A Call-by-Need Lambda Calculus with Scoped Work Decorations

David Sabel[1] and Manfred Schmidt-Schauß[2]

Abstract: The polymorphically typed functional core language LRP is a lambda calculus with recursive `let`-expressions, data constructors, case-expressions, and a `seq`-operator that uses call-by-need evaluation. In this work LRP is extended by scoped work decorations to ease computations when reasoning on program improvements. The considered language LRPw extends LRP by two constructs to represent work (i.e. numbers of reduction steps) that can be shared between several subexpressions. Due to a surprising observation that this extension is proper, some effort is required to re-establish the correctness and optimization properties of a set of program transformations also in LRPw. Based on these results, correctness of several useful computation rules for work decorations is shown.

1 Introduction

Motivation. This paper is motivated by our recent investigations on program transformations and on the question whether these transformations are optimizations w.r.t. the runtime behavior: We analyzed such optimizations in core languages of lazy functional programming languages: The calculus LR [SSSS08] is an untyped call-by-need lambda calculus extended by data-constructors, `case`-expressions, `seq`-expressions, and `letrec`-expressions. This calculus e.g. models the (untyped) core language of Haskell. The calculus LRP is the polymorphically typed variant of LR [SSS15a], where typing in LRP is by `let`-polymorphism [Pi02, Pi00, VPJ13]. Polymorphism is made explicit in the syntax and there are also reduction rules for computing the specific types of functions.

In [SSS15b, SSS15a] a notion of improvement for program transformations was defined and analyzed for LR and LRP. Several results concerning different length measures and proving the improvement property for concrete transformation rules, like common subexpression elimination, were established. In [SSS15c] we considered improvements in LRP with a special focus on list-processing functions. To enable those proofs, a notion of shared work between several subexpression is very helpful, since it supports modular reasoning. While in [SSS15c] these work decorations were added in a sound but somehow ad-hoc manner, in this paper we provide a formal treatment and add a scoping for decorations.

Correct Program Transformations and Improvements. For reasoning on the correctness of program transformations, we use contextual equivalence as program equivalence: Contextual equivalence identifies two programs if exchanging one program by the other program

[1] Computer Science Institute, Goethe-University Frankfurt am Main, sabel@ki.informatik.uni-frankfurt.de
[2] Computer Science Institute, Goethe-University Frankfurt am Main, schauss@ki.informatik.uni-frankfurt.de

in any surrounding larger program (the context) is not observable. Due to the quantification over all contexts it is sufficient to only observe the termination behavior of the programs, since e.g. different values like `True` and `False` can be distinguished by plugging them into a context C s.t. $C[\texttt{True}]$ terminates while $C[\texttt{False}]$ diverges. A program transformation is correct if it preserves the semantics, i.e. it preserves contextual equivalence.

For reasoning whether program transformations are also optimizations, i.e. so-called *improvements*, we adopt the improvement theory originally invented by Moran and Sands [MS99] for an abstract machine semantics, but slightly modified and adapted it in [SSS15b, SSS15a] for the calculi LR and LRP. Roughly speaking, a program transformation is an improvement, if it is correct (i.e. preserves contextual equivalence) and the number of computation steps is never strictly increased after applying the transformation.

Scoped work decorations. In [MS99] a tick-algebra was introduced to prove correctness of improvement laws in a modular way. A tick $\checkmark^n$ can be attached to an expression to add a fixed amount of work to the expression (i.e. n execution steps). Several laws for computing with ticks are formulated and proved correct. In this paper we introduce the calculus LRPw which extends LRP in a similar way, where ticks are called decorations, but they are extended to a formalism that can express *work* which is *shared* between several subexpressions, which makes reasoning more comfortable and also more exact. In LRPw there are two new (compared to LRP) constructs: Bindings of the form $a{:=}n$ and decorations of the form $s^{[a]}$. Here $s^{[a]}$ means that the work expressed by the binding for a (i.e. n essential steps) has to be done before the expression s can be further evaluated. If decoration a occurs at several subexpressions, the work is shared between the subexpressions (and thus at most performed once). The bindings $a{:=}n$ occur in usual `letrec`-expressions and thus also define the scope of the sharing, and a notion of α-equivalence w.r.t. the labels a. This makes a rigorous formal treatment possible.

As a very simple example for the usefulness of shared-work decoration, consider the expression $s = (\texttt{let}\ x{=}(\lambda w.w)\ \texttt{True}\ \texttt{in}\ (x,x))$ and assume that we want to transform s into a value with work decorations (e.g. for further reasoning): The binding x is only evaluated if the first, the second, or both components of the pair are demanded by an outer context. Since we use call-by-need evaluation, in any of the three cases the binding for x is evaluated only once. This can be expressed by work decorations with $\texttt{let}\ a{:=}1\ \texttt{in}\ (\texttt{True}^{[a]}, \texttt{True}^{[a]})$.

Results. A surprising and counter-intuitive observation is that the extension LRPw cannot be encoded in LRP (see Proposition 4.8), and that the non-encodable expressions cannot be excluded, since these arise naturally when computing with the work decorations. This makes it necessary to explicitly consider the calculus LRPw and to reconsider claims and proofs of properties. Thus in this paper we show that known improvement laws for LRP also hold in LRPw, that a context lemma for improvement holds in LRPw and that several computation rules which simplify the reasoning with decorated expressions are invariant w.r.t. the improvement relation. The results of this paper allow to use shared work decorations as a reasoning tool, e.g. for proving improvement laws on list-processing expressions and functions (as in [SSS15c], but now with a formal semantic foundation).

Types: Types *Typ* and polymorphic types *PTyp* are generated by the grammar:

$$\begin{array}{ll} \tau \in \mathit{Typ} & ::= A \mid (\tau_1 \to \tau_2) \mid K\ \tau_1 \ldots \tau_{ar(K)} \\ \rho \in \mathit{PTyp} & ::= \tau \mid \lambda A.\rho \end{array}$$

Expressions: Expressions Expr_F, patterns $\mathit{pat}_{K,i}$, bindings Bind_i, and polymorphic abstractions PExpr_F are generated by the following grammar:

$$\begin{array}{ll} s,t \in \mathit{Expr}_F & ::= u \mid x :: \rho \mid (s\ \tau) \mid (s\ t) \mid (\texttt{letrec}\ \mathit{Bind}_1, \ldots, \mathit{Bind}_n\ \texttt{in}\ t) \mid (s^{[a]}) \mid (\texttt{seq}\ s\ t) \\ & \quad \mid (c_{K,i} :: \tau\ s_1\ \ldots\ s_{ar(c_{K,i})}) \mid (\texttt{case}_K\ s\ (\mathit{pat}_{K,1} \to t_1) \ldots (\mathit{pat}_{K,|D_K|} \to t_{|D_K|})) \\ \mathit{pat}_{K,i} & ::= (c_{K,i} :: \tau\ x_1 :: \tau_1 \ldots x_{ar(c_{K,i})} :: \tau_{ar(c_{K,i})}) \\ \mathit{Bind}_i & ::= x :: \rho = s \mid a := n \text{ where } a \text{ is a label, and } n \in \mathbb{N}_0 \\ u \in \mathit{PExpr}_F & ::= \Lambda A_1 \ldots . \Lambda A_k . \lambda x :: \tau . s \end{array}$$

Typing rules:

$$\frac{u :: \rho}{\Lambda A.u :: \lambda A.\rho} \qquad \frac{s :: \tau_1 \quad \mathit{pat}_i :: \tau_1 \quad t_i :: \tau_2}{(\texttt{case}_K\ s\ (\mathit{pat}_1 \to t_1) \ldots (\mathit{pat}_{|D_K|} \to t_{|D_K|})) :: \tau_2} \qquad \frac{s :: \tau \quad t :: \tau'}{(\texttt{seq}\ s\ t) :: \tau'} \qquad \frac{s :: \lambda A.\rho}{(s\ \tau) :: \rho[\tau/A]}$$

$$\frac{s :: \tau_1 \to \tau_2 \quad t :: \tau_1}{(s\ t) :: \tau_2} \qquad \frac{s_1 :: \rho_1 \quad \ldots \quad s_n :: \rho_n \quad t :: \rho}{(\texttt{letrec}\ a_1 := n_1, \ldots, a_m := n_m,\ x_1 :: \rho_1 = s_1, \ldots, x_n :: \rho_n = s_n\ \texttt{in}\ t) :: \rho}$$

$$\frac{s :: \tau_2}{(\lambda x :: \tau_1 . s) :: \tau_1 \to \tau_2} \qquad \frac{\begin{array}{c} s_1 :: \tau_1, \ldots, s_{ar(c)} :: \tau_{ar(c)} \qquad \tau = \tau_1 \to \ldots \to \tau_{ar(c)} \to \tau_{ar(c)+1} \\ \mathit{type}(c) = \lambda A_1, \ldots, A_m . \tau'' \qquad \exists \tau'_1, \ldots, \tau'_m : \tau''[\tau'_1/A_1, \ldots, \tau'_m/A_m] = \tau \end{array}}{(c :: \tau\ s_1 \ldots s_{ar(c)}) :: \tau_{ar(c)+1}}$$

Fig. 1: Syntax of expressions and types, and typing rules, where $A, A_i \in \mathit{TVar}$ denote type variables, $x, x_i \in \mathit{Var}$ denote term variables, and a, b, a_i, b_i are labels used for sharing work.

Outline. In Sect. 2 we introduce the calculi LRP and LRPw, and transfer the basic definitions and context lemmas from LRP to LRPw. In Sect. 3 we show that several reduction rules and program transformations are correct and are improvements. In Sect. 4 we show that LRP and LRPw are isomorphic w.r.t. contextual equivalence, and we investigate the relation between both calculi w.r.t. the improvement relation. In Sect. 5 we show that several computation rules for work decorations hold. We conclude in Sect. 6.

2 The Polymorphically Typed Lazy Lambda Calculus LRPw

We introduce the calculus LRPw. It is an extension of the polymorphically typed, extended call-by-need lambda calculus LRP [SSS15a, SS14] and its untyped variant LR [SSSS08].

2.1 Syntax and Operational Semantics of LRPw

Let $\mathscr{K}$ be a fixed set of type constructors, s.t. every $K \in \mathscr{K}$ has an arity $ar(K) \geq 0$ and an associated finite, non-empty set D_K of data constructors, s.t. every $c_{K,i} \in D_K$ has an arity $ar(c_{K,i}) \geq 0$. We assume that $\mathscr{K}$ includes type constructors for lists, pairs and Booleans together with the data constructors `Nil` and `Cons`, `Pair`, and the constants `True` and `False`.

The syntax of expressions and types of LRPw is defined in Fig. 1, where we assume that variables have a fixed type, written as $x :: \rho$. The calculus LRPw extends the lambda-

calculus by recursive let-expressions, data constructors, `case`-expressions (for every type constructor K), `seq`-expressions and by type abstractions $\Lambda A.s$ and type applications $(s\ \tau)$ in order to express polymorphic functions and type instantiation and by shared work-decorations $a{:=}n$ and $^{[a]}$. A `letrec`-binding $a{:=}n$ means that a work load of n essential reduction steps is associated with label a where the shared position is the top of the `letrec`-expression, the construct $s^{[a]}$ means that before expression s can be evaluated the work associated with label a has to be evaluated.

Polymorphically typed variables are only permitted for usual bindings of let-environments; at other places, the language is monomorphic where the concrete types can be computed through type reductions. For example, the identity can be written as $\Lambda A.\lambda x :: A.x$, and an application to the constant `True` is written $(\Lambda A.\lambda x :: A.x)$ `Bool True`. The reduction is $(\Lambda A.\lambda x :: A.x)$ `Bool True` $\rightarrow (\lambda x ::$ `Bool`$.x)$ `True` $\rightarrow$ (`letrec` x=`True in` x). An expression s is *well-typed* with type τ (polymorphic type ρ, resp.), written as $s :: \tau$ (or $s :: \rho$, resp.), if s can be typed with the typing rules in Fig. 1 with type τ (ρ, resp.).

The calculus LR [SSSS08] is the untyped variant of LRPw (without work-decorations), where types and type-reduction are removed. In the following we often ignore the types and omit the types at variables and also sometimes omit the type reductions. We use some abbreviations: We write $\lambda x_1,\ldots,x_n.s$ instead of $\lambda x_1.\ldots.\lambda x_n.s$. Parts of a `letrec`-environment are abbreviated by *Env*, and with $\{x_{g(i)}{=}s_{f(i)}\}_{i=j}^{m}$ we abbreviate the bindings $x_{g(j)}{=}s_{f(j)},\ldots,x_{g(m)}{=}s_{g(m)}$. Alternatives of `case`-expressions are abbreviated by *alts*. Constructor applications $(c_{K,i}\ s_1\ \ldots\ s_{ar(c_{K,i})})$ are abbreviated using vector notation, omitting the index as $c\,\overrightarrow{s}$. We use $FV(s)$ and $BV(s)$ to denote the free and bound variables of an expression s, and $FN(s)$ and $BN(s)$ to denote the free and bound label-names of an expression s. An expression s is closed iff $FV(s) = \emptyset$ and $FN(s) = \emptyset$. In an environment $Env = \{x_i{=}t_i\}_{i=1}^{n}$, we define $LV(Env) = \{x_1,\ldots,x_n\}$.

A *value* is an abstraction $\lambda x.s$, a type abstraction $\Lambda A_1.\ldots.\Lambda A_n.\lambda x.s$, or a constructor application $c\,\overrightarrow{s}$. A *context* C is an expression with one hole $[\cdot]$ at expression position.

The reduction rules of the calculus are in Fig. 3, where we use the following unions: (case) is the union of (case-c), (case-in), (case-e); (seq) is the union of (seq-c), (seq-in), (seq-e); (cp) is the union of (cp-in), (cp-e); (llet) is the union of (llet-in), (llet-e); (lll) is the union of (lapp),(lcase),(lseq),(llet-in),(llet-e); (letwn) is the union of (letwn-in), (letwn-e); (letw0) is the union of (letw0-in), (letw0-e); (letw) is the union of (letwn), (letw0).

The operational semantics of LRPw is defined by the normal order reduction strategy which is a call-by-need strategy. The labeling algorithm shown in Fig. 2 is used to detect the position to which a reduction rule is applied according to normal order. It uses the labels top, sub, vis, and nontarg where top means reduction of the top term, sub means reduction of a subterm, vis marks already visited subexpressions, and nontarg marks already visited variables that are not target of a (cp)-reduction. Note that the labeling algorithm does not descend into sub-labeled `letrec`-expressions. The rules of the labeling algorithm are in Fig. 3. If the labeling algorithm terminates without a fail, then a potential normal order redex is found, which can only be the direct superterm of the sub-marked subexpression.

Labeling algorithm: Labeling of s starts with s^{top}. The rules from below are applied exhaustively or until a fail occurs, where $a \vee b$ means label a or label b.

$(s\ t)^{\mathsf{sub}\vee\mathsf{top}}$	$\to (s^{\mathsf{sub}}\ t)^{\mathsf{vis}}$
$(\mathtt{letrec}\ Env\ \mathtt{in}\ s)^{\mathsf{top}}$	$\to (\mathtt{letrec}\ Env\ \mathtt{in}\ s^{\mathsf{sub}})^{\mathsf{vis}}$
$\mathtt{letrec}\ x{=}s, Env\ \mathtt{in}\ C[x^{\mathsf{sub}}]$	$\to \mathtt{letrec}\ x{=}s^{\mathsf{sub}}, Env\ \mathtt{in}\ C[x^{\mathsf{vis}}]$
$\mathtt{letrec}\ x{=}s, y{=}C[x^{\mathsf{sub}}], Env\ \mathtt{in}\ t$	$\to \mathtt{letrec}\ x{=}s^{\mathsf{sub}}, y{=}C[x^{\mathsf{vis}}], Env\ \mathtt{in}\ t$, if $C{\neq}[\cdot]$
$\mathtt{letrec}\ x{=}s, y{=}x^{\mathsf{sub}}, Env\ \mathtt{in}\ t$	$\to \mathtt{letrec}\ x{=}s^{\mathsf{sub}}, y{=}x^{\mathsf{nontarg}}, Env\ \mathtt{in}\ t$
$(\mathtt{seq}\ s\ t)^{\mathsf{sub}\vee\mathsf{top}}$	$\to (\mathtt{seq}\ s^{\mathsf{sub}}\ t)^{\mathsf{vis}}$
$(\mathtt{case}_K\ s\ alts)^{\mathsf{sub}\vee\mathsf{top}}$	$\to (\mathtt{case}_K\ s^{\mathsf{sub}}\ alts)^{\mathsf{vis}}$
$\mathtt{letrec}\ x{=}s^{\mathsf{vis}\vee\mathsf{nontarg}}, y{=}C[x^{\mathsf{sub}}] \ldots$	$\to Fail$
$\mathtt{letrec}\ x{=}C[x^{\mathsf{sub}}], Env\ \mathtt{in}\ t$	$\to Fail$

Fig. 2: Labeling algorithm

The labelings in the expressions in the reduction rules shown in Fig. 3 indicate the exact place and positions of the expressions and subexpressions involved in the reduction step.

However, it may also happen that there is no normal order reduction, since either the evaluation is already finished, or a black hole is detected, or the labeling fails.

Definition 2.1. *For an expression t, a* normal order reduction step $t \xrightarrow{\text{LRPw}} t'$ *is defined by first applying the labeling algorithm to t, and if the labeling algorithm terminates without a fail, then one of the rules in Fig. 3 has to be applied resulting in t', if possible, where the labels* sub, vis *must match the labels in the expression t. A* weak head normal form (WHNF) *is a value v, or an expression* letrec *Env* in *v, where v is a value, or an expression* $\mathtt{letrec}\ x_1{=}c\ \overrightarrow{t}, \{x_i{=}x_{i-1}\}_{i=2}^{m}, Env\ \mathtt{in}\ x_m$. *An expression s* converges*, denoted as $s{\downarrow_{\text{LRPw}}}$, iff there exists a normal-order reduction $s \xrightarrow{\text{LRPw},*} s'$, where s' is a WHNF. We write $s{\uparrow_{\text{LRPw}}}$ iff $s{\downarrow_{\text{LRPw}}}$ does not hold. With $\bot$ we denote a diverging, closed expression.*

Note that there are diverging expressions of any type, for example $\mathtt{letrec}\ x :: \rho{=}x\ \mathtt{in}\ x$.

Definition 2.2. *The calculus* LRP *is the subcalculus of* LRPw *which does not have the syntactic constructs $a{:=}n$ and $s^{[a]}$, and the operational semantics of* LRP *does not have the reduction rules (letwn) and (letw0). WHNFs are defined as in* LRPw. *Convergence $\downarrow_{\text{LRP}}$ is defined accordingly.*

Lemma 2.3. *For every* LRPw*-expression s which is also an* LRP*-expression (i.e. s has no decorations and no $a{:=}n$-construct): $s \downarrow_{\text{LRPw}} \iff s \downarrow_{\text{LRP}}$.*

Definition 2.4. *A* reduction context *R is any context, such that its hole will be labeled with* sub *or* top *by the labeling algorithm in Fig. 1. A* weak reduction context*, R^{-}, is a reduction context, where the hole is not within a* letrec*-expression.* Surface contexts *S are contexts where the hole is not in an abstraction,* top contexts *T are surface contexts where the hole is not in an alternative of a case, and* weak top contexts *are top contexts where the hole does not occur in a* letrec*. A context C is* strict *iff $C[\bot] \sim_c \bot$.*

(lbeta)	$C[((\lambda x.s)^{\mathsf{sub}}\ r)] \to C[\mathtt{letrec}\ x{=}r\ \mathtt{in}\ s]$
(Tbeta)	$((\Lambda A.u)^{\mathsf{sub}}\ \tau) \to u[\tau/A]$
(cp-in)	$\mathtt{letrec}\ x_1{=}v^{\mathsf{sub}}, \{x_i{=}x_{i-1}\}_{i=2}^{m}, Env\ \mathtt{in}\ C[x_m^{\mathsf{vis}}]$
	$\to \mathtt{letrec}\ x_1{=}v, \{x_i{=}x_{i-1}\}_{i=2}^{m}, Env\ \mathtt{in}\ C[v]$, where v is a polymorphic abstraction
(cp-e)	$\mathtt{letrec}\ x_1{=}v^{\mathsf{sub}}, \{x_i{=}x_{i-1}\}_{i=2}^{m}, Env, y{=}C[x_m^{\mathsf{vis}}]\ \mathtt{in}\ r$
	$\to \mathtt{letrec}\ x_1{=}v, \{x_i{=}x_{i-1}\}_{i=2}^{m}, Env, y{=}C[v]\ \mathtt{in}\ r$, where v is a polymorphic abstraction
(llet-in)	$(\mathtt{letrec}\ Env_1\ \mathtt{in}\ (\mathtt{letrec}\ Env_2\ \mathtt{in}\ r)^{\mathsf{sub}}) \to (\mathtt{letrec}\ Env_1, Env_2\ \mathtt{in}\ r)$
(llet-e)	$\mathtt{letrec}\ Env_1, x{=}(\mathtt{letrec}\ Env_2\ \mathtt{in}\ t)^{\mathsf{sub}}\ \mathtt{in}\ r \to \mathtt{letrec}\ Env_1, Env_2, x{=}t\ \mathtt{in}\ r$
(lapp)	$C[((\mathtt{letrec}\ Env\ \mathtt{in}\ t)^{\mathsf{sub}}\ s)] \to C[(\mathtt{letrec}\ Env\ \mathtt{in}\ (t\ s))]$
(lcase)	$C[\mathtt{case}_K\ (\mathtt{letrec}\ Env\ \mathtt{in}\ t)^{\mathsf{sub}}\ alts] \to C[(\mathtt{letrec}\ Env\ \mathtt{in}\ \mathtt{case}_K\ t\ alts)]$
(lseq)	$C[(\mathtt{seq}\ (\mathtt{letrec}\ Env\ \mathtt{in}\ s)^{\mathsf{sub}}\ t)] \to C[(\mathtt{letrec}\ Env\ \mathtt{in}\ (\mathtt{seq}\ s\ t))]$
(seq-c)	$C[(\mathtt{seq}\ v^{\mathsf{sub}}\ t)] \to C[t]$, if v is a value
(seq-in)	$(\mathtt{letrec}\ x_1{=}(c\ \overrightarrow{s})^{\mathsf{sub}}, \{x_i{=}x_{i-1}\}_{i=2}^{m}, Env\ \mathtt{in}\ C[(\mathtt{seq}\ x_m^{\mathsf{vis}}\ t)])$
	$\to (\mathtt{letrec}\ x_1{=}(c\ \overrightarrow{s}), \{x_i{=}x_{i-1}\}_{i=2}^{m}, Env\ \mathtt{in}\ C[t])$
(seq-e)	$(\mathtt{letrec}\ x_1{=}(c\ \overrightarrow{s})^{\mathsf{sub}}, \{x_i{=}x_{i-1}\}_{i=2}^{m}, Env, y{=}C[(\mathtt{seq}\ x_m^{\mathsf{vis}}\ t)]\ \mathtt{in}\ r)$
	$\to (\mathtt{letrec}\ x_1{=}(c\ \overrightarrow{s}), \{x_i{=}x_{i-1}\}_{i=2}^{m}, Env, y{=}C[t]\ \mathtt{in}\ r)$
(case-c)	$C[\mathtt{case}_K\ (c\ \overrightarrow{t})^{\mathsf{sub}} \ldots ((c\ \overrightarrow{y}) \twoheadrightarrow t) \ldots] \to C[\mathtt{letrec}\ \{y_i{=}t_i\}_{i=1}^{ar(c)}\ \mathtt{in}\ t]$ if $ar(c) \geq 1$
(case-c)	$C[\mathtt{case}_K\ c^{\mathsf{sub}} \ldots (c \twoheadrightarrow t) \ldots] \to C[t]$, if $ar(c) = 0$
(case-in)	$\mathtt{letrec}\ x_1{=}(c\ \overrightarrow{t})^{\mathsf{sub}}, \{x_i{=}x_{i-1}\}_{i=2}^{m}, Env\ \mathtt{in}\ C[\mathtt{case}_K\ x_m^{\mathsf{vis}} \ldots ((c\ \overrightarrow{z}) \twoheadrightarrow t) \ldots]$
	$\to \mathtt{letrec}\ x_1{=}(c\ \overrightarrow{y}), \{y_i{=}t_i\}_{i=1}^{ar(c)}, \{x_i{=}x_{i-1}\}_{i=2}^{m}, Env\ \mathtt{in}\ C[\mathtt{letrec}\ \{z_i{=}y_i\}_{i=1}^{ar(c)}\ \mathtt{in}\ t]$, if $ar(c) \geq 1$ and where y_i are fresh
(case-in)	$\mathtt{letrec}\ x_1{=}c^{\mathsf{sub}}, \{x_i{=}x_{i-1}\}_{i=2}^{m}, Env\ \mathtt{in}\ C[\mathtt{case}_K\ x_m^{\mathsf{vis}} \ldots (c \twoheadrightarrow t) \ldots]$
	$\to \mathtt{letrec}\ x_1{=}c, \{x_i{=}x_{i-1}\}_{i=2}^{m}, Env\ \mathtt{in}\ C[t]$ if $ar(c) = 0$
(case-e)	$\mathtt{letrec}\ x_1{=}(c\ \overrightarrow{t})^{\mathsf{sub}}, \{x_i{=}x_{i-1}\}_{i=2}^{m}, u{=}C[\mathtt{case}_K\ x_m^{\mathsf{vis}} \ldots ((c\ \overrightarrow{z}) \twoheadrightarrow r) \ldots], Env\ \mathtt{in}\ s$
	$\to \mathtt{letrec} x_1{=}(c\ \overrightarrow{y}), \{y_i{=}t_i\}_{i=1}^{n}, \{x_i{=}x_{i-1}\}_{i=2}^{m}, u{=}C[\mathtt{letrec}\ \{z_i{=}y_i\}_{i=1}^{n}\ \mathtt{in} r], Env\,\mathtt{in}\, s$
	where $n = ar(c) \geq 1$ and y_i are fresh
(case-e)	$\mathtt{letrec}\ x_1{=}c^{\mathsf{sub}}, \{x_i{=}x_{i-1}\}_{i=2}^{m}, u{=}C[\mathtt{case}_K\ x_m^{\mathsf{vis}}\ \ldots (c \twoheadrightarrow t) \ldots], Env\ \mathtt{in}\ s$
	$\to \mathtt{letrec}\ x_1{=}c, \{x_i{=}x_{i-1}\}_{i=2}^{m}, u{=}C[t], Env\ \mathtt{in}\ s$, if $ar(c) = 0$
(letwn-in)	$\mathtt{letrec}\ Env, a{:=}n,\ \mathtt{in}\ C[(s^{[a]})^{\mathsf{sub}}] \to \mathtt{letrec}\ Env, a{:=}n{-}1\ \mathtt{in}\ C[s^{[a]}]$, if $n > 0$
(letwn-e)	$\mathtt{letrec}\ a{:=}n, x{=}C[(s^{[a]})^{\mathsf{sub}}], Env\ \mathtt{in}\ r \to \mathtt{letrec}\ a{:=}n{-}1, x{=}C[s^{[a]}], Env\ \mathtt{in}\ r$, if $n > 0$
(letw0-in)	$\mathtt{letrec}\ Env, a{:=}0,\ \mathtt{in}\ C[(s^{[a]})^{\mathsf{sub}}] \to \mathtt{letrec}\ Env, a{:=}0\ \mathtt{in}\ C[s]$
(letw0-e)	$\mathtt{letrec}\ a{:=}0, x{=}C[(s^{[a]})^{\mathsf{sub}}], Env\ \mathtt{in}\ r \to \mathtt{letrec}\ a{:=}0, x{=}C[s], Env\ \mathtt{in}\ r$

Fig. 3: Reduction rules

2.2 Contextual Equivalence and Improvement in LRP and LRPw

The main measure for estimating the time consumption of computation in this paper is a measure counting *essential* reduction steps in the normal-order reduction of expressions. We omit the type reductions in this measure, since these are always terminating and usually can be omitted after compilation. We define the essential reduction length for both calculi, where we allow some freedom in which reduction rules (as a subset of $\{lbeta, case, seq, letwn\}$) should be seen as essential. Clearly, we require that *letwn*-reductions are always counted (since they represent essential work). We also require that

(lbeta)-reductions are always counted, since there are expressions which have no (case)- or (seq)-reductions but an unbounded number of (lbeta)-reductions (for a detailed discussion and analysis on the length measures, see [SSS15a]).

Definition 2.5. *Let $\mathfrak{A} = \{lbeta, case, seq, letwn\}$, $A_{min} = \{letwn, lbeta\}$, $A_{min} \subseteq A \subseteq \mathfrak{A}$, $L \in \{\text{LRP}, \text{LRPw}\}$, and let t be a closed L-expression with $t{\downarrow_L}t_0$. Then $\mathtt{rln}_A(t)$ is the number of a-reductions in the normal order reduction $t{\downarrow_L}t_0$ where $a \in A$. We define the measures as ∞, if $t{\uparrow_L}$. For a reduction $t \xrightarrow{L,*} t'$, we define $\mathtt{rln}(t \xrightarrow{L,*} t')$ as the number of (lbeta)-, (case)-, (seq)-, and (in* LRPw*) (letwn)-reductions in it.*

We define contextual equivalence and the improvement relation for LRPw and LRP:

Definition 2.6. *For $L \in \{\text{LRP}, \text{LRPw}\}$, let s,t be two L-expressions of the same type ρ and let $A_{min} \subseteq A \subseteq \mathfrak{A}$. We define the* contextual preorder $\leq_{c,L}$*, the* contextual equivalence $\sim_{c,L}$*, and the A-*improvement relation $\preceq_{A,L}$*:*

$$
\begin{array}{lll}
s \leq_{c,L} t & \textit{iff} & \textit{for all L-contexts } C[\cdot :: \rho]\textit{: } C[s]{\downarrow_L} \implies C[t]{\downarrow_L} \\
s \sim_{c,L} t & \textit{iff} & \textit{for all L-contexts } C[\cdot :: \rho]\textit{: } C[s]{\downarrow_L} \iff C[t]{\downarrow_L} \\
s \preceq_{A,L} t & \textit{iff} & s \sim_{c,L} t \textit{ and for all L-contexts } C[\cdot :: \rho] \textit{ s.t. } C[s], C[t] \textit{ are closed:} \\
 & & \mathtt{rln}_A(C[s]) \leq \mathtt{rln}_A(C[t])
\end{array}
$$

If $s \preceq_{A,L} t$ then we say s A-improves t. If $s \preceq_{A,L} t$ and $t \preceq_{A,L} s$, then we write $s \approx_{A,L} t$. A program transformation P is correct *if $P \subseteq \sim_{c,L}$ and it is an A-*improvement *iff $P \subseteq \succeq_{A,L}$.*

The following context lemma for $\sim_c$ holds in LRP and also in LRPw. The proof is standard, so we omit it.

Lemma 2.7 (Context Lemma for Equivalence)**.** *Let $L \in \{\text{LRP}, \text{LRPw}\}$ and s,t be L-expressions of the same type. Then $s \leq_c t$ iff for all $C \in \{R, S, T\}$: $C[s]{\downarrow_L} \implies C[t]{\downarrow_L}$.*

Let $\eta \in \{\leq, =, \geq\}$ be a relation on non-negative integers, X be a class of contexts X (we will instantiate X with: all contexts C; all reduction contexts R; all surface contexts S; or all top-contexts T), and let $A_{min} \subseteq A \subseteq \mathfrak{A}$. For expressions s,t of type ρ, let $s \bowtie_{A,\eta,X} t$ iff for all X-contexts $X[\cdot : \rho]$, s.t. $X[s], X[t]$ are closed: $\mathtt{rln}_A(X[s])\ \eta\ \mathtt{rln}_A(X[t])$. In particular, $\bowtie_{A,\leq,C} = \preceq_A$, $\bowtie_{A,\geq,C} = \succeq_A$, and $\bowtie_{A,=,C} = \approx_A$.

In the following we formulate statements for the calculus LRPw, if not stated otherwise.

The context lemma for improvement shows that it suffices to take reduction contexts into account for proving improvement. Its proof is similar to the ones for context lemmas for contextual equivalence in call-by-need lambda calculi (see [SSS15b, SSSS08, SSS10]). The proof is nearly a complete copy of the proof of the context lemma for improvement in LRP (see [SSS15b]). We omit it, but it can be found in the technical report [SSS15d].

Lemma 2.8 (Context Lemma for Improvement)**.** *Let s,t be expressions with $s \sim_c t$, $\eta \in \{\leq, =, \geq\}$, and let $X \in \{R, S, T\}$. Then $s \bowtie_{A,\eta,X} t$ iff $s \bowtie_{A,\eta,C} t$.*

(gc1)	$\texttt{letrec}\ \{x_i{=}s_i\}_{i=1}^{n}, Env\ \texttt{in}\ t \to \texttt{letrec}\ Env\ \texttt{in}\ t,$ if $\forall i : x_i \notin FV(t, Env)$
(gc2)	$\texttt{letrec}\ x_1{=}s_1, \ldots, x_n{=}s_n\ \texttt{in}\ t \to t,$ if for all $i : x_i \notin FV(t)$
(gcW1)	$\texttt{letrec}\ Env, a_1{:=}n_1, \ldots, a_m{:=}n_m\ \texttt{in}\ s \to \texttt{letrec}\ Env\ \texttt{in}\ s,$ if labels $a_1, \ldots, a_m$ do not occur in Env or s
(gcW2)	$\texttt{letrec}\ a_1{:=}n_1, \ldots, a_m{:=}n_m\ \texttt{in}\ s \to s,$ if $a_1, \ldots, a_m$ do not occur in s
(cpx-in)	$\texttt{letrec}\ x{=}y, Env\ \texttt{in}\ C[x] \to \texttt{letrec}\ x{=}y, Env\ \texttt{in}\ C[y],$ if $y \in Var, x \neq y$
(cpx-e)	$\texttt{letrec}\ x{=}y, z{=}C[x], Env\ \texttt{in}\ t \to \texttt{letrec}\ x{=}y, z{=}C[y], Env\ \texttt{in}\ t,$ if $y \in Var, x \neq y$
(cpcx-in)	$\texttt{letrec}\ x{=}c\,\overrightarrow{t}, Env\ \texttt{in}\ C[x] \to \texttt{letrec}\ x{=}c\,\overrightarrow{y}, \{y_i{=}t_i\}_{i=1}^{n}, Env\ \texttt{in}\ C[c\,\overrightarrow{y}]$
(cpcx-e)	$\texttt{letrec}\ x{=}c\,\overrightarrow{t}, z{=}C[x], Env\ \texttt{in}\ t \to \texttt{letrec}\ x{=}c\,\overrightarrow{y}, \{y_i{=}t_i\}_{i=1}^{n}, z{=}C[c\,\overrightarrow{y}], Env\ \texttt{in}\ t$
(abs)	$\texttt{letrec}\ x{=}c\,\overrightarrow{t}, Env\ \texttt{in}\ s \to \texttt{letrec}\ x{=}c\,\overrightarrow{x}, \{y_i{=}t_i\}_{i=1}^{ar(c)}, Env\ \texttt{in}\ s$
(abse)	$(c\,\overrightarrow{t}) \to \texttt{letrec}\ \{y_i{=}t_i\}_{i=1}^{ar(c)}\ \texttt{in}\ c\,\overrightarrow{x}$
(xch)	$\texttt{letrec}\ x{=}t, y{=}x, Env\ \texttt{in}\ r \to \texttt{letrec}\ y{=}t, x{=}y, Env\ \texttt{in}\ r$
(lwas)	$T[\texttt{letrec}\ Env\ \texttt{in}\ t] \to \texttt{letrec}\ Env\ \texttt{in}\ T[t],$ if T is a weak top context of hole depth 1
(ucp1)	$\texttt{letrec}\ Env, x{=}t\ \texttt{in}\ S[x] \to \texttt{letrec}\ Env\ \texttt{in}\ S[t]$
(ucp2)	$\texttt{letrec}\ Env, x{=}t, y{=}S[x]\ \texttt{in}\ r \to \texttt{letrec}\ Env, y{=}S[t]\ \texttt{in}\ r$
(ucp3)	$\texttt{letrec}\ x{=}t\ \texttt{in}\ S[x] \to S[t]$ where in the (ucp)-rules, $x \notin FV(S, Env, t, r)$ and S is a surface context

Fig. 4: Extra transformation rules

3 Properties of Reductions and Transformations

In this section we prove properties about the reduction rules and the additional transformation rules shown in Fig. 4 where we use the following unions: (gc) is the union of (gc1), (gc2); (gcW) is the union of (gcW1), (gcW2); (cpx) is the union of (cpx-in), (cpx-e); (cpcx) is the union of (cpcx-in), (cpcx-e); and (ucp) is the union of (ucp1), (ucp2), (ucp3).

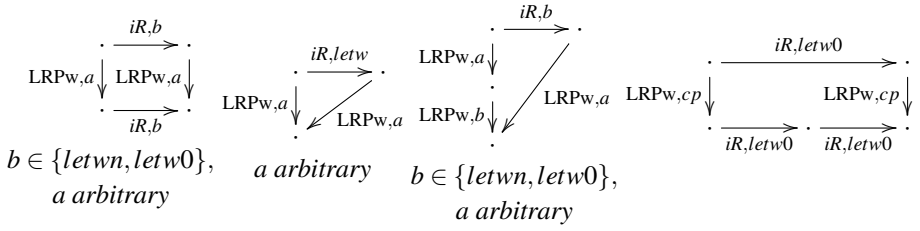
A program transformation P is a binary relation on expressions, we write $s \xrightarrow{P} t$, if $(s,t) \in P$. For a set of contexts X, we write (X,P) for the closure of P w.r.t. the contexts in X, i.e. $s \xrightarrow{X,P} t$ iff there exists $C \in X$ with $C[s] \xrightarrow{P} C[t]$. A step $s \xrightarrow{X,P} t$ is *internal* if it is not a normal order reduction step. We denote internal steps by $\xrightarrow{iX,P}$ or by $\xrightarrow{i,P}$ (for all contexts), i.e. $\xrightarrow{iX,P} = \xrightarrow{X,P} \setminus \xrightarrow{\text{LRPw}}$.

As a further technique, we use so-called forking and commuting diagrams. Let $\xrightarrow{P}$ be a program transformation. A forking diagram describes how overlappings $r \xleftarrow{\text{LRPw}} s \xrightarrow{P} t$ can be closed by a sequence of the form $r \xrightarrow{P',*} s' \xleftarrow{\text{LRPw},*} t$. A commuting diagram describes how overlappings $r \xrightarrow{\text{LRPw}} s \xrightarrow{P} t$ can be closed by a sequence of the form $r \xrightarrow{\text{LRPw},*} s' \xrightarrow{P',*} t$. Here P' may be the transformation P or other transformations. The diagrams abstract from the concrete expressions r,s,t and thus the symbol $\cdot$ is used as a place-holder for the universal and existentially quantified expressions. A set of forking diagrams is complete for transformation P if for every concrete overlapping $r \xleftarrow{\text{LRPw}} s \xrightarrow{P} t$ an applicable diagram is in the set, where applicable means that the expressions, reductions, and transformations exist. Accordingly, a set of commuting diagrams is complete if for every concrete sequence

$r \xrightarrow{\text{LRPw}} s \xrightarrow{P} t$ an applicable diagram is in the set. Often forking and commuting diagrams have a common representation and thus we will depict the diagrams only once.

3.1 Analyzing the Transformation (letw)

Lemma 3.1. *A complete set of forking and commuting diagrams for internal (letw)-transformations applied in reduction contexts can be read off the diagrams:*



Proof. The first diagram describes the commuting case and it also includes cases where a (letw-in)-transformation is flipped into an (letw-e)-transformation, if the normal order reduction is (LRPw,llet). The second diagram describes the case where the a-labeled expression of the (letw)-transformation is removed by the normal order reduction. The third diagram describes the case where the internal (letw)-transformation becomes a normal-order reduction. The fourth diagram describes the case where an a-labeled expression is inside an abstraction which is copied by (LRPw,cp). If the transformation is a (letwn), then the transformations commute, but if the transformation is (letw0), then the transformation is duplicated, since it has to remove the a-label twice. □

Lemma 3.2. *If $s \xrightarrow{iR,letw} t$ then s is a WHNF iff t is a WHNF.*

Proposition 3.3. *The transformations (letw0) and (letwn) are correct.*

Proof. We use the context lemma (Lemma 2.7) and thus it suffices to show that whenever $s \xrightarrow{letw} t$, then for all reduction contexts: $R[s]{\downarrow_{\text{LRPw}}} \iff R[t]{\downarrow_{\text{LRPw}}}$. We first show $R[s]{\downarrow_{\text{LRPw}}} \implies R[t]{\downarrow_{\text{LRPw}}}$: Assume that $R[s] \xrightarrow{\text{LRPw},k} r$ where r is a WHNF. We show $R[t] \xrightarrow{\text{LRPw},k'} r'$ where r' is a WHNF, and $k' \leq k$ by induction on k. The base case $k = 0$ holds by Lemma 3.2. For the induction step let $R[s] \xrightarrow{\text{LRPw}} r_1 \xrightarrow{\text{LRPw},k-1} r$. If $R[s] \xrightarrow{\text{LRPw},letw} R[t]$, then $r_1 = R[t]$ and $R[t] \xrightarrow{\text{LRPw},k-1} r$ and the claim holds. If the reduction is internal, then apply a forking diagram to $r_1 \xleftarrow{\text{LRPw}} R[s] \xrightarrow{\text{LRPw},letw} R[t]$. For the first diagram, we have $r_1 \xrightarrow{iR,letw} r_1'$, $R[t] \xrightarrow{\text{LRPw}} r_1'$ and $r_1 \xrightarrow{\text{LRPw},k-1} r$. Applying the induction hypothesis to r_1 and r_1' yields $r_1' \xrightarrow{\text{LRPw},k''} r'$ where r' is a WHNF and $k'' \leq k-1$. Thus $R[t] \xrightarrow{\text{LRPw},k'} r'$ where r' is a WHNF and $k' \leq k$. If the second diagram is applied, then $R[t] \xrightarrow{\text{LRPw}} r_1 \xrightarrow{\text{LRPw},k-1} r$ and the claim holds. If the third diagram is applied, then $R[t] \xrightarrow{\text{LRPw}} r_2 \xrightarrow{\text{LRPw},k-2} r$ (where

$r_1 \xrightarrow{\text{LRPw}} r_2$) and the claim holds. In case of diagram (4), we apply the induction hypothesis twice for each $(iR, letw)$-transformation, which shows that $R[t] \xrightarrow{\text{LRPw},cp} r_1' \xrightarrow{\text{LRPw},k''} r'$ where r' is a WHNF, $k'' \leq k-1$. Thus the claim holds.

For proving $R[t]{\downarrow_{\text{LRPw}}} \implies R[s]{\downarrow_{\text{LRPw}}}$, let $\#cp(r)$ be the number of (LRPw,cp)-reductions in the normal order reductions from r to a WHNF and $\#cp(r) = \infty$ if $r \uparrow$. Assume that $R[t] \xrightarrow{\text{LRPw},k} r$ where r is a WHNF. We show $R[s]{\downarrow_{\text{LRPw}}}$ and $\#cp(R[s]) \leq \#cp(R[t])$ by induction on the measure $(\#cp(R[t]), k)$. For the base case (0,0) $R[t]$ is a WHNF and thus by Lemma 3.2 also $R[s]$ is a WHNF and the claim holds. For the induction step let $(l,k) > (0,0)$. Then $R[t] \xrightarrow{\text{LRPw}} t' \xrightarrow{\text{LRPw},k-1} r$ where r is a WHNF. If $R[s] \xrightarrow{\text{LRPw},letw} R[t]$, then the claim holds, i.e. $R[s]{\downarrow_{\text{LRPw}}}$ and $\#cp(R[s]) = \#cp(R[t])$. If the transformation is internal, then we apply a commuting diagram to $R[s] \xrightarrow{iR,letw} R[t] \xrightarrow{\text{LRPw}} t_1$. For the second and the third diagram, the claim obviously holds. For the first diagram, there exists s_1 s.t. $R[s] \xrightarrow{\text{LRPw},a} s_1$, $s_1 \xrightarrow{iR,letw} s_2$ and the measure for t_1 is $(\#cp(t_1), k-1)$ which is strictly smaller than (l,k) (since $\#cp(t_1) \leq l$). Thus we can apply the induction hypothesis and derive $s_1{\downarrow_{\text{LRPw}}}$ and $\#cp(s_1) \leq \#cp(t_1)$. This shows $R[s]{\downarrow_{\text{LRPw}}}$ and $\#cp(R[s]) \leq \#(R[t])$. For the last diagram, we apply the induction hypothesis twice, which is possible since $\#cp(\cdot)$ is strictly decreased. □

Proposition 3.4. *For $A_{min} \subseteq A \subseteq \mathfrak{A}$: (letw0) $\subseteq \approx_A$.*

Proof. We use the context lemma for improvement and since $(letw0)$ is correct, it suffices to show that if $s \xrightarrow{letw0} t$ implies $\mathtt{rln}_A(R[s]) = \mathtt{rln}_A(R[t])$ for all reduction contexts R, s.t. $R[s]$, $R[t]$ are closed. Since $(letw0)$ is correct, we know that $\mathtt{rln}_A(R[s]) = \infty \iff \mathtt{rln}_A(R[t]) = \infty$. So suppose that $\mathtt{rln}_A(R[s]) = n$. We show $\mathtt{rln}_A(R[t]) = n$ by induction on a normal order reduction $R[s] \xrightarrow{\text{LRPw},k} s'$ where s' is a WHNF. The base case holds by Lemma 3.2. For the induction step, let $R[s] \xrightarrow{\text{LRPw}} s_1 \xrightarrow{\text{LRPw},k-1} s'$. If $R[s] \xrightarrow{\text{LRPw},letw0} R[t]$, then $\mathtt{rln}_A(R[s]) = \mathtt{rln}_A(R[t]) = \mathtt{rln}_A(s_1)$ and the claim holds. If the transformation is internal, then we apply a forking diagram to s_1. For the first diagram we have $s_1 \xrightarrow{iR,letw0} t_1$ and we apply the induction hypothesis to s_1 and thus have $\mathtt{rln}_A(s_1) = \mathtt{rln}_A(t_1)$. This also shows $\mathtt{rln}_A(R[s]) = \mathtt{rln}_A(R[t])$. For the second diagram the claim holds. For the third diagram the claim also holds, since the additional (LRPw,letw0)-reduction in the normal order reduction for $R[s]$ is not counted in the $\mathtt{rln}_A$-measure. For the fourth diagram we have $s_1 \xrightarrow{iR,letw0} s_1' \xrightarrow{iR,letw0} t_1 \xleftarrow{\text{LRPw},cp} R[t]$. We apply the induction hypothesis twice: For s_1 we get $\mathtt{rln}_A(s_1) = \mathtt{rln}_A(s_1')$ and for s_1' we get $\mathtt{rln}_A(s_1') = \mathtt{rln}_A(t_1)$ which finally shows $\mathtt{rln}_A(R[t]) = \mathtt{rln}_A(t_1) = \mathtt{rln}_A(s_1) = \mathtt{rln}_A(R[s])$. □

Proposition 3.5. *For $A_{min} \subseteq A \subseteq \mathfrak{A}$, (letwn) $\subseteq \succeq_A$.*

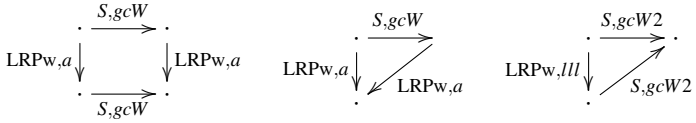
Proof. We use the context lemma for improvement. We already know that $(letwn)$ is correct. We show that if $s \xrightarrow{letwn} t$, then for all reduction contexts R s.t. $R[s]$ and $R[t]$ are closed: $\mathtt{rln}_A(R[s]) = \mathtt{rln}_A(R[t])$ or $\mathtt{rln}_A(R[s]) = 1 + \mathtt{rln}_A(R[t])$. Since $(letwn)$ is correct, we

know that $\mathtt{rln}_A(R[s]) = \infty \iff \mathtt{rln}_A(R[t]) = \infty$. So suppose that $\mathtt{rln}_A(R[s]) = n$. We show $\mathtt{rln}_A(R[t]) = n$ or $\mathtt{rln}_A(R[t]) = n+1$ by induction on a normal order reduction $R[s] \xrightarrow{\text{LRPw},k} s'$ where s' is a WHNF. The base case holds by Lemma 3.2. For the induction step, let $R[s] \xrightarrow{\text{LRPw}} s_1 \xrightarrow{\text{LRPw},k-1} s'$. If $R[s] \xrightarrow{\text{LRPw},letwn} R[t]$, then $\mathtt{rln}_A(R[s]) = 1 + \mathtt{rln}_A(R[t])$ and the claim holds. If the transformation is internal, then we apply a forking diagram to s_1. For the first diagram we have $s_1 \xrightarrow{iR,letwn} t_1$ and we apply the induction hypothesis to s_1 and thus have $\mathtt{rln}_A(s_1) = 1 + \mathtt{rln}_A(t_1)$ or $\mathtt{rln}_A(s_1) = \mathtt{rln}_A(t_1)$. This also shows $\mathtt{rln}_A(R[s]) = 1 + \mathtt{rln}_A(R[t])$ or $\mathtt{rln}_A(R[s]) = \mathtt{rln}_A(R[t])$. For the second diagram we have $\mathtt{rln}_A(R[s]) = \mathtt{rln}_A(R[t])$. For the third diagram we have $\mathtt{rln}_A(R[s]) = 1 + \mathtt{rln}_A(R[t])$. The fourth diagram is not applicable, since the given transformation is (letwn). □

3.2 Analyzing the Transformation (gcW)

We prove correctness and (invariance w.r.t. $\approx$) for (gcW), the transformation which performs garbage collection of $a{:=}n$-bindings which have no corresponding $^{[a]}$-label.

Lemma 3.6. *Complete sets of forking and commuting diagrams for (S,gcW) are:*



a arbitrary *a arbitrary*

Proof. The first diagram covers the case where the transformation and the reduction commute. There are also cases where a (gcW2) becomes a (gcW1)-transformation, for example in the transformation $\mathtt{letrec}\ x{=}(\mathtt{letrec}\ a{:=}n\ \mathtt{in}\ s)\ \mathtt{in}\ r \xrightarrow{S,gcW2} \mathtt{letrec}\ x{=}s\ \mathtt{in}\ r$ where $\mathtt{letrec}\ x{=}(\mathtt{letrec}\ a{:=}n\ \mathtt{in}\ s)\ \mathtt{in}\ r \xrightarrow{\text{LRPw},llet} \mathtt{letrec}\ x{=}s, a{:=}n\ \mathtt{in}\ r$. The second diagram covers the case where the (gcW)-redex is removed by the normal order reduction, e.g. if it is in an unused alternative of `case` or inside the first argument of `seq`. The last diagram covers the case where the redex of (LRPw,lll) is removed by (gcW2). □

The following lemma describes the base case for the (gcW)-transformation:

Lemma 3.7. *Let $s \xrightarrow{S,gcW} t$. If s is a WHNF, then t is a WHNF; and if t is a WHNF, then $s \xrightarrow{\text{LRPw},llet,0\vee1} s'$ where s' is a WHNF.*

Proposition 3.8. *The transformation (gcW) is correct and for $A_{min} \subseteq A \subseteq \mathfrak{A}$: (gcW) $\subseteq \approx_A$.*

Proof. We first show correctness. Let $s \xrightarrow{S,gcW} t$. For $s{\downarrow_{\text{LRPw}}} \implies t{\downarrow_{\text{LRPw}}}$, we use induction on k in $s \xrightarrow{\text{LRPw},k} s'$ where s' is a WHNF. For $k = 0$, Lemma 3.7 shows $t{\downarrow_{\text{LRPw}}}$. For the induction step, we apply a forking diagram. For the first diagram, we have $s \xrightarrow{\text{LRPw},a} s_1$, $s_1 \xrightarrow{S,gcW}$

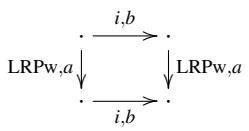
$t_1, t \xrightarrow{\text{LRPw},a} t_1$. The induction hypothesis for s_1 and t_1 shows $t_1{\downarrow_{\text{LRPw}}}$ and thus $t{\downarrow_{\text{LRPw}}}$. For the second diagram, $t{\downarrow_{\text{LRPw}}}$ holds. For the third diagram, we have $s \xrightarrow{\text{LRPw},lll} s_1 \xrightarrow{gcW2} t$. We apply the induction hypothesis to s_1 and t which shows $t{\downarrow_{\text{LRPw}}}$. For $t{\downarrow_{\text{LRPw}}} \implies s{\downarrow_{\text{LRPw}}}$, we use an induction on k in $t \xrightarrow{\text{LRPw},k} t'$ where t' is a WHNF. For $k = 0$, Lemma 3.7 shows $s{\downarrow_{\text{LRPw}}}$. For the induction step, we apply a commuting diagram. For diagrams (1) and (2), the cases are analogous to the previous part. For the third diagram, we apply the diagram as long as possible which terminates, since there are no infinite sequences of $(\text{LRPw},,lll)$-reductions, i.e. we derive s' with either $s \xrightarrow{\text{LRPw},lll,+} s'$ where s' is a WHNF and thus $s{\downarrow_{\text{LRPw}}}$, or we apply the first or second diagram to t and s', and then the induction hypothesis (in case of diagram 1). In any case we derive $s{\downarrow_{\text{LRPw}}}$.

The two parts and the context lemma for $\sim_c$ show that (gcW) is correct. Now we consider improvement. Let $s \xrightarrow{S,gcW} t$. We show $\mathtt{rln}_A(s) = \mathtt{rln}_A(t)$. The context lemma for improvement then implies $(gcW) \subseteq \approx_A$. Since (gcW) is correct, we already have $\mathtt{rln}_A(s) = \infty \iff \mathtt{rln}_A(t) = \infty$. Now let $s{\downarrow_{\text{LRPw}}}s'$ (where $s \xrightarrow{\text{LRPw},k} s'$) and $\mathtt{rln}_A(s) = n$. We show $\mathtt{rln}_A(t) = n$ by induction on k. If $k = 0$, then Lemma 3.7 shows $\mathtt{rln}_A(s) = 0 = \mathtt{rln}_A(t)$. If $k > 0$, then we apply the forking diagrams. The cases are analogous as for the correctness proof, where have to verify, that the first and the second diagram do neither introduce nor remove normal order reductions, and the third diagram may only remove (LRPw,lll)-reductions which are not counted by the $\mathtt{rln}_A$-measure. □

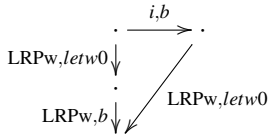
3.3 Properties of the Reduction Rules and Additional Transformations

Proposition 3.9. *All reduction rules are correct.*

Proof. For the (letwn)-rules this is already proved. For the other rules, correctness was shown in the untyped calculus LR in [SSSS08], which can be directly transferred to LRP. However, LRPw has shared-work decorations and the (letwn)-rules as normal order reduction. To keep the proof compact, we only consider these new cases. The reasoning to show correctness of the reduction rules in LRPw is the same as for LR, since all additional diagrams between an internal transformation step (i,b) and a $(\text{LRPw},letw)$-reduction are:



$a \in \{letwn, letw0\}, b \in \{lbeta, cp, case, seq, lll\}$



$b \in \{lbeta, cp, case, seq, lll\}$

In the first case the transformations commute, in the second case the internal transformation becomes a normal order reduction after removing the a label. These cases are already covered by the diagram proofs in LR (see [SSSS08]) and thus can easily be added. □

The following results from [SSSS08, SSS15a] also hold in LRPw, since the overlappings for $(letw)$ and the corresponding transformation are analogous to already covered cases.

Theorem 3.10. *Let t be a closed* LRPw*-expression with $t{\downarrow_{\text{LRPw}}}t_0$ and $A_{min} \subseteq A \subseteq \mathfrak{A}$.*

1. *If $t \xrightarrow{C,\ a} t'$, and $a \in \mathfrak{A}$, then $\mathtt{rln}_A(t) \geq \mathtt{rln}_A(t')$.*

2. *If $t \xrightarrow{C,cp} t'$, then $\mathtt{rln}_A(t) = \mathtt{rln}_A(t')$.*

3. *If $t \xrightarrow{S,a} t'$, and $a \in \mathfrak{A}$, then $\mathtt{rln}_A(t) \geq \mathtt{rln}_A(t')$ and $\mathtt{rln}_A(t') \geq \mathtt{rln}_A(t) - 1$ if $a \in A$, and $\mathtt{rln}_A(t') = \mathtt{rln}_A(t)$ if $a \notin A$.*

4. *If $t \xrightarrow{C,\ a} t'$, and $a \in \{lll, gc\}$, then $\mathtt{rln}_A(t) = \mathtt{rln}_A(t')$.*

5. *If $t \xrightarrow{C,a} t'$, and $a \in \{cpx, xch, cpcx, abs, lwas\}$, then $\mathtt{rln}_A(t) = \mathtt{rln}_A(t')$.*

6. *If $t \xrightarrow{C,\ ucp} t'$, then $\mathtt{rln}_A(t) = \mathtt{rln}_A(t')$.*

Corollary 3.11. *Let $A_{min} \subseteq A \subseteq \mathfrak{A}$. If $s \xrightarrow{S,a} s'$ where a is any rule from Figs. 3 and 4, then $s' \preceq_A s$. If $s \xrightarrow{C,a} s'$ where a is (lll), (cp), (letw0) or any rule of Fig. 4, then $s' \approx_A s$.*

Proof. The claims follow from Theorem 3.10 and the context lemma. For (gcW) this follows from Proposition 3.8. For (letw0) it follows from Proposition 3.4, and for (letwn) it follows from Proposition 3.5. □

4 On the Relationship Between LRPw and LRP

In this section we analyze the relationship between the calculus LRP and its extension LRPw. We show that the calculi are isomorphic w.r.t. the equivalence classes of contextual equivalence. This result is more or less obvious, since adding work decorations to expression does not change their termination behavior. We then investigate the more interesting question whether such an isomorphism also exists w.r.t. the equivalence classes of the improvement relation $\approx_A$. We show that if $(seq) \notin A$, then such an isomorphism exists, while it does not exist if $A = \mathfrak{A}$. For $A = \mathfrak{A}$, we have to leave open the question whether the embedding of LRP into LRPw is conservative w.r.t. $\approx_A$ (i.e. $s \approx_{A,\text{LRP}} t \implies s \approx_{A,\text{LRPw}} t$). We conjecture that this conservativity holds, but we did not find a proof.

4.1 Contextual Equivalence in LRP and LRPw

We define a translation from expressions with work-decorations into decoration-free expressions:

Definition 4.1. *For an* LRPw*-expression t, the expression* $\mathtt{rmw}(t)$ *is derived from t by removing the work-syntax, i.e.* $\mathtt{rmw}(\mathtt{letrec}\ x_1{=}s_1,\ldots,x_n{=}s_n, a_1{:=}n_1,\ldots,a_m{:=}n_m\ \mathtt{in}\ s) = \mathtt{letrec}\ x_1{=}\mathtt{rmw}(s_1),\ldots,x_n{=}\mathtt{rmw}(s_n)\ \mathtt{in}\ \mathtt{rmw}(s)$*, for $m \geq 0$ and $n \geq 1$;* $\mathtt{rmw}(s^{[a]}) = \mathtt{rmw}(s)$*;* $\mathtt{rmw}(\mathtt{letrec}\ a_1{:=}n_1,\ldots,a_m{:=}n_m\ \mathtt{in}\ s) = \mathtt{rmw}(s)$*; and for all other language constructs f:* $\mathtt{rmw}(f[s_1,\ldots,s_n]) = f[\mathtt{rmw}(s_1),\ldots,\mathtt{rmw}(s_n)]$.

Since $t \xrightarrow{\text{LRPw}} t'$ implies $\mathtt{rmw}(t) = \mathtt{rmw}(t')$ or $\mathtt{rmw}(t) \xrightarrow{\text{LRPw}} \mathtt{rmw}(t')$, we have:

Proposition 4.2. *Let t be an expression in* LRPw*, then* $t\downarrow_{\text{LRPw}} \iff \mathtt{rmw}(t)\downarrow_{\text{LRPw}}$.

An immediate consequence is the following theorem, where it is necessary to observe that for any LRPw-context C and LRPw-expression s: $\mathtt{rmw}(C[s]) = \mathtt{rmw}(C)[\mathtt{rmw}(s)]$ and $\mathtt{rmw}(C)$ is also an LRP-context.

Theorem 4.3. *The embedding of* LRP *into* LRPw *w.r.t. $\sim_c$ is conservative and the calculi* LRP *and* LRPw *are isomorphic*

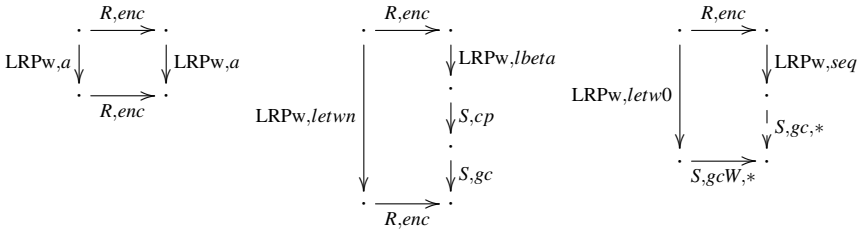
4.2 Comparing the Improvement Relations of LRP and LRPw

We show that the embedding of LRP into LRPw is an isomorphism w.r.t. $\approx_A$ if $seq \notin A$. Let (enc) be the transformation

$$\mathtt{letrec}\ a{:=}n, Env\ \mathtt{in}\ s \xrightarrow{enc} \mathtt{letrec}\ x_a{:=}\mathtt{id}^{n+1}, Env[\mathtt{seq}\ x_a\ t/t^{[a]}]\ \mathtt{in}\ s[\mathtt{seq}\ x_a\ t/t^{[a]}]$$

where $[\mathtt{seq}\ x_a\ t/t^{[a]}]$ means that every subterm $t^{[a]}$ is replaced by $\mathtt{seq}\ x_a\ t$, and $\mathtt{id}^k$ abbreviates $\underbrace{\mathtt{id}\ \ldots\ \mathtt{id}}_{k}$, and $\mathtt{id} = \lambda x.x$.

Lemma 4.4. *Complete sets of forking and commuting diagrams for (R, enc) are:*



Lemma 4.5. *If $s \xrightarrow{R,enc} t$ then s is a WHNF iff t is a WHNF.*

Proposition 4.6. *Let $A_{min} \subseteq A \subset \mathfrak{A}$, s.t. $seq \notin A$. Then $(enc) \subseteq \approx_A$.*

Proof. We show correctness using the context lemma. Let $s \xrightarrow{R,enc} t$. We prove $s\downarrow_{\text{LRPw}} \implies t\downarrow_{\text{LRPw}}$ by induction on k in $s \xrightarrow{\text{LRPw},k} s_k$ where s_k is a WHNF. The case $k = 0$ holds by Lemma 4.5. For the induction step, we apply a forking diagram to $t \xleftarrow{R,enc} s \xrightarrow{\text{LRPw}} s_1$:

For the first diagram, we have $t \xrightarrow{\text{LRPw}} t_1 \xleftarrow{\text{LRPw}} s_1$. Since $s_1 \xrightarrow{\text{LRPw},k-1} s_k$, the induction hypothesis shows $t_1{\downarrow_{\text{LRPw}}}$ and thus $t{\downarrow_{\text{LRPw}}}$. For the second diagram, we have $t \xrightarrow{\text{LRPw},lbeta} t' \xrightarrow{C,cp} \xrightarrow{C,gc} t_1$ s.t. $s_1 \xrightarrow{enc} t_1$. The induction hypothesis shows $t_1{\downarrow_{\text{LRPw}}}$ and correctness of (cp) and (gc) shows $t'{\downarrow_{\text{LRPw}}}$ and thus $t{\downarrow_{\text{LRPw}}}$. For the last diagram, we have $t \xrightarrow{\text{LRPw},seq} t' \xrightarrow{C,gc,*} \xleftarrow{C,gcW,*} s_1$. Correctness of (gc) and (gcW) implies $t'{\downarrow_{\text{LRPw}}}$ and thus $t{\downarrow_{\text{LRPw}}}$.

We prove $t{\downarrow_{\text{LRPw}}} \implies s{\downarrow_{\text{LRPw}}}$ by induction on $(\mathtt{rln}_{\mathfrak{A}}(t),k)$ where $t \xrightarrow{\text{LRPw},k} t_k$ and t_k is a WHNF. For the case $(0,0)$, Lemma 4.5 shows the claim. For the induction step, we apply a commuting diagram to $s \xrightarrow{R,enc} t \xrightarrow{\text{LRPw}} t_1$. If $\mathtt{rln}_{\mathfrak{A}}(t) = 0$, but $k > 0$, then only the first diagram is applicable. For the first diagram, we have $s \xrightarrow{\text{LRPw}} s_1$ s.t. $s_1 \xrightarrow{\text{LRPw}} t_1$. Since $t_1 \xrightarrow{\text{LRPw},k-1} t_k$, and $\mathtt{rln}_{\mathfrak{A}}(t_1) \leq \mathtt{rln}_{\mathfrak{A}}(t)$, the induction hypothesis shows $s_1{\downarrow_{\text{LRPw}}}$ and thus $s{\downarrow_{\text{LRPw}}}$. For the second diagram, we have $s \xrightarrow{\text{LRPw},letwn} s' \xrightarrow{R,enc} s'' \xleftarrow{S,gc} \xleftarrow{S,cp} t_1$. Then $\mathtt{rln}_{\mathfrak{A}}(t_1) < \mathtt{rln}_{\mathfrak{A}}(t)$ and by Theorem 3.10 $\mathtt{rln}_{\mathfrak{A}}(s'') < \mathtt{rln}_{\mathfrak{A}}(t)$. Thus the induction hypothesis applied to s'' shows $s'{\downarrow_{\text{LRPw}}}$ and thus $s{\downarrow_{\text{LRPw}}}$. For the third diagram, we have $s \xrightarrow{\text{LRPw},letw0} s' \xrightarrow{S,gcW,*} \xleftarrow{S,gc,*} t_1$. Correctness of (letw0), (gcW) and (gc) shows $s{\downarrow_{\text{LRPw}}}$.

For proving $(enc) \subseteq \approx_A$, we use Lemma 2.8 and show that if $s \xrightarrow{R,enc} t$, then $\mathtt{rln}_A(s) = \mathtt{rln}_A(t)$. Clearly, $\mathtt{rln}_A(s) = \infty \iff \mathtt{rln}_A(t) = \infty$. So let $s \xrightarrow{\text{LRPw},k} s_k$ where s_k is a WHNF. By induction on k, we show $\mathtt{rln}_A(s) = \mathtt{rln}_A(t)$. If $k = 0$, then Lemma 4.5 implies that t is a WHNF and $\mathtt{rln}_A(s) = 0 = \mathtt{rln}_A(t)$. If $k > 0$, then we apply a forking diagram to $s_1 \xleftarrow{\text{LRPw}} s \xrightarrow{R,enc} t$. For the first diagram, we have $t \xrightarrow{\text{LRPw}} t_1$ s.t. $s_1 \xrightarrow{\text{LRPw}} t_1$. Since $s_1 \xrightarrow{\text{LRPw},k-1} s_k$, the induction hypothesis shows $\mathtt{rln}_A(s_1) = \mathtt{rln}_A(t_1)$ and thus $\mathtt{rln}_A(s) = \mathtt{rln}_A(t)$. For the second diagram, we have $t \xrightarrow{\text{LRPw},lbeta} t' \xrightarrow{C,cp} \xrightarrow{C,gc} t_1$ s.t. $s_1 \xrightarrow{enc} t_1$. Clearly, $\mathtt{rln}_A(s) = 1 + \mathtt{rln}_A(s_1)$ and $\mathtt{rln}_A(t) = 1 + \mathtt{rln}_A(t')$. The induction hypothesis shows $\mathtt{rln}_A(s_1) = \mathtt{rln}_A(t_1)$ and Theorem 3.10 shows $\mathtt{rln}_A(t_1) = \mathtt{rln}_A(t')$. For the last diagram, we have $t \xrightarrow{\text{LRPw},seq} t' \xrightarrow{C,gc,*} \xleftarrow{C,gcW,*} s_1$. Then $\mathtt{rln}_A(s) = \mathtt{rln}_A(s_1)$ and (since $seq \notin A$) $\mathtt{rln}_A(t) = \mathtt{rln}_A(t')$. Finally, Theorem 3.10 and Proposition 3.8 show $\mathtt{rln}_A(t') = \mathtt{rln}_A(s_1)$. □

Theorem 4.7. *Let $A_{min} \subseteq A \subset \mathfrak{A}$, s.t. $seq \notin A$. Then every decorated expression s can be represented as an* LRP*-expression s' with $s \approx_A s'$. This means the embedding of* LRP *into* LRPw *is an isomorphism w.r.t. $\approx_A$.*

Proof. It suffices to show that $s \approx_{\text{LRP},A} t$ implies $s \approx_{\text{LRPw},A} t$. Let $s \approx_{\text{LRP},A} t$ and let C be an LRPw-context. Then $\mathtt{rln}_A(C[s]) = \mathtt{rln}_A(C[t])$ in LRPw and thus $s \approx_{\text{LRPw},A} t$: We apply (enc)-transformations to $C[s]$ and $C[t]$ (without changing s,t, since they do neither contain $a{:=}n$ labels nor $\cdot^{[a]}$ labels.) until we get expressions $C'[s]$, $C'[t]$ s.t. both are free of bindings $a{:=}n$ and labels $\cdot^{[a]}$. We have $C'[s] \approx_A C[s]$ and $C'[t] \approx_A C[t]$ by Proposition 4.6 and thus $\mathtt{rln}_A(C'[s]) = \mathtt{rln}_A(C[s])$ and $\mathtt{rln}_A(C'[t]) = \mathtt{rln}_A(C[t])$. Since C' is an LRP-context, the precondition $s \approx_{\text{LRP},A} t$ shows $\mathtt{rln}_A(C[s]) = \mathtt{rln}_A(C[t])$ which shows the claim. □

We show that the isomorphism property w.r.t. $\approx_A$ does not hold for $A = \mathfrak{A}$:

Proposition 4.8. *Let $A = \mathfrak{A}$ and let c_1 and c_2 be different constants. The expression* `letrec` $a{:=}1$ `in` $(\texttt{Pair}\ c_1^{[a]}\ c_2^{[a]})$ *is not equivalent w.r.t. $\approx_A$ to any* LRP*-expression.*

Proof. Assume there is such an expression s. Then $s \sim_c (\texttt{Pair}\ c_1\ c_2)$ and $\texttt{rln}_A(s) = 0$, so we can assume that s is a WHNF. Using correctness w.r.t. $\approx_A$ of program transformations and $c_1 \not\sim_c c_2$, we can assume that s is of the form $\texttt{letrec}\ x{=}s_1, y{=}s_2, Env\ \texttt{in}\ (\texttt{Pair}\ x\ y)$. We see that s_1 as well as s_2 alone have $\texttt{rln}_A$-count 1 in the environment. Using invariance of (lll), (cpx) and (gc) w.r.t. $\approx_A$, we can assume that s_1, s_2 are applications, `seq`- or case-expressions, and evaluation of them requires at least one $\texttt{rln}_A$-reduction that is independent of the other to become a WHNF. Hence, the context $\texttt{let}\ z{=}[\cdot]\ \texttt{in}\ \texttt{seq}\ (\texttt{fst}\ z)\ (\texttt{snd}\ z)$ applied to $\texttt{letrec}\ a{:=}1\ \texttt{in}\ (\texttt{Pair}\ c_1^{[a]}\ c_2^{[a]})$ requires $6 = 5+1$ $\texttt{rln}_A$-steps: 2 for `fst`, 2 for `snd`, 1 for `seq`, and 1 for evaluating $a{:=}1$, whereas s requires at least $5+2$: the 2 reductions are the minimum to reach a WHNF for the first as well as for the second component. □

Even though the calculi are not isomorphic w.r.t. $\approx_{\mathfrak{A}}$, it may be the case that the embedding of LRP into LRPw w.r.t. $\approx_{\mathfrak{A}}$ is conservative (i.e. for all LRP-expressions s,t : $s \approx_{\mathfrak{A},\text{LRP}} t \implies s \approx_{\mathfrak{A},\text{LRPw}} t$). We conjecture that this conservativity holds, but we were unable to show it, since we did not find a proof. A simple approach to encode LRPw-expressions and -contexts into LRP-expressions and -contexts fails due to the example given in Proposition 4.8. So conservativity remains an open problem. Conservativity would allow to lift results on improvements from LRP to LRPw more easily. However, the following lemma shows that $\approx_A$-relations proved LRPw can be used for reasoning on $\approx_A$ in LRP. It holds, since every LRP-context is also an LRPw-context and on decoration-free expressions the `rln`-length is the same in both calculi.

Lemma 4.9. *Let $A_{min} \subseteq A \subseteq \mathfrak{A}$. Let s,t be* LRP*-expressions s.t. $s \preceq_{A,\text{LRPw}} t$. Then also $s \preceq_{A,\text{LRP}} t$ holds.*

5 Computation Rules for Work Decorations

In this section we develop computation rules for work decorations which allow to shift-up or merge such decorations. Such computation rules are a helpful tool when reasoning on improvements. As a first step we introduce a notation for (unshared) work decorations:

Definition 5.1. *If $n \in \mathbb{N}$, then we write $s^{[n]}$ where $[n]$ is called a* (unshared) work decoration*. The semantics of $s^{[n]}$ is* $\texttt{letrec}\ a{:=}n\ \texttt{in}\ s^{[a]}$ *where a is a fresh label.*

We show that these work-decorations are redundant:

Proposition 5.2. *Work decorations $s^{[n]}$ can be encoded as* $\texttt{letrec}\ x{=}(id^n)\ \texttt{in}\ (x\ s)$.

Proof. Let $A_{min} \subseteq A \subseteq \mathfrak{A}$. We show $s^{[n]} = \texttt{letrec}\, a{:=}n\, \texttt{in}\, s^{[a]} \approx_A \texttt{letrec}\, x{=}(id^n)\, \texttt{in}\, (x\ s)$: By the context lemmas for $\sim_c$ and $\approx_A$, it suffices to show $r_1 := R[\texttt{letrec}\ a{:=}n\ \texttt{in}\ s^{[a]}] \sim_c R[\texttt{letrec}\ x{=}(id^n)\ \texttt{in}\ (x\ s)] := r_2$ and $\texttt{rln}_A(r_1) = \texttt{rln}_A(r_2)$ for all reduction contexts

R, A case analysis of the structure of context R, shows that there always is a term r_3 s.t. $r_1 \xrightarrow{\text{LRPw},lll,*} \xrightarrow{\text{LRPw},letwn,n} \xrightarrow{\text{LRPw},letw0} \xrightarrow{gcW} r_3$ and $r_2 \xrightarrow{\text{LRPw},lll,*} (\xrightarrow{\text{LRPw},lbeta} \xrightarrow{\text{LRPw},llet})^{n-1} \xrightarrow{\text{LRPw},cp} \xrightarrow{\text{LRPw},lbeta} \xrightarrow{\text{LRPw},lll,*} \xrightarrow{ucp} \xrightarrow{gc} r_3$. which shows $r_1 \sim_c r_3 \sim_c r_2$ and $\mathtt{rln}_A(r_1) = n + \mathtt{rln}_A(r_3) = \mathtt{rln}_A(r_2)$, since (ucp), (gc), and (gcW) are invariant w.r.t. $\approx$. □

A corollary from the theorem on reduction lengths (Theorem 3.10) is:

Corollary 5.3. *Let $A_{min} \subseteq A \subseteq \mathfrak{A}$ and let S be a surface context. If $s \xrightarrow{S} s'$ by any reduction or transformation rule from Figs. 3 and 4, then $s' \preceq_A s$ and $s \preceq_A s'^{[1]}$.*

The following theorem summarizes several computation rules for work decorations. It will be proved in the remainder of this section in a series of lemmas.

Theorem 5.4. *Let $A_{min} \subseteq A \subseteq \mathfrak{A}$.*

1. If $s \xrightarrow{\text{LRPw},a} t$ with $a \in A$, then $s \approx_A t^{[1]}$, and if $a \notin A$, then $s \approx_A t$.

2. $R[\mathtt{letrec}\ a{:=}n\ \mathtt{in}\ s^{[a]}] \approx_A \mathtt{letrec}\ a{:=}n\ \mathtt{in}\ R[s]^{[a]}$ and thus $R[s^{[n]}] \approx_A R[s]^{[n]}$.

3. $\mathtt{rln}_A(\mathtt{letrec}\ a{:=}n\ \mathtt{in}\ s^{[a]}) = n + \mathtt{rln}_A(s')$ where s' is s where all $^{[a]}$-labels are removed. In particular this also shows $\mathtt{rln}_A(s^{[n]}) = n + \mathtt{rln}_A(s)$.

4. For every reduction context R: $\mathtt{rln}_A(R[\mathtt{letrec}\ a{:=}n\ \mathtt{in}\ s^{[a]}]) = n + \mathtt{rln}_A(R[s'])$ where s' is s where all $^{[a]}$-labels are removed. In particular, $\mathtt{rln}_A(R[s^{[n]}]) = n + \mathtt{rln}_A(R[s])$.

5. $(s^{[n]})^{[m]} \approx_A s^{[n+m]}$

6. $S[\mathtt{letrec}\ a{:=}n\ \mathtt{in}\ T[s^{[a]}]] \preceq_A \mathtt{letrec}\ a{:=}n\ \mathtt{in}\ S[T[s]]^{[a]}$ holds for all surface contexts S,T, and if $S[T]$ is strict, also $S[\mathtt{letrec}\ a{:=}n\ \mathtt{in}\ T[s^{[a]}]] \approx_A \mathtt{letrec}\ a{:=}n\ \mathtt{in} S[T[s]]^{[a]}$.

7. $\mathtt{letrec}\ a{:=}n, b{:=}m\ \mathtt{in}\ (s^{[a]})^{[b]} \approx_A \mathtt{letrec}\ a{:=}n, b{:=}m\ \mathtt{in}\ (s^{[b]})^{[a]}$

8. $\mathtt{letrec}\ a{:=}n\ \mathtt{in}\ (s^{[a]})^{[a]} \approx_A \mathtt{letrec}\ a{:=}n\ \mathtt{in}\ (s^{[a]})$

9. Let $S[\cdot,\ldots,\cdot]$ be a multi-context where all holes in S are in surface position. Then $\mathtt{letrec}\ a{:=}n\ \mathtt{in}\ S[s_1^{[a]},\ldots,s_n^{[a]}] \preceq_A \mathtt{letrec}\ a{:=}n\ \mathtt{in}\ S[s_1,\ldots,s_n]^{[a]}$. If some hole in S is in strict position, then $\mathtt{letrec}\ a{:=}n\ \mathtt{in}\ S[s_1^{[a]},\ldots,s_n^{[a]}] \approx_A \mathtt{letrec}\ a{:=}n\ \mathtt{in}\ S[s_1,\ldots,s_n]^{[a]}$.

Proof. Item (1) is proved in Theorem 5.11. Item (2) is proved in Proposition 5.6. Item (3) is proved in Lemma 5.7. Item (4) is proved in Corollary 5.8. Item (5) is proved in Proposition 5.9. Item (6) is proved in Corollary 5.15. Items(7) and (8) are proved in Proposition 5.16. Item (9) is proved in Proposition 5.17. □

Lemma 5.5. *Let $A_{min} \subseteq A \subseteq \mathfrak{A}$. If $s \xrightarrow{\text{LRPw},letwn} t$, then $s \approx_A t^{[1]}$*

Proof. We use the context lemma for improvement and thus have to show for all reduction contexts R: $\mathtt{rln}_A(R[s]) = \mathtt{rln}_A(R[\mathtt{letrec}\ a{:=}1\ \mathtt{in}\ t^{[a]}])$. A case analysis on the reduction context R shows that there exists an expression r s.t. $R[s] \xrightarrow{\text{LRPw},lll,*} \xrightarrow{\text{LRPw},letwn} r$ and $R[\mathtt{letrec}\ a{:=}1\ \mathtt{in}\ t^{[a]}] \xrightarrow{\text{LRPw},lll,*} \xrightarrow{\text{LRPw},letwn} \xrightarrow{\text{LRPw},letw0} \xrightarrow{gcW} \xrightarrow{\text{LRPw},lll,*} r$. Since all rules except of $(\text{LRPw}, letwn)$ leave the $\mathtt{rln}_A$-measure unchanged, the claim is proved. □

Proposition 5.6. *$R[\mathtt{letrec}\ a{:=}n\ \mathtt{in}\ s^{[a]}] \approx_A \mathtt{letrec}\ a{:=}n\ \mathtt{in}\ R[s]^{[a]}$ for $A_{min} \subseteq A \subseteq \mathfrak{A}$.*

Proof. The equation $R[\mathtt{letrec}\ a{:=}n\ \mathtt{in}\ s^{[a]}] \approx_A \mathtt{letrec}\ a{:=}n\ \mathtt{in}\ R[s]^{[a]}$ holds by induction on n: The case $n = 0$ holds, since $(letw0) \subseteq \approx_A$. For the induction step, a case distinction of the context R, Lemma 5.5, the induction hypothesis, and $(lll) \subseteq \approx_A$ show $R[\mathtt{letrec}\ a{:=}n\ \mathtt{in}\ s^{[a]}] \approx_A \mathtt{letrec}\ a{:=}n\ \mathtt{in}\ R[s]^{[a]}$ for $n > 0$. □

Lemma 5.7. *Let $A_{min} \subseteq A \subseteq \mathfrak{A}$. Then $\mathtt{rln}_A(\mathtt{letrec}\ a{:=}n\ \mathtt{in}\ s^{[a]}) = n + \mathtt{rln}_A(s')$ where s' is s where all $^{[a]}$-labels are removed. In particular this shows $\mathtt{rln}_A(s^{[n]}) = n + \mathtt{rln}_A(s)$.*

Proof. The reduction $\mathtt{letrec}\ a{:=}n\ \mathtt{in}\ s^{[a]} \xrightarrow{\text{LRPw},letwn,n} \xrightarrow{C,letw0,*} \mathtt{letrec}\ a{:=}0\ \mathtt{in}\ s'$ shows $\mathtt{rln}_A(s^{[n]}) = n + \mathtt{rln}_A(\mathtt{letrec}\ a{:=}0\ \mathtt{in}\ s)$. Finally, $(gcW) \subseteq \approx_A$ shows the claim. □

Corollary 5.8. *For $A_{min} \subseteq A \subseteq \mathfrak{A}$, the equation $\mathtt{rln}_A(R[\mathtt{letrec}\ a{:=}n\ \mathtt{in}\ s^{[a]}]) = n + \mathtt{rln}_A(R[s'])$ holds for all reduction contexts R, where s' is s where all $^{[a]}$-labels are removed. In particular, this shows $\mathtt{rln}_A(R[s^{[n]}]) = n + \mathtt{rln}_A(R[s])$.*

Proposition 5.9. *Let $A_{min} \subseteq A \subseteq \mathfrak{A}$. Then $(s^{[n]})^{[m]} \approx_A s^{[n+m]}$.*

Proof. Clearly, $(s^{[n]})^{[m]} \sim_c s^{[n+m]}$. Since Corollary 5.8 implies $\mathtt{rln}_A(R[(s^{[n]})^{[m]}]) = m + n + \mathtt{rln}_A(R[s]) = \mathtt{rln}_A(R[s^{[n+m]}])$ for all reduction contexts R, the context lemma for improvement shows the claim. □

Lemma 5.10. *If $s \xrightarrow{\text{LRPw},a} t$ and $a \in \{lbeta, case\text{-}c, seq\text{-}c\}$, then $s \approx_A t^{[1]}$ if $a \in A$.*

Proof. By the context lemma for improvement it suffices to show for all reduction contexts R: $\mathtt{rln}_A(R[s]) = \mathtt{rln}_A(R[t^{[1]}])$, if $a \in A$. By Corollary 5.8 we have $\mathtt{rln}_A(R[t^{[1]}]) = 1 + \mathtt{rln}_A(R[t])$ and thus it suffices to show $\mathtt{rln}_A(R[s]) = 1 + \mathtt{rln}_A(R[t])$. Let $s_0 \xrightarrow{a} t_0$ for $a \in \{\text{lbeta}, \text{case-c}, \text{seq-c}\}$ and assume $a \in A$. We verify all cases:

If $s = R_0^-[s_0]$ and $t = R_0^-[t_0]$ for a weak reduction context R_0^-, then $R[s] \xrightarrow{\text{LRPw},a} R[t]$.

If $s = \mathtt{letrec}\ Env\ \mathtt{in}\ R_0^-[s_0]$ and $t = \mathtt{letrec}\ Env\ \mathtt{in}\ R_0^-[t_0]$ where R_0^- is a weak reduction context, then we consider the cases for the context R: If R is a weak reduction context, then $R[s] \xrightarrow{\text{LRPw},lll,*} \mathtt{letrec}\ Env\ \mathtt{in}\ R[R_0^-[s_0]] \xrightarrow{\text{LRPw},a} \mathtt{letrec}\ Env\ \mathtt{in}\ R[R_0^-[t_0]] \xleftarrow{\text{LRPw},lll,*} R[t]$. If $R = \mathtt{letrec}\ Env'\ \mathtt{in}\ R'$ or $R = \mathtt{letrec}\ Env', u{=}R'\ \mathtt{in}\ r$, for a weak reduction context R', then $\mathtt{rln}(R[s]) = \mathtt{rln}(R[\mathtt{letrec}\ a{:=}1\ \mathtt{in}\ t^{[a]}])$, since $R[s] \xrightarrow{\text{LRPw},lll,*} \xrightarrow{\text{LRPw},a} \xleftarrow{C,lll,*} R[t]$.

Finally, if $s = \texttt{letrec}\ Env, y{=}R_0^-[s_0]\ \texttt{in}\ u_0$ and $t = \texttt{letrec}\ Env, y{=}R_0^-[t_0]\ \texttt{in}\ u_0$ where R_0^- is a weak reduction context, then a case analysis on the structure of R again shows that $R[s] \xrightarrow{\text{LRPw},lll,*} \xrightarrow{\text{LRPw},a} \xleftarrow{C,lll,*} R[t]$ which shows the claim. □

Theorem 5.11. *Let $A_{min} \subseteq A \subseteq \mathfrak{A}$. If $s \xrightarrow{\text{LRPw},a} t$ with $a \in A$, then $s \approx t^{[1]}$.*

Proof. For (letwn) the claim is Lemma 5.5, for (case-c), (seq-c), and (lbeta) the claim is Lemma 5.10. Other (case)- and (seq)-reductions can be expressed by a (case-c)- or (seq-c)-reduction and (cpcx)-, (gc)-, and (lll)-transformations which are invariant w.r.t. $\approx_A$. □

Proposition 5.12. *For A with $A_{min} \subseteq A \subseteq \mathfrak{A}$. and any strict surface context S the equation $S[\texttt{letrec}\ a{:=}n\ \texttt{in}\ s^{[a]}] \approx_A \texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s]^{[a]}$ holds. In particular, $S[s^{[n]}] \approx_A S[s]^{[n]}$.*

Proof. If $S[r] \sim_c \bot$ for all r, then $S[\texttt{letrec}\ a{:=}n\ \texttt{in}\ s^{[a]}] \sim_c \bot \sim_c \texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s]^{[a]}$. Since for any reduction context R, the expressions $R[\bot]$, $R[S[\texttt{letrec}\ a{:=}n\ \texttt{in}\ s^{[a]}]]$, and $R[\texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s]^{[a]}]$ diverge, the equation $\texttt{rln}_A(R[\texttt{letrec}\ a{:=}n\ \texttt{in}\ s^{[a]}]) = \infty = \texttt{rln}_A(R[\texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s]^{[a]}])$ holds. Now the context lemma for improvement shows $S[\texttt{letrec}\ a{:=}n\ \texttt{in}\ s^{[a]}] \approx_A \texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s]^{[a]}$.

Otherwise, verify that for all r reduction contexts R a reduction $R[S[r]] \xrightarrow{\text{LRPw},k} R'[r]$ exists, where R' is a reduction context and $\texttt{rln}_A(R[S[r]] \xrightarrow{\text{LRPw},k} R'[r]) = m$ for some $m \leq k$. For $R[S[\texttt{letrec}\ a{:=}n\ \texttt{in}\ s^{[a]}]]$, the equation $\texttt{rln}_A(R[S[\texttt{letrec}\ a{:=}n\ \texttt{in}\ s^{[a]}]]) = m + \texttt{rln}_A(R'[\texttt{letrec}\ a{:=}n\ \texttt{in}\ s^{[a]}])$ holds. Applying Proposition 5.6 shows that the equation $\texttt{rln}_A(R'[\texttt{letrec}\ a{:=}n\ \texttt{in}\ s^{[a]}]) = \texttt{rln}_A(\texttt{letrec}\ a{:=}n\ \texttt{in}\ R'[s]^{[a]})$ holds. By Lemma 5.7 we have $\texttt{rln}_A(\texttt{letrec}\ a{:=}n\ \texttt{in}\ R'[s]^{[a]}) = n + \texttt{rln}(R'[s'])$ where s' is s where all $^{[a]}$-labels are removed. Thus $\texttt{rln}_A(R[S[\texttt{letrec}\ a{:=}n\ \texttt{in}\ s^{[a]}]]) = m + n + \texttt{rln}_A(R'[s'])$ For $R[\texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s]^{[a]}]$, we have by Corollary 5.8 $\texttt{rln}_A(R[\texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s]^{[a]}]) = n + \texttt{rln}_A(R[S[s']])$ where s' is s where all $^{[a]}$-labels are removed. Since $\texttt{rln}_A(R[S[s']]) = m + \texttt{rln}_A(R'[S[s']])$, we have $\texttt{rln}_A(R[\texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s]^{[a]}]) = n + m + \texttt{rln}_A(R'[S[s']])$. Thus we have shown $\texttt{rln}_A(R[S[\texttt{letrec}\ a{:=}n\ \texttt{in}\ s^{[a]}]]) = \texttt{rln}_A(R[\texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s]^{[a]}])$. Now the context lemma for improvement shows the claim. □

Proposition 5.13. *For all A with $A_{min} \subseteq A \subseteq \mathfrak{A}$ and all surface contexts S the inequations $S[\texttt{letrec}\ a{:=}n\ \texttt{in}\ s^{[a]}] \preceq_A \texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s]^{[a]}$ and $S[s^{[n]}] \preceq_A S[s]^{[n]}$ hold.*

Proof. Let R be a reduction context. If $R[S]$ is strict, then by applying Proposition 5.12 we derive $\texttt{rln}_A(R[S[\texttt{letrec}\ a{:=}n\ \texttt{in}\ s^{[a]}]]) \leq \texttt{rln}_A(R[\texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s]^{[a]}])$.

If $R[S]$ is non-strict, then $\texttt{rln}_A(R[S[r]]) = m_R$ for any R and where m_R only depends on the context $R[S]$. Then $\texttt{rln}_A(R[S[\texttt{letrec}\ a{:=}n\ \texttt{in}\ s^{[a]}]]) = m_R$. From Corollary 5.8 we have $\texttt{rln}_A(R[\texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s]^{[a]}]) = n + \texttt{rln}_A(R[S[s']])$ where s' is s where all $^{[a]}$-labels are removed. Thus, the equation $\texttt{rln}_A(R[\texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s]^{[a]}]) = n + m_R$ holds. Since $S[\texttt{letrec}\ a{:=}n\ \texttt{in}\ s^{[a]}] \sim_c \texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s]^{[a]}$ (by correctness of $(letw)$ and (gcW)), the context lemma for improvement shows the claim. □

Proposition 5.14. *Let $A_{min} \subseteq A \subseteq \mathfrak{A}$ and S be a surface context. Then the inequation* $\texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s^{[a]}] \preceq_A \texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s]^{[a]}$ *holds, and if S is strict, then the equation* $\texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s^{[a]}] \approx_A \texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s]^{[a]}$ *holds.*

Proof. First assume that S is strict. Then $S' := R[\texttt{letrec}\ a{:=}n\ \texttt{in}\ S[\cdot]$ for all reduction contexts R is also strict. If $S'[r] \sim_c \bot$ for all r, then $\texttt{rln}_A(R[\texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s^{[a]}])) = \infty$ and $\texttt{rln}_A(R[\texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s])^{[a]}) = n + \texttt{rln}_A(R[S[s]]) = n + \infty = \infty$.

Now assume that S is not strict. Let R be a reduction context. By (lll)-transformations we have $R[\texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s^{[a]}]] \approx_A \texttt{letrec}\ a{:=}n\ \texttt{in}\ R[S[s^{[a]}]$. If $R[S[\cdot]]$ is strict, then we have $\texttt{letrec}\ a{:=}n\ \texttt{in}\ R[S[s^{[a]}]] \approx_A \texttt{letrec}\ a{:=}n\ \texttt{in}\ R[S[s]]^{[a]}$ (since $R[S[\cdot]]$ is a strict surface context) and $\texttt{letrec}\ a{:=}n\ \texttt{in}\ R[S[s]]^{[a]} \approx_A \texttt{letrec}\ a{:=}n\ \texttt{in}\ R[S[s]^{[a]}]$. By (lll)-transformations we have $\texttt{letrec}\ a{:=}n\ \texttt{in}\ R[S[s]^{[a]}] \approx_A R[\texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s]^{[a]}]$ and $\texttt{rln}_A(R[\texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s^{[a]}]]) = \texttt{rln}_A(R[\texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s]^{[a]}])$. If $R[S[\cdot]]$ is non-strict, then $\texttt{rln}_A(R[S[r]]) = m_R$ for any r and where m_R only depends on $R[S]$. Then $\texttt{rln}_A(R[\texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s^{[a]}]]) = \texttt{rln}_A(\texttt{letrec}\ a{:=}n\ \texttt{in}\ R[S[s^{[a]}]]]) = m_R$. We also have $\texttt{rln}_A(R[\texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s]^{[a]}]) = n + \texttt{rln}_A(R[S[s]]) = n + m_R$ by Corollary 5.8.

Thus, in any case $\texttt{rln}_A(R[\texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s^{[a]}]]) \leq \texttt{rln}_A(R[\texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s]^{[a]}])$ and the context lemma for improvement shows the claim. □

Corollary 5.15. *Let $A_{min} \subseteq A \subseteq \mathfrak{A}$ and let S_1, S_2 be surface contexts. Then the inequation* $S_1[\texttt{letrec}\ a{:=}n\ \texttt{in}\ S_2[s^{[a]}]] \preceq_A \texttt{letrec}\ a{:=}n\ \texttt{in}\ S_1[S_2[s]]^{[a]}$ *holds and if $S_1[S_2]$ is strict, also the equation* $S_1[\texttt{letrec}\ a{:=}n\ \texttt{in}\ S_2[s^{[a]}]] \approx_A \texttt{letrec}\ a{:=}n\ \texttt{in}\ S_1[S_2[s]]^{[a]}$.

Proposition 5.16. *The equations* $\texttt{letrec}\ a{:=}n\ \texttt{in}\ (s^{[a]})^{[a]} \approx_A \texttt{letrec}\ a{:=}n\ \texttt{in}\ (s^{[a]})$ *and* $\texttt{letrec}\ a{:=}n, b{:=}m\ \texttt{in}\ (s^{[a]})^{[b]} \approx_A \texttt{letrec}\ a{:=}n, b{:=}m\ \texttt{in}\ (s^{[b]})^{[a]}$ *hold for all A with $A_{min} \subseteq A \subseteq \mathfrak{A}$.*

Proof. For the first part, the expressions are clearly contextually equivalent. We have $\texttt{rln}_A(R[\texttt{letrec}\ a{:=}n\ \texttt{in}\ (s^{[a]})^{[a]}]) = n + \texttt{rln}_A(R[s']) = \texttt{rln}_A(R[\texttt{letrec}\ a{:=}n\ \texttt{in}\ (s^{[a]})])$ by Corollary 5.8, for all reduction context R, where s' is s where all $^{[a]}$-labels are removed. Now the context lemma for improvement shows the claim. For the second part, let R be a reduction context. Since $(llet) \subseteq \approx_A$, the equation $R[\texttt{letrec}\, a{:=}n, b{:=}m\, \texttt{in}\, (s^{[a]})^{[b]}] \approx_A R[\texttt{letrec}\ b{:=}m\ \texttt{in}\ (\texttt{letrec}\ a{:=}n\ \texttt{in}\ (s^{[a]})^{[b]}]$ holds. Applying Corollary 5.8 two times implies $\texttt{rln}_A(R[\texttt{letrec}\, b{:=}m\, \texttt{in}\, (\texttt{letrec}\, a{:=}n\, \texttt{in}\, (s^{[a]})^{[b]})) = m + n \cdot \texttt{rln}_A(R[s''])$ where s'' is s where the labels $^{[a]}$ and $^{[b]}$ are removed. Completely analogously it can be shown that also the equation $\texttt{rln}_A(R[\texttt{letrec}\ a{:=}n, b{:=}m\ \texttt{in}\ (s^{[b]})^{[a]}]) = n + m + \texttt{rln}_A(R[s''])$ holds. Clearly, $\texttt{letrec}\ a{:=}n, b{:=}m\ \texttt{in}\ (s^{[a]})^{[b]} \sim_c \texttt{letrec}\ a{:=}n, b{:=}m\ \texttt{in}\ (s^{[b]})^{[a]}$. Thus the context lemma for improvement shows the claim. □

Repeated application of Corollary 5.15 and Proposition 5.16 shows:

Proposition 5.17. *Let $A_{min} \subseteq A \subseteq \mathfrak{A}$ and $S[\cdot, \ldots, \cdot]$ be a multi-context where all holes are in surface position. Then* $\texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s_1^{[a]}, \ldots, s_n^{[a]}] \preceq_A \texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s_1, \ldots, s_n]^{[a]}$.

If some hole $\cdot_i$ with $i \in \{1,\ldots,n\}$ is in strict position in $S[.,\ldots,.]$, then also the equation $\texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s_1^{[a]},\ldots,s_n^{[a]}] \approx_A \texttt{letrec}\ a{:=}n\ \texttt{in}\ S[s_1,\ldots,s_n]^{[a]}$ *holds.*

6 Conclusion

We introduced and analyzed the calculus LRPw – an extension of LRP by scoped work decorations. By proving several computation rules for the decoration, we lay the foundations for reasoning about program improvements using shared work decorations. Future work is to use the computation rules and to develop proof techniques to show that several program transformations are improvements.

References

[MS99] Moran, Andrew; Sands, David: Improvement in a Lazy Context: An operational theory for call-by-need. In: Proc. POPL 1999. ACM Press, pp. 43–56, 1999.

[Pi00] Pitts, Andrew M.: Parametric Polymorphism and Operational Equivalence. Math. Structures Comput. Sci., 10:321–359, 2000.

[Pi02] Pierce, Benjamin C.: Types and Programming Languages. The MIT Press, 2002.

[SS14] Schmidt-Schauß, Manfred; Sabel, David: Contextual Equivalences in Call-by-Need and Call-By-Name Polymorphically Typed Calculi (Preliminary Report). In: Proc. WPTE 2014. volume 40 of OASICS. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 63–74, 2014.

[SSS10] Schmidt-Schauß, Manfred; Sabel, David: On generic context lemmas for higher-order calculi with sharing. Theoret. Comput. Sci., 411(11-13):1521 – 1541, 2010.

[SSS15a] Schmidt-Schauß, Manfred; Sabel, David: Improvements in a Functional Core Language with Call-By-Need Operational Semantics. Frank report 55, Institut für Informatik, Goethe-Universität Frankfurt am Main, 2015. `http://www.ki.cs.uni-frankfurt.de/papers/frank/`.

[SSS15b] Schmidt-Schauß, Manfred; Sabel, David: Improvements in a Functional Core Language with Call-By-Need Operational Semantics. In (Albert, Elvira, ed.): Proc. PPDP '15. ACM, New York, NY, USA, pp. 220–231, 2015.

[SSS15c] Schmidt-Schauß, Manfred; Sabel, David: Sharing-Aware Improvements in a Call-by-Need Functional Core Language. presented at IFL 2015, 2015.

[SSS15d] Schmidt-Schauß, Manfred; Sabel, David: Sharing Decorations for Improvements in a Functional Core Language with Call-By-Need Operational Semantics. Frank report 56, Institut für Informatik. Goethe-Universität Frankfurt am Main, 2015. `http://www.ki.cs.uni-frankfurt.de/papers/frank/`.

[SSSS08] Schmidt-Schauß, Manfred; Schütz, Marko; Sabel, David: Safety of Nöcker's Strictness Analysis. J. Funct. Programming, 18(04):503–551, 2008.

[VPJ13] Vytiniotis, Dimitrios; Peyton Jones, Simon: Evidence Normalization in System FC (Invited Talk). In (van Raamsdonk, Femke, ed.): Proc. RTA 2013. volume 21 of LIPIcs, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, pp. 20–38, 2013.