,seq}}\}$*; or $s \sim_D \mathtt{case}_K\ A[x]\ t'$ where x is a free variable and A is an applicative D-context, for* $D \in \{\mathsf{AL}_{\mathsf{cc}},\mathsf{AL}_{\mathsf{cc,seq}}\}$.

Proof. This follows from the correctness of normal order reductions and the following observations: For an AL-expression s either $s \sim_{\mathsf{AL}} \bot$, or $s{\downarrow_{\mathsf{AL}}}\lambda x.s'$, or $s \xrightarrow{*}_{\mathsf{AL}} x\ t_1\ \ldots\ t_n$. For an $\mathsf{AL}_{\mathsf{seq}}$-expression s either $\sim_{\mathsf{AL}_{\mathsf{seq}}} \bot$, or $s{\downarrow_{\mathsf{AL}_{\mathsf{seq}}}}\lambda x.s'$, or $s \xrightarrow{*}_{\mathsf{AL}_{\mathsf{seq}}} R[x]$. For the last case Lemma B.4 shows $s \sim_{\mathsf{AL}_{\mathsf{seq}}} A[x]$ or $s \sim_{\mathsf{AL}_{\mathsf{seq}}} \mathtt{seq}\ A[x]\ t$. For an $\mathsf{AL}_{\mathsf{cc}}$-expression s either $s \sim_{\mathsf{AL}_{\mathsf{cc}}} \bot$, or $s{\downarrow_{\mathsf{AL}_{\mathsf{cc}}}}\lambda x.s'$, or $s{\downarrow_{\mathsf{AL}_{\mathsf{cc}}}}(c\ t_1\ \ldots t_n)$ or $s \xrightarrow{*}_{\mathsf{AL}_{\mathsf{cc}}} R[\mathtt{case}_K\ \lambda x.s'\ \ldots] \sim_{\mathsf{AL}_{\mathsf{cc}}} \bot$, or $s \xrightarrow{*}_{\mathsf{AL}_{\mathsf{cc}}} R[\mathtt{case}_{K,i}\ (c_{K_j,k}\ t_1 \ldots t_n)\ \ldots]$ where $T_j \neq T_i$ and thus $s \sim_{\mathsf{AL}_{\mathsf{cc}}} \bot$, $s \xrightarrow{*}_{\mathsf{AL}_{\mathsf{cc}}} R[(c\ t_1 \ldots t_{ar(c)})\ s] \sim_{\mathsf{AL}_{\mathsf{cc}}} \bot$, or $s \xrightarrow{*}_{\mathsf{AL}_{\mathsf{cc}}} R[x]$, and Lemma B.4 then shows $s \sim_{\mathsf{AL}_{\mathsf{cc}}} A[x]$, or $s \sim_{\mathsf{AL}_{\mathsf{cc}}} \mathtt{case}_K\ A[x]\ alts$. For $\mathsf{AL}_{\mathsf{cc,seq}}$ the cases are analogous where for the case $s \to_{\mathsf{AL}_{\mathsf{cc,seq}}} R[x]$ again Lemma B.4 shows the claim.

Corollary B.6. *For* $D \in \{\mathsf{AL},\mathsf{AL}_{\mathsf{seq}},\mathsf{AL}_{\mathsf{cc}},\mathsf{AL}_{\mathsf{cc,seq}}\}$ *and any closed D-expression s either $s \sim_D \bot$, or $s \sim_D \lambda x.t$, or $s \sim_D (c_{K_i,j}\ t_1\ \ldots\ t_{ar(c_{K_i,j})})$.*

Definition B.7. *Let* $\mathfrak{A}_V(D) = \{[\cdot]\ r \mid r$ *is a closed D-abstraction, or* $\bot\}$ *for* $D \in \{\mathsf{AL},\mathsf{AL}_{\mathsf{seq}}\}$. *For* $D \in \{\mathsf{AL}_{\mathsf{cc}},\mathsf{AL}_{\mathsf{cc,seq}}\}$ *we define the experiments* $\mathfrak{A}_V(D) = \{([\cdot]\ r) \mid r$ *is a closed D-abstraction, a constructor application, or* $\bot\} \cup \{\mathtt{case}_{K_i}\ [\cdot]\ \ldots (c_{K_i,j}\ x_1 \ldots x_{ar(c_{K_i,j})})\ \mathtt{->}\ x_k \ldots \mid$ *for all* $i,j,k\}$

Proposition B.8. $\preceq_{D,\mathfrak{A}_V(D)} = \preceq_{D,\mathfrak{A}(D)}$ *for* $D \in \{\mathsf{AL},\mathsf{AL}_{\mathsf{seq}},\mathsf{AL}_{\mathsf{cc}},\mathsf{AL}_{\mathsf{cc,seq}}\}$.

Proof. One direction is trivial. For the other direction, Corollary B.6 shows that every expression of the form $Q_1(Q_2(\ldots(Q_n(s))\ldots))$ where all $Q_i \in \mathfrak{A}(D)$ is contextually equivalent to an expression $Q'_1(Q'_2(\ldots(Q'_n(s))\ldots))$ where all $Q'_i \in \mathfrak{A}_V(D)$.

$$\frac{}{x^T :: T} \qquad \frac{}{\texttt{Bot} :: T} \qquad \frac{}{\texttt{Fix} :: (T \to T) \to T}$$

$$\frac{s :: T_2}{(\lambda x^{T_1}.s) :: T_1 \to T_2} \quad \frac{s :: T_1 \to T_2, t :: T_1}{(s\ t) :: T_2} \quad \frac{s :: T_1, t :: T_2}{(\texttt{seq}\ s\ t) :: T_2}$$

$$\frac{s :: (K_i\ T'_1\ \dots\ T'_{ar(K_i)}), \qquad \text{for } i = 1, \dots, m : (c_{K_i,k}\ x_{k,1} \dots x_{k,n_k}) :: (K_i\ T'_1\ \dots\ T'_{ar(K_i)}) \text{ and } t_i :: T}{(\texttt{case}_{K_i}\ s\ (c_{K_i,1}\ x_{1,1} \dots x_{1,n_1}\ \texttt{->}\ t_1) \dots (c_{K_i,m}\ x_{m,1} \dots x_{m,n_m}\ \texttt{->}\ t_m) :: T)}$$

$$\frac{\begin{array}{c} c_{K_i} \text{ has type-schema } T_1 \to \dots \to T_n \to (K_j\ T'_1\ \dots\ T'_{ar(K_j)}), \\ \text{there is an instantiation } \sigma \text{ such that for } j = 1, \dots, n{:}\sigma(T_j) = T''_j \text{ and } s_j :: T''_j, \end{array}}{(c_{K_i}\ s_1\ \dots\ s_n) :: \sigma(K_j\ T'_1, \dots, T'_{ar(K_j)})}$$

Fig. 5. Monomorphic Typing Rules

Theorem B.9. *For $D \in \{\mathsf{AL}, \mathsf{AL}_{\mathsf{seq}}, \mathsf{AL}_{\mathsf{cc}}, \mathsf{AL}_{\mathsf{cc,seq}}\}$ we have $\leq_D = \preceq^o_{D,\mathfrak{A}_V(D)} = \leq^o_{b,D,\mathfrak{A}_V(D)}$. Moreover, $s \leq_D t$ iff for all i and AP_i-contexts: $AP_i[s]{\downarrow_D} \implies AP_i[t]{\downarrow_D}$.*

Proof. Normal order reduction in the calculi satisfies that D is convergence-admissible w.r.t. $\mathfrak{A}_V(D)$. Now Propositions B.8, A.16 and Theorem A.14 show the claim.

Theorem B.9 and Proposition A.6 imply:

Corollary B.10. *For $D \in \{\mathsf{AL}, \mathsf{AL}_{\mathsf{seq}}, \mathsf{AL}_{\mathsf{cc}}, \mathsf{AL}_{\mathsf{cc,seq}}\}$ closed D-expressions s and t are contextually equivalent if there exists $i \in \mathbb{N}_0$ such that*

1. *$s{\downarrow} \iff t{\downarrow}$.*
2. *For all $1 < k \leq i$, and for all $Q_k, \dots, Q_i$ with $Q_j \in \mathfrak{A}_V(D)$ (for $k \leq j \leq i$): $Q_k(Q_{k+1}(\dots(Q_i(s)\dots))){\downarrow_D} \iff Q_k(Q_{k+1}(\dots(Q_i(t)\dots))){\downarrow_D}$.*
3. *For all $Q_1, \dots, Q_i$ with $Q_j \in \mathcal{Q}_{T_j,T_{j-1}}$ (for $1 \leq j \leq i$): $Q_1(Q_2(\dots(Q_i(s)\dots))) \sim_D Q_1(Q_2(\dots(Q_i(t)\dots)))$ holds.*

Since $\mathfrak{A}_V(D)$ coincides with AP_1-contexts, Corollary B.10 implies:

Theorem B.11. *For $D \in \{\mathsf{AL}, \mathsf{AL}_{\mathsf{seq}}, \mathsf{AL}_{\mathsf{cc}}, \mathsf{AL}_{\mathsf{cc,seq}}\}$ closed D-expressions s and t are contextually equivalent if there exists $i \in \mathbb{N}_0$ such that*

1. *$AP_j[s]{\downarrow_D} \iff AP_j[t]{\downarrow_D}$ for all $0 \leq j < i$ and all AP_j-contexts.*
2. *$AP_i[s] \sim_D AP_i[t]$ for all AP_i-contexts.*

C On the Typed Calculi

The typing rules for $\mathsf{AL}^\tau, \mathsf{AL}^\tau_{\mathsf{cc}}, \mathsf{AL}^\tau_{\mathsf{seq}}, \mathsf{AL}^\tau_{\mathsf{cc,seq}}$ are in Fig. 5. The typing judgments do not contain a typing environment, since the types of variables are built in. Howe's method can be used to show that contextual equivalence and applicative bisimilarity coincide. All of the calculi are convergence-admissible, and thus finite simulation also coincides with contextual equivalence. As experiment set we adjust the contexts $\mathfrak{A}_V(D)$ (Definition B.7) to the typed calculi: Applications and case-expressions are only allowed if they are correctly typed, and $\perp$ is replaced by `Bot`. Analogously to Theorem B.9, we have:

Theorem C.1. *For $D \in \{\mathsf{AL}^\tau, \mathsf{AL}^\tau_{\mathsf{seq}}, \mathsf{AL}^\tau_{\mathsf{cc}}, \mathsf{AL}^\tau_{\mathsf{cc,seq}}\}$ we have:*

1. *$\leq_D = \leq^o_{b,\mathfrak{A}_V(D),\mathcal{Q}} = \preceq^o_{\mathfrak{A}_V(D),\mathcal{Q}}$*
2. *For closed, equally typed D-expressions s, t holds: $s \leq_D t$ iff for all i and all AP_i-contexts, s.t. $AP_i[s]$ and $AP_i[t]$ are well-typed: $AP_i[s]{\downarrow_D} \implies AP_i[t]{\downarrow_D}$.*

Also a variant of Corollary B.10 obviously holds in these calculi:

Lemma C.2. *For $D \in \{\mathsf{AL}^\tau, \mathsf{AL}^\tau_{\mathsf{seq}}, \mathsf{AL}^\tau_{\mathsf{cc}}, \mathsf{AL}^\tau_{\mathsf{cc,seq}}\}$ equally typed closed D-expressions s, t are contextually equivalent if there exists $i \in \mathbb{N}_0$ such that*

1. *$s{\downarrow_D} \iff t{\downarrow_D}$.*
2. *For all $1 < k \leq i$, and for all $Q_k, \dots, Q_i$ with $Q_j \in \mathfrak{A}_V(D)$ (for $k \leq j \leq i$): $Q_k(Q_{k+1}(\dots(Q_i(s)\dots))){\downarrow_D} \iff Q_k(Q_{k+1}(\dots(Q_i(t)\dots))){\downarrow_D}$*

3. For all $Q_1,\ldots,Q_i$ with $Q_j \in \mathcal{Q}_{T_j,T_{j-1}}$ (for $1 \leq j \leq i$): $Q_1(Q_2(\ldots(Q_i(s)\ldots))) \sim_D Q_1(Q_2(\ldots(Q_i(t)\ldots)))$ holds.

This implies the following theorem:

Theorem C.3. *For $D \in \{\mathsf{AL}^\tau, \mathsf{AL}^\tau_{\mathsf{seq}}, \mathsf{AL}^\tau_{\mathsf{cc}}, \mathsf{AL}^\tau_{\mathsf{cc,seq}}\}$ closed, equally typed D-expressions s and t are contextually equivalent if there exists $i \in \mathbb{N}_0$ such that $AP_i[s] \sim_D AP_i[t]$ for all AP_i-contexts, and $AP_j[s]{\downarrow_D} \iff AP_j[t]{\downarrow_D}$ for all $0 \leq j < i$ and all AP_j-contexts.*

We show correctness of transformation in the typed calculi. First we define an untyped calculus $\mathsf{AL}_{\mathsf{cc,seq,FB}}$ as $\mathsf{AL}_{\mathsf{cc,seq}}$ with two additional constants: `Fix` with the reduction rule as defined in Fig. 1, and `Bot` as a constant that stands for any diverging expression.

Lemma C.4. *Let $\delta_1 : \mathsf{AL}_{\mathsf{cc,seq,FB}} \to \mathsf{AL}_{\mathsf{cc,seq}}$ be the translation defined as the identity applied homomorphically over the term structure except for the cases: $\delta_1(\mathtt{Fix}) = \lambda f.(\lambda x.(f\ x\ x))(\lambda x.(f\ x\ x))$ and $\delta_1(\mathtt{Bot}) = \Omega$. δ_1 is fully-abstract, i.e. for all $\mathsf{AL}_{\mathsf{cc,seq,FB}}$-expressions s,t: $s \sim_{\mathsf{AL}_{\mathsf{cc,seq,FB}}} t \iff \delta_1(s) \sim_{\mathsf{AL}_{\mathsf{cc,seq}}} \delta_1(t)$.*

Proof. The translations $\delta_1(\mathtt{Fix})$ and $\delta_1(\mathtt{Bot})$ simulate the behavior of these constants in $\mathsf{AL}_{\mathsf{cc,seq}}$ so that $s{\downarrow_{\mathsf{AL}_{\mathsf{cc,seq,FB}}}} \iff \delta_1(s){\downarrow_{\mathsf{AL}_{\mathsf{cc,seq}}}}$, i.e. δ_1 is convergence equivalent. Since δ_1 is compositional and the embedding from $\mathsf{AL}_{\mathsf{cc,seq}}$ into $\mathsf{AL}_{\mathsf{cc,seq,FB}}$ is also convergence equivalent, full abstractness follows.

Corollary C.5. *δ, as defined in Definition 5.2, is adequate.*

Proof. Assume $\delta(s) \sim_{\mathsf{AL}_{\mathsf{cc,seq}}} \delta(t)$ for some equally typed $\mathsf{AL}^\tau_{\mathsf{cc,seq}}$-expressions s,t. Then Lemma C.4 shows that $s' \sim_{\mathsf{AL}_{\mathsf{cc,seq,FB}}} t'$ where s',t' are s,t without types. Then also $s \sim_{\mathsf{AL}^\tau_{\mathsf{cc,seq}}} t$, since there are fewer contexts in $\mathsf{AL}^\tau_{\mathsf{cc,seq}}$ than in $\mathsf{AL}_{\mathsf{cc,seq,FB}}$.

Remark C.6. δ is not fully abstract, e.g. the typed equation $\lambda x^o.x \sim_{\mathsf{AL}^\tau_{\mathsf{cc,seq}}} \lambda x^o.\mathtt{Bot}$ holds, but $\lambda x.x \not\sim_{\mathsf{AL}_{\mathsf{cc,seq,FB}}} \lambda x.\mathtt{Bot}$, as well as $\lambda x.x \not\sim_{\mathsf{AL}_{\mathsf{cc,seq}}} \lambda x.\Omega$.

Lemma C.7. *Bot-reductions in Fig. 4 are correct in $\mathsf{AL}_{\mathsf{cc,seq,FB}}$ and in $\mathsf{AL}^\tau_{\mathsf{cc,seq}}$.*

Proof. For $\mathsf{AL}_{\mathsf{cc,seq,FB}}$ we use full-abstractness of δ_1, and thus it is sufficient to show that $\delta_1(s) \sim_{\mathsf{AL}_{\mathsf{cc,seq}}} \delta_1(t)$ where $s \to t$ is a bot-reduction. In $\mathsf{AL}_{\mathsf{cc,seq}}$ we use bisimilarity (see Theorem B.9). Consider Ω, $(\Omega\ s)$, $\mathtt{seq}\ \Omega\ t$, $\mathtt{case}_K\ \Omega\ t$. All expressions diverge and thus are bisimilar and hence contextually equivalent in $\mathsf{AL}_{\mathsf{cc,seq}}$.

Since Bot-reductions are type preserving and correct in the untyped $\mathsf{AL}_{\mathsf{cc,seq,FB}}$ where more contexts are used in the definition of contextual equivalence than in $\mathsf{AL}^\tau_{\mathsf{cc,seq}}$, the Bot-reductions are also correct in $\mathsf{AL}^\tau_{\mathsf{cc,seq}}$.

Note that since `Bot` stands for a family of constants (one for every type), the type of `Bot` on the left-hand-side of a bot-reduction may not be the same as that on the right-hand-side. For instance, in $(\mathtt{Bot}\ s) \to \mathtt{Bot}$, assuming s_2 is of type τ_1, the first occurrence of `Bot` is of a type $\tau_1 \to \tau_2$, and the second occurrence is of type τ_2.

Lemma C.8. *$R[\mathtt{Bot}] \sim_D \mathtt{Bot}$, where D ranges over $\mathsf{AL}_{\mathsf{cc,seq,FB}}$ and the four monomorphically typed calculi, and R is a D-reduction context.*

Proof. In $\mathsf{AL}_{\mathsf{cc,seq,FB}}$ the equivalence follows from Lemma C.7 and an induction on the size of R. The proof for $\mathsf{AL}^\tau_{\mathsf{cc,seq}}$ can be obtained analogously by assuming that all of the expressions are typed and using Lemma C.7 for the correctness of `Bot`-simplifications. For the other three typed calculi this follows, since there are fewer contexts.

We also observe that (case) and (seq) simplifications are correct in $\mathsf{AL}^\tau_{\mathsf{cc,seq}}$:

Lemma C.9. *The simplifications defined in Fig. 2 are correct in $\mathsf{AL}^\tau_{\mathsf{cc,seq}}$.*

Proof. Since δ is adequate (Corollary C.5), it suffices to observe that for any simplification rule $s \to t$ in $\mathsf{AL}^\tau_{\mathsf{cc,seq}}$, the equivalence $\delta(s) \sim_{\mathsf{AL}_{\mathsf{cc,seq}}} \delta(t)$ holds, which is proven by Lemma B.3.

We extend the addition of `Fix` and `Bot` to the untyped calculi $\mathsf{AL}, \mathsf{AL}_{\mathsf{seq}}, \mathsf{AL}_{\mathsf{cc}}$ in a completely analogous way:

Lemma C.10. *(botapp), (botcase), (botseq), (caseapp), (casecase), (botseq), (seqseq), (seqapp), (caseseq), (seqcase) are correct in $\mathsf{AL}^\tau, \mathsf{AL}^\tau_{\mathsf{seq}}, \mathsf{AL}^\tau_{\mathsf{cc}}, \mathsf{AL}^\tau_{\mathsf{cc,seq}}$ (if the constructs are defined).*

Proof. The results for $\mathsf{AL}^{\tau}_{\mathsf{cc,seq}}$ are proven in Lemmas C.7 and C.9. The results for the other calculi obviously hold, since they have a restricted syntax (compared to $\mathsf{AL}^{\tau}_{\mathsf{cc,seq}}$) and since there are fewer contexts in their contextual equivalences.

The following properties are straightforward to verify:

Lemma C.11. *1. Any expression of the type $(o \to o) \to (o \to o)$ is equivalent to one of:* $\mathtt{Bot}$, $\lambda x.\mathtt{Bot}$, $\lambda x, y.\mathtt{Bot}$, *and* $\lambda x.x$.
2. Any closed expression of the type $(o \to o) \to (o \to o) \to (o \to o)$ is equivalent to one of: $\mathtt{Bot}$, $\lambda x.\mathtt{Bot}$, $\lambda x, y.\mathtt{Bot}$ $\lambda x, y, z.\mathtt{Bot}$, $\lambda x, y.x$, *and* $\lambda x, y.y$.

Proof. We use applicative bisimulation for the proof, and that there is only one equivalence class, namely $\mathtt{Bot}$, of type o.
We argue for the second item: the cases for the syntax of the body are restricted. For example $\lambda x, y.r$: r may be $\mathtt{Bot}$, an abstraction, x,y or an application with a variable as head. An abstraction can only be equivalent to $\lambda z.\mathtt{Bot}$, and an application can only be of type o, which also can only be equivalent to $\mathtt{Bot}$.

D Reduction Length for Simplifications

We consider simplifications defined in Figs. 2 and 4 in arbitrary contexts, and analyze their interactions with normal order reductions in $\mathsf{AL}_{\mathsf{cc,seq,FB}}$.

Since the diagrams given below hold in the untyped calculus, they also hold in its typed version $\mathsf{AL}^{\tau}_{\mathsf{cc,seq}}$ under the assumption that the initial expression is typable, since all reductions and simplifications preserve types. There are two kinds of the diagrams: *commuting* and *forking* (see [KSS98,SSSS08]) In both kinds of diagrams a simplification step (denoted by $\xrightarrow{a}$) and a normal order step (denoted by $\xrightarrow{b}_{\mathsf{AL}_{\mathsf{cc,seq,FB}}}$) are given, and both diagrams show how the given pair can be replaced by another sequence connecting the given expressions. The two kinds of diagrams differ by directions of the given steps. In a commuting diagram a given pair of reduction steps has the form $s_1 \xrightarrow{a} s_2 \xrightarrow{b}_{\mathsf{AL}_{\mathsf{cc,seq,FB}}} s_3$, and the diagram is completed by stating that there exists a sequence of steps from s_1 to s_3. In a forking diagram the given sequence is of the form $s_1 \xleftarrow{a} s_2 \xrightarrow{b}_{\mathsf{AL}_{\mathsf{cc,seq,FB}}} s_3$, and the diagram is again completed by stating that there exists a sequence of steps from s_1 to s_3.

The graphical representation in Lemma D.1 may be interpreted as a commuting or a forking diagram, depending on which steps are assumed to be given and which ones are claimed to exist.

Lemma D.1. *If $\xrightarrow{a}$ is a simplification of Figs. 2 and 4 and $\xrightarrow{b}_{\mathsf{AL}_{\mathsf{cc,seq,FB}}}$ is a (β)-, (case)-, (seq)-, or (fix)-normal order reduction, then commuting and forking diagrams are as follows, where $\xrightarrow{a,*}$ means zero or more occurrences of a.*

$$\begin{tikzcd} \cdot \arrow[r, "a"] \arrow[d, "{b,\mathsf{AL}_{\mathsf{cc,seq,FB}}}"'] & \cdot \arrow[d, "{b,\mathsf{AL}_{\mathsf{cc,seq,FB}}}"] \\ \cdot \arrow[r, "{a,*}"] & \cdot \end{tikzcd}$$

Proof. There are several possibilities for the redex of the b-reduction:

1. a is located in one of the operands of b. E.g., if b is a β-reduction $((\lambda x.s_1)s_2)$, b can be located in s_1 or s_2. If it is located in s_1 then it does not change because the shape of a redex is not changed by a substitution into it. If it is located in s_2 then it can be duplicated, preserved, or eliminated, which creates 0 or more occurrences of a in the diagram. A similar situation happens when b is embedded in a $\mathtt{case}$ reduction (either in the scrutinee or in the alternatives), in $\mathtt{seq}$ (in this case b cannot be duplicated, only preserved or eliminated), and in (fix) (then it occurs in the result exactly twice).
2. The redex for a appears in a reduction context and contains the redex for b as its immediate (depth 1) subredex. For instance, in (seqseq) the expression may be of the form $R[\mathtt{seq}\ (\mathtt{seq}\ v\ s_1)\ s_2]$, where R is a reduction context and v is a value. Then the result of both the (seqseq) followed by a normal order reduction and the normal order (seq) result in the same expression $\mathtt{seq}\ s_1\ s_2$. The same situation occurs with (seqapp) and a normal order (seq), (seqcase) and a normal order (case), (caseseq) and a normal order (seq), and for both (caseapp) and (casecase) and a normal order (case). This case is impossible for (fix) (since its operand is $\mathtt{Fix}$ which does not match any cases for a).
3. The redex for a contains the redex for b, but not as in item 2. In this case a and b do not affect each other since the shape of the redex for a is not changed by b and the redex for b is moved, but not changed, by a and remains in a reduction context.

4. The two reductions have non-overlapping redexes. Then the diagram holds with exactly one instance of a in the bottom line.

Combining all cases, we obtain the diagram in the lemma.

Lemma D.2. *If $s \xrightarrow{a} t$, where a is one of the simplifications defined in Figures 2 and 4, then s is a WHNF if and only if t is a WHNF.*

Proof. In $\mathsf{AL}_{\mathsf{cc,seq,FB}}$ WHNFs are either abstractions or constructor applications. All auxiliary reductions have either `seq` or `case` or an application as their top-level construct, so a WHNF cannot be a source or a target of such a reduction.

Lemma D.3. *There is no infinite sequence in $\mathsf{AL}^{\tau}_{\mathsf{cc,seq}}$ consisting only of* `seq`*,* `case`*-, and* `Bot`*-simplifications.*

Proof. Let the term measure $css(s)$ be defined as: For $a = \mathtt{Bot}, a = \mathtt{Fix}, a = x$: $css(a) = 1$; $css(\mathtt{case}_T\ s\ (p_1 \texttt{->} r_1) \dots (p_n \texttt{->} r_n)) = 1 + 2css(s) + \max_{i=1,..,n}(css(r_i))$; $css(s\ t) = 1 + 2css(s) + 2css(t)$; $css(\mathtt{seq}\ s\ t) = 2css(s) + css(t)$; $css(c\ s_1 \dots s_n) = 1 + css(s_1) + \dots + css(s_n)$; and $css(\lambda x.s) = 1 + css(s)$. Every simplification rule strictly decreases css if applied at the top of an expression. We show the details for (caseapp) and (casecase), the remaining rules are analogous, but simpler.

– (caseapp):

$$\begin{aligned}
&1 + 2(1 + 2css(t_0) + \max_{i=1\dots n} css(t_i)) + 2css(r) = \\
&3 + 4css(t_0) + 2\max_{i=1\dots n} css(t_i) + 2css(r) > \\
&1 + 2css(t_0) + \max_{i=1\dots n}(1 + 2css(t_i) + 2css(r)) = \\
&2 + 2css(t_0) + 2\max_{i=1\dots n} css(t_i) + 2css(r)
\end{aligned}$$

– (casecase):

$$\begin{aligned}
&1 + 2(1 + 2css(t_0) + \max_{i=1\dots n} css(t_i)) + \max_{j=1\dots m} css(r_j) = \\
&3 + 4css(t_0) + 2\max_{i=1\dots n} css(t_i) + max_{j=1\dots m} css(r_j) \\
&1 + 2css(t_0) + \max_{i=1\dots n}(1 + 2css(t_i) + \max_{j=1\dots m} css(r_j)) = \\
&2 + 2css(t_0) + 2\max_{i=1\dots n} css(t_i) + \max_{j=1\dots m} css(r_j))
\end{aligned}$$

Note that for (casecase) $\max_{i=1\dots n} css(t_i)$ is independent from $\max_{j=1\dots m} css(r_j)$. If the simplification occurs in a non-maximal alternative of a `case`, the measure for that alternative decreases, but the measure for the entire expression remains unchanged. We use multisets as a measure to account for all case alternatives. I.e., as a measure for expressions s we use the multiset $mcss(.)$ consisting of the following numbers: $css(s)$ for the expression s; and all numbers $css(s')$, where s' is a case-alternative of a subterm of s. It is well-known [DM79] that multisets of non-negative integers are well-founded w.r.t. the multiset-ordering: $M_1 << M_2$, iff there exists multisets X_1, X_2 such that $\emptyset \neq X_2 \subseteq M_2$, $M_1 = (M_2 \setminus X_2) \cup X_1$, and $\forall x_1 \in X_1.\exists x_2 \in X_2.x_1 < x_2$. The measure of a subexpression is always strictly smaller than the measure for the expression itself. Every simplification rule results in one of the two options: i) decreasing the largest number in the multiset which is the measure for the entire expression. ii) preserving the measure for the entire expression if simplification takes place in a non-maximal case alternative. Then the measure for that alternative decreases, although the measure for its subalternatives may increase (both (caseapp) and (casecase) rules above allow for such a possibility). Since the decreased number is larger than the measure for the subalternatives, the multiset becomes smaller.

We summarize the results and transfer them to $\mathsf{AL}^{\tau}_{\mathsf{cc,seq}}$ and $\mathsf{AL}^{\tau}_{\mathsf{cc}}$:

Lemma D.4. *For $D \in \{\mathsf{AL}_{\mathsf{cc,seq,FB}}, \mathsf{AL}^{\tau}_{\mathsf{cc,seq}}, \mathsf{AL}^{\tau}_{\mathsf{seq}}, \mathsf{AL}^{\tau}_{\mathsf{cc}}\}$ if $s \xrightarrow{a} s'$ where a is a simplification rule then $s \xrightarrow{n}_D v$, where v is a WHNF, if and only if $s' \xrightarrow{n}_D v'$, where v' is a WHNF.*

Proof. The result in $\mathsf{AL}_{\mathsf{cc,seq,FB}}$ follows from Lemmas D.1, D.2, and D.3, where the proof for closed expressions is by induction on the length of a normal order reduction ending with a WHNF. The results trivially transfer to $\mathsf{AL}^{\tau}_{\mathsf{cc,seq}}$ by limiting the expressions to well-typed ones (and by type preservation of normal order reductions and simplifications) and to $\mathsf{AL}^{\tau}_{\mathsf{cc}}$ and $\mathsf{AL}^{\tau}_{\mathsf{seq}}$ by limiting our consideration only to constructs that exist in that calculus.

Lemma D.5. *For $D \in \{\mathsf{AL}_{\mathsf{cc,seq,FB}}, \mathsf{AL}^{\tau}_{\mathsf{cc,seq}}, \mathsf{AL}^{\tau}_{\mathsf{seq}}, \mathsf{AL}^{\tau}_{\mathsf{cc}}\}$ if $s \xrightarrow{bcsfC} s'$ and $s \xrightarrow{n}_D v$, where v is a D-WHNF, then $s' \xrightarrow{n'}_D v'$, where v' is a D-WHNF, with $n' \leq n$.*

Proof. We only need forking diagrams in all contexts between the non-normal order reduction and the normal order reduction. These are:

$$\begin{tikzcd} \cdot \arrow[r, "a"] \arrow[d, "b,D"'] & \cdot \arrow[d, dashed, "b,D"] \\ \cdot \arrow[r, dashed, "a,*"] & \cdot \end{tikzcd}$$

The claim of the lemma is proved by induction on the length of the normal order reduction. The base case obviously holds. If the reduction length is > 0, then there are possibilities: The $\xrightarrow{a}$-reduction is a normal order reduction. Since normal order reduction is deterministic, the reduction length of s' is strictly smaller than that of s. Otherwise, we use the diagram and can apply the induction hypothesis multiple times to the closing $\xrightarrow{a,*}$-reduction. The arguments are valid in all the mentioned calculi, hence the lemma holds.

E Nonisomorphism of AL and $\mathsf{AL}_{\mathsf{seq}}$

We argue that the four untyped calculi are nonisomorphic by analysing the order structure of the different calculi. It turns out that the order structure is fundamentally different for these calculi and distinguishes them up to the pair of calculi $\mathsf{AL}_{\mathsf{cc,seq}}$ and $\mathsf{AL}_{\mathsf{cc}}$, which we leave open, but where a thorough analysis presumably will result in non-equivalence.

Remember that we use $\bot$ as the abbreviation for a closed nonconverging expression, and $\top$ is the abbreviation for $(Y\ K)$, which for $D \in \{\mathsf{AL}, \mathsf{AL}_{\mathsf{seq}}\}$ is characterised by the following property: for all $n \geq 0$ and all closed expressions $v_1, \ldots, v_n$, $(\top\ v_1 \ldots v_n){\downarrow_D}$.

In the following we will write $s <_D t$ if $s \leq_D t$ holds but $t \leq_D s$ does not hold.

Lemma E.1. *In* AL*, the following holds:* $\bot <_{\mathsf{AL}} \lambda x.\bot$ *are the two smallest elements in* AL*. Also, there is no closed* AL*-epression* t *with* $\lambda x.\bot <_{\mathsf{AL}} t <_{\mathsf{AL}} \lambda x, y.\bot$*.*

Proof. Let $\bot <_{\mathsf{AL}} t \leq_{\mathsf{AL}} \lambda x.\bot$. Then $(t\ r) \sim_{\mathsf{AL}} \bot$ for all closed r, hence $t \sim_{\mathsf{AL}} \lambda x.\bot$.
For every abstraction t, the relation $\lambda x.\bot \leq_{\mathsf{AL}} t$ follows easily from applicative bisimulation.

Let $\lambda x.\bot <_{\mathsf{AL}} t \leq_{\mathsf{AL}} \lambda x.\lambda y.\bot$. Then $(t\ \top) \not\sim_{\mathsf{AL}} \bot$. Also, for all closed arguments r, we have $t\ r \leq_{\mathsf{AL}} \lambda y.\bot$, which implies $t\ r \sim_{\mathsf{AL}} \lambda y.\bot$, hence applicative bisimulation shows $t \sim_{\mathsf{AL}} \lambda x.\lambda y.\bot$.

Lemma E.2. *In* AL*, there is no direct descendent of* $\top$*: I.e.,* $t <_{\mathsf{AL}} \top$ *implies that there is a* t' *with* $t <_{\mathsf{AL}} t' <_{\mathsf{AL}} \top$*.*

Proof. In the case $t \sim_{\mathsf{AL}} \bot$, the expression $t' = \lambda x.\bot$ is sufficient.
If $t \sim_{\mathsf{AL}} \lambda x_1, \ldots x_n.\bot$, then $t' \sim_{\mathsf{AL}} \lambda x_1, \ldots x_n, x_{n+1}.\bot$ is the desired expression.

If $t \sim_{\mathsf{AL}} \lambda x_1, \ldots, x_n.x_i\ t_1 \ldots t_m$ for $m \geq 1$, then $t' = \lambda x_1, \ldots, x_n, y.x_i\ \underbrace{\top \ldots \top}_{m+1}$ is the desired expression: Applicative bisimulation shows that $t \leq_{\mathsf{AL}} t'$: if there are n arguments $r_1, \ldots, r_n$, then $t'\ r_1 \ldots r_n$ converges. For $n+1$ arguments, we have $r_i\ t'_1 \ldots t'_m\ r_{n+1} \leq_{\mathsf{AL}} r_i\ \underbrace{\top \ldots \top}_{m+1}$. For more arguments, the $\leq_{\mathsf{AL}}$-relation follows by induction. The expressions are not equivalent, by selecting $r_{n+1} \sim_{\mathsf{AL}} \bot$, and r_i as a projection to the $(m+1)^{st}$ argument. Similarly, $t' \not\sim_{\mathsf{AL}} \top$: Let r_i be the projection to the $(m+2)^{nd}$ argument, and let $r_{n+2} \sim_{\mathsf{AL}} \bot$.

Lemma E.3. *In* $\mathsf{AL}_{\mathsf{seq}}$*, the following holds:* $\lambda x.\mathtt{seq}\ x\ \top <_{\mathsf{AL}_{\mathsf{seq}}} \top$*, and there is no closed expression* t *with* $\lambda x.\mathtt{seq}\ x\ \top <_{\mathsf{AL}_{\mathsf{seq}}} t <_{\mathsf{AL}_{\mathsf{seq}}} \top$*,*

Proof. Assume there is a closed expression t with $\lambda x.\mathtt{seq}\ x\ \top <_{\mathsf{AL}_{\mathsf{seq}}} t <_{\mathsf{AL}_{\mathsf{seq}}} \top$. The $\mathsf{AL}_{\mathsf{seq}}$-WHNF of t is an abstraction, thus we can assume that $t = \lambda x.t'$. The body t' cannot be an abstraction, since then $t \sim_{\mathsf{AL}_{\mathsf{seq}}} \top$, since $t\ v_1 \ldots v_n{\downarrow_{\mathsf{AL}_{\mathsf{seq}}}}$ for any $n \geq 0$ and closed $\mathsf{AL}_{\mathsf{seq}}$-expressions v_i. The WHNF of the body t' of t has x in reduction position. There are several cases:

- It cannot be x alone, since then the $<_{\mathsf{AL}_{\mathsf{seq}}}$-relation does not hold.
- It can also not be $x\ t_1 \ldots t_n$, since then $x \mapsto \lambda u.\bot$ would refute this.
- Thus the body t' is a $\mathtt{seq}$-expression. Analysing all arguments: if $(t\ \bot){\downarrow_{\mathsf{AL}_{\mathsf{seq}}}}$, then the inequations imply that $t \sim_{\mathsf{AL}_{\mathsf{seq}}} \top$. Otherwise, if $(t\ \bot){\Uparrow_{\mathsf{AL}_{\mathsf{seq}}}}$, then for all sequences of arguments, $\lambda x.\mathtt{seq}\ x\ \top$ and t, behave the same, and thus they are equivalent.

Theorem E.4. AL *and* $\mathsf{AL}_{\mathsf{seq}}$ *are nonisomorphic.*

Proof. $\top$ is also a greatest element in $\mathsf{AL}_{\mathsf{seq}}$. If $\phi : \mathsf{AL} \to \mathsf{AL}_{\mathsf{seq}}$ is such an isomorphism, then $\phi(\top)$ must map to $\top$. However, there is no AL-expression t such that $\phi(t) = \lambda x.\mathtt{seq}\ x\ \top$, since for every $t \not\sim_{\mathsf{AL}} \top$, there is some AL-expression t' with $t <_{\mathsf{AL}} t' <_{\mathsf{AL}} \top$, but there is no such image $\phi(t')$ in $\mathsf{AL}_{\mathsf{seq}}$.

Proposition E.5. *All pairs from* $\{\mathsf{AL}, ALseq\} \times \{\mathsf{AL}_{\mathsf{cc}}, \mathsf{AL}_{\mathsf{cc,seq}}\}$ *are nonisomorphic.*

Proof. This holds, since AL and $\mathsf{AL}_{\mathsf{seq}}$ have $\top$ as a greatest elements, and neither $\mathsf{AL}_{\mathsf{cc}}$ not $\mathsf{AL}_{\mathsf{cc,seq}}$ have a greatest element.

This leaves the case $\mathsf{AL}_{\mathsf{cc}}$ and $\mathsf{AL}_{\mathsf{cc,seq}}$ open.