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

David Sabel and Manfred Schmidt-Schauß

Goethe University Frankfurt am Main, Germany

ATPS 2016, Vienna, Austria
Motivation

Reasoning on program transformations, like

\[ \text{map } f \ (\text{map } g \ xs) \rightarrow \text{map } (\lambda x. f \ (g \ x)) \ xs \]

Are transformations optimizations / improvements?

- w.r.t. time consumption, i.e. the number of computation steps
- in a core language of Haskell:
  - extended polymorphically typed lambda calculus
  - with call-by-need evaluation
Some Previous and Related Work

[Moran & Sands, POPL’99]:
Improvement theory in an untyped call-by-need lambda calculus
  - counting based on an abstract machine semantics
  - **tick-algebra** for modular reasoning on improvements
  - no concrete technique for list induction proofs

[Hackett & Hutton, ICFP’14]:
Improvement for worker-wrapper-transformations
  - based on Moran & Sands’ tick algebra
  - argue for the requirement of a typed language

[Schmidt-Schauß & S., PPDP’15, IFL’15]:
Improvement in call-by-need lambda calculi: untyped LR, typed LRP
  - counting essential reduction steps of a small-step semantics
  - core language with seq-operator
  - **proving list-laws** being improvements, using **work-decorations**
Motivation: Equational Reasoning for List-Expressions

Example:

\[ s_1 := \text{letrec } x := 0 : x \text{ in } x \]
\[ s_2 := \text{letrec } y = ((\lambda x.x) 0), x := y : y : x \text{ in } x \]
\[ s_3 := \text{letrec } y = ((\lambda x.x) 0), x := y : ((\lambda x.x) y) : x \text{ in } x \]
\[ s_4 := \text{letrec } x := \lambda y. y : (x : y) \text{ in } (x : 0) \]

**Contextual equivalence:** \( s \sim_c t \) iff \( \forall \) contexts \( C : C[s] \downarrow \iff C[t] \downarrow \)

Prove \( s_i \sim_c s_j \) for all \( i, j \in \{1, 2, 3, 4\} \)
Motivation: Equational Reasoning for List-Expressions

Example:

\[
\begin{align*}
s_1 & := \text{letrec } xs=0 : xs \text{ in } xs \\
s_2 & := \text{letrec } y=((\lambda x.x) \ 0), xs=y : y : xs \text{ in } xs \\
s_3 & := \text{letrec } y=((\lambda x.x) \ 0), xs=y : ((\lambda x.x) \ y) : xs \text{ in } xs \\
s_4 & := \text{letrec } xs=\lambda y.y : (xs \ y) \text{ in } (xs \ 0)
\end{align*}
\]

Contextual equivalence: \( s \sim_c t \iff \forall \text{ contexts } C : C[s] \Downarrow \iff C[t] \Downarrow \)

Prove \( s_i \sim_c s_j \) for all \( i, j \in \{1, 2, 3, 4\} \)

- For list-expressions \( r_1, r_2 \) define:
  \[
  r_1 \ R \ r_2 \iff r_1 \sim_c (h_1 : t_1), \ r_2 \sim_c (h_2 : t_2)
  \]
  such that \( h_1 \sim_c h_2 \) and \( t_1 \ R \ t_2 \)

- Principle of co-induction: \( r_1 \ \text{gfp}(R) \ r_2 \implies r_1 \sim_c r_2 \)
Motivation: Equational Reasoning for List-Expressions

Example:

\[
\begin{align*}
  s_1 & := \ \text{letrec } x s = 0 : x s \ \text{in } x s \\
  s_2 & := \ \text{letrec } y = ((\lambda x. x) \ 0), x s = y : y : x s \ \text{in } x s \\
  s_3 & := \ \text{letrec } y = ((\lambda x. x) \ 0), x s = y : ((\lambda x. x) y) : x s \ \text{in } x s \\
  s_4 & := \ \text{letrec } x s = \lambda y. y : (x s \ y) \ \text{in } (x s \ 0)
\end{align*}
\]

Contextual equivalence: \( s \sim_c t \) iff \( \forall \) contexts \( C : C[s] \downarrow \iff C[t] \downarrow \)

Prove \( s_i \sim_c s_j \) for all \( i, j \in \{1, 2, 3, 4\} \)

- For list-expressions \( r_1, r_2 \) define:
  \( r_1 R r_2 \) iff \( r_1 \sim_c (h_1 : t_1), \ r_2 \sim_c (h_2 : t_2) \)
  such that \( h_1 \sim_c h_2 \) and \( t_1 R t_2 \)

- Principle of co-induction: \( r_1 \ \text{gfp}(R) \ r_2 \ \implies \ r_1 \sim_c r_2 \)
Motivation: Equational Reasoning for List-Expressions

Example:

\[ s_1 \sim_c 0 : 0 \text{ (letrec } xs=0 : xs \text{ in } xs \text{)} \]
\[ s_2 := \text{ letrec } y=((\lambda x.x) 0), xs=y : y : xs \text{ in } xs \]
\[ s_3 := \text{ letrec } y=((\lambda x.x) 0), xs=y : ((\lambda x.x) y) : xs \text{ in } xs \]
\[ s_4 := \text{ letrec } xs=\lambda y.y : (xs y) \text{ in } (xs 0) \]

Contextual equivalence: \( s \sim_c t \) iff \( \forall \) contexts \( C : C[s] \downarrow \iff C[t] \downarrow \)

Prove \( s_i \sim_c s_j \) for all \( i, j \in \{1, 2, 3, 4\} \)

- For list-expressions \( r_1, r_2 \) define:
  \[ r_1 \text{ } R \text{ } r_2 \text{ iff } r_1 \sim_c (h_1 : t_1), \text{ } r_2 \sim_c (h_2 : t_2) \]
  \[ \text{ such that } h_1 \sim_c h_2 \text{ and } t_1 \text{ } R \text{ } t_2 \]
- Principle of co-induction: \( r_1 \text{ } \text{gfp}(R) \text{ } r_2 \implies r_1 \sim_c r_2 \)
Example:

\[ s_1 \sim_c 0 : 0 : (\text{letrec } xs=0 : xs \text{ in } xs) \]
\[ s_2 := \text{letrec } y=((\lambda x.x) \ 0), xs=y : y : xs \text{ in } xs \]
\[ s_3 := \text{letrec } y=((\lambda x.x) \ 0), xs=y : ((\lambda x.x) \ y) : xs \text{ in } xs \]
\[ s_4 := \text{letrec } xs=\lambda y.y : (xs \ y) \text{ in } (xs \ 0) \]

**Contextual equivalence:** \( s \sim_c t \) iff \( \forall \) contexts \( C : C[s] \downarrow \iff C[t] \downarrow \)

Prove \( s_i \sim_c s_j \) for all \( i, j \in \{1, 2, 3, 4\} \)

- For list-expressions \( r_1, r_2 \) define:
  \[ r_1 \ R \ r_2 \iff r_1 \sim_c (h_1 : t_1), \ r_2 \sim_c (h_2 : t_2) \]
  such that \( h_1 \sim_c h_2 \) and \( t_1 \ R \ t_2 \)
- Principle of co-induction: \( r_1 \ \text{gfp}(R) \ r_2 \implies r_1 \sim_c r_2 \)
Motivation: Equational Reasoning for List-Expressions

Example:

\[ s_1 \sim_c 0 : 0 : (\text{letrec } x s = 0 : x s \text{ in } x s) \]
\[ s_2 \sim_c 0 : 0 : (\text{letrec } x s = 0 : 0 : x s \text{ in } x s) \]
\[ s_3 := \text{letrec } y = ((\lambda x.x) 0), x s = y : ((\lambda x.x) y) : x s \text{ in } x s \]
\[ s_4 := \text{letrec } x s = \lambda y.y : (x s y) \text{ in } (x s 0) \]

Contextual equivalence: \( s \sim_c t \iff \forall \text{ contexts } C : C[s] \downarrow \iff C[t] \downarrow \)

Prove \( s_i \sim_c s_j \) for all \( i, j \in \{1, 2, 3, 4\} \)

- For list-expressions \( r_1, r_2 \) define:
  \[ r_1 \ R \ r_2 \ \text{iff} \ r_1 \sim_c (h_1 : t_1), \ r_2 \sim_c (h_2 : t_2) \]
  such that \( h_1 \sim_c h_2 \) and \( t_1 \ R \ t_2 \)
- Principle of co-induction: \( r_1 \ \text{gfp}(R) \ r_2 \ \implies \ r_1 \sim_c r_2 \)
Motivation: Equational Reasoning for List-Expressions

Example:

\[ s_1 \sim_c 0 : 0 : (\text{letrec } xs = 0 : xs \text{ in } xs) \]
\[ s_2 \sim_c 0 : 0 : (\text{letrec } xs = 0 : 0 : xs \text{ in } xs) \]
\[ s_3 := \text{letrec } y = ((\lambda x.x) 0) \text{, } xs = y : ((\lambda x.x) y) : xs \text{ in } xs \]
\[ s_4 := \text{letrec } xs = \lambda y. y : (xs y) \text{ in } (xs 0) \]

Contextual equivalence: \( s \sim_c t \) iff \( \forall \) contexts \( C : C[s] \downarrow \iff C[t] \downarrow \)

Prove \( s_i \sim_c s_j \) for all \( i, j \in \{1, 2, 3, 4\} \)

- For list-expressions \( r_1, r_2 \) define:
  \( r_1 R r_2 \) iff \( r_1 \sim_c (h_1 : t_1) \), \( r_2 \sim_c (h_2 : t_2) \)
  such that \( h_1 \sim_c h_2 \) and \( t_1 R t_2 \)

- Principle of co-induction: \( r_1 \text{ gfp}(R) r_2 \implies r_1 \sim_c r_2 \)
Example:

\[ s_1 \sim_c 0 : 0 : (\text{letrec } \; x s = 0 : x s \; \text{in } x s) \]
\[ s_2 \sim_c 0 : 0 : (\text{letrec } \; x s = 0 : 0 : x s \; \text{in } x s) \]
\[ s_3 \sim_c 0 : 0 : (\text{letrec } \; x s = 0 : 0 : x s \; \text{in } x s) \]
\[ s_4 := \text{letrec } \; x s = \lambda y. y : (x s \; y) \; \text{in } (x s \; 0) \]

Contextual equivalence: \( s \sim_c t \) iff \( \forall \) contexts \( C : C[s] \downarrow \iff C[t] \downarrow \)

Prove \( s_i \sim_c s_j \) for all \( i, j \in \{1, 2, 3, 4\} \)

- For list-expressions \( r_1, r_2 \) define:
  \[ r_1 \ R \ r_2 \text{ iff } r_1 \sim_c (h_1 : t_1), \ r_2 \sim_c (h_2 : t_2) \]
  such that \( h_1 \sim_c h_2 \) and \( t_1 \ R \ t_2 \)
- Principle of co-induction: \( r_1 \ \text{gfp}(R) \ r_2 \implies r_1 \sim_c r_2 \)
Motivation: Equational Reasoning for List-Expressions

Example:

\[
\begin{align*}
s_1 & \sim_c 0 : 0 : (\text{letrec } x s = 0 : x s \text{ in } x s) \\
s_2 & \sim_c 0 : 0 : (\text{letrec } x s = 0 : 0 : x s \text{ in } x s) \\
s_3 & \sim_c 0 : 0 : (\text{letrec } x s = 0 : 0 : x s \text{ in } x s) \\
s_4 & := \text{letrec } x s = \lambda y. y : (x s y) \text{ in } (x s 0)
\end{align*}
\]

**Contextual equivalence:** \( s \sim_c t \) iff \( \forall \) contexts \( C : C[s] \downarrow \iff C[t] \downarrow \)

Prove \( s_i \sim_c s_j \) for all \( i, j \in \{1, 2, 3, 4\} \)

- For list-expressions \( r_1, r_2 \) define:
  
  \( r_1 \ R \ r_2 \) iff \( r_1 \sim_c (h_1 : t_1), \ r_2 \sim_c (h_2 : t_2) \)
  
  such that \( h_1 \sim_c h_2 \) and \( t_1 \ R \ t_2 \)

- Principle of co-induction: \( r_1 \ \text{gfp}(R) \ r_2 \implies r_1 \sim_c r_2 \)
Motivation: Equational Reasoning for List-Expressions

Example:

\[ s_1 \sim_c 0 : 0 : (\text{letrec } xs=0 : xs \text{ in } xs) \]
\[ s_2 \sim_c 0 : 0 : (\text{letrec } xs=0 : 0 : xs \text{ in } xs) \]
\[ s_3 \sim_c 0 : 0 : (\text{letrec } xs=0 : 0 : xs \text{ in } xs) \]
\[ s_4 \sim_c 0 : 0 : (\text{letrec } xs=\lambda y.y : (xs y) \text{ in } (xs 0)) \]

**Contextual equivalence:** \( s \sim_c t \text{ iff } \forall \text{ contexts } C : C[s] \downarrow \iff C[t] \downarrow \)

Prove \( s_i \sim_c s_j \) for all \( i, j \in \{1, 2, 3, 4\} \)

- For list-expressions \( r_1, r_2 \) define:
  \[ r_1 \ R \ r_2 \text{ iff } r_1 \sim_c (h_1 : t_1), \ r_2 \sim_c (h_2 : t_2) \]
  such that \( h_1 \sim_c h_2 \) and \( t_1 \ R \ t_2 \)

- Principle of co-induction: \( r_1 \ \text{gfp}(R) \ r_2 \implies r_1 \sim_c r_2 \)
Motivation: Equational Reasoning for List-Expressions

Example:

\[
\begin{align*}
  s_1 & \sim_c 0 : 0 : (\text{letrec } xs = 0 : xs \text{ in } xs) \\
  s_2 & \sim_c 0 : 0 : (\text{letrec } xs = 0 : 0 : xs \text{ in } xs) \\
  s_3 & \sim_c 0 : 0 : (\text{letrec } xs = 0 : 0 : xs \text{ in } xs) \\
  s_4 & \sim_c 0 : 0 : (\text{letrec } xs = \lambda y. y : (xs y) \text{ in } (xs 0))
\end{align*}
\]

Further processing with the tails indeed shows \(s_i \sim_c s_j\)

Contextual equivalence:

\(s \sim_c t\) iff \(\forall\) contexts \(C : C[s] \Downarrow \iff C[t] \Downarrow\)

Prove \(s_i \sim_c s_j\) for all \(i, j \in \{1, 2, 3, 4\}\)

- For list-expressions \(r_1, r_2\) define:
  \(r_1 \mathcal{R} r_2\) iff \(r_1 \sim_c (h_1 : t_1), r_2 \sim_c (h_2 : t_2)\)
  such that \(h_1 \sim_c h_2\) and \(t_1 \mathcal{R} t_2\)

- Principle of co-induction: \(r_1 \text{gfp}(\mathcal{R}) r_2 \implies r_1 \sim_c r_2\)
In [Schmidt-Schauß & S., IFL 2015]:

- analogous reasoning,
- but w.r.t. improvement and cost-equivalence

**Improvement and Cost-Equivalence**

**Improvement:**
\[ s \leq t \text{ iff } s \sim_c t \text{ and } \forall \text{ closing contexts } C : rln(C[s]) \leq rln(C[t]) \]

where \( rln(\cdot) \) is the reduction length, counting essential reduction steps

**Cost-Equivalence:**
\[ s \approx t \text{ iff } s \leq t \text{ and } t \leq s \]
Motivation: Reasoning including Resources

Equational reasoning w.r.t. cost equivalence:

\[ s_1 := \text{letrec } xs=0 : xs \text{ in } xs \]
\[ s_2 := \text{letrec } y=((\lambda x.x) 0), xs=y : y : xs \text{ in } xs \]
\[ s_3 := \text{letrec } y=((\lambda x.x) 0), xs=y : ((\lambda x.x) y) : xs \text{ in } xs \]
\[ s_4 := \text{letrec } xs=\lambda y.y : (xs y) \text{ in } (xs 0) \]
Motivation: Reasoning including Resources

Equational reasoning w.r.t. cost equivalence:

\[
\begin{align*}
  s_1 & := \text{letrec } xs = 0 : xs \text{ in } xs \\
  s_2 & := \text{letrec } y = ((\lambda x. x) \, 0), xs = y : y : xs \text{ in } xs \\
  s_3 & := \text{letrec } y = ((\lambda x. x) \, 0), xs = y : ((\lambda x. x) \, y) : xs \text{ in } xs \\
  s_4 & := \text{letrec } xs = \lambda y. y : (xs \, y) \text{ in } (xs \, 0)
\end{align*}
\]
Equational reasoning w.r.t. cost equivalence:

\[ s_1 \approx 0 : 0 : (\text{letrec } xs=0 : xs \text{ in } xs) \]
\[ s_2 := \text{letrec } y=((\lambda x.x) 0), xs=y : y : xs \text{ in } xs \]
\[ s_3 := \text{letrec } y=((\lambda x.x) 0), xs=y : ((\lambda x.x) y) : xs \text{ in } xs \]
\[ s_4 := \text{letrec } xs=\lambda y.y : (xs y) \text{ in } (xs 0) \]
Motivation: Reasoning including Resources

Equational reasoning w.r.t. cost equivalence:

\[ s_1 \approx 0:0:\text{letrec } xs=0:xs \text{ in } xs \]
\[ s_2 := \text{letrec } y=((\lambda x.x) 0), xs=y : y : xs \text{ in } xs \]
\[ s_3 := \text{letrec } y=((\lambda x.x) 0), xs=y : ((\lambda x.x ) y) : xs \text{ in } xs \]
\[ s_4 := \text{letrec } xs=\lambda y.y : (xs y) \text{ in } (xs 0) \]
Equational reasoning w.r.t. cost equivalence:

\[ s_1 \approx 0 : 0 : (\text{letrec}\ x = 0 : x \text{ in } x) \]
\[ s_2 \approx \text{letrec}\ y = 0[1], x = y : y : x \text{ in } x \]
\[ s_3 := \text{letrec}\ y = ((\lambda x. x) 0), x = y : ((\lambda x. x) y) : x \text{ in } x \]
\[ s_4 := \text{letrec}\ x = \lambda y. y : (x \ y) \text{ in } (x \ 0) \]

Work-decorations to keep track of \( r\ln\text{-work}: \)

\[ [n] := n \text{ essential reduction steps} \]
Motivation: Reasoning including Resources

Equational reasoning w.r.t. cost equivalence:

\[ s_1 \approx 0 : 0 : (\text{letrec } xs = 0 : xs \text { in } xs) \]
\[ s_2 \approx \text{letrec } xs = 0^{[a \mapsto 1]} : 0^{[a \mapsto 1]} : xs \text { in } xs \]
\[ s_3 := \text{letrec } y = ((\lambda x.x) 0), xs = y : ((\lambda x.x) y) : xs \text { in } xs \]
\[ s_4 := \text{letrec } xs = \lambda y.y : (xs y) \text { in } (xs 0) \]

Work-decorations to keep track of rln-work:

- \([n] := n\) essential reduction steps
- \([a \mapsto n] := n\) shared essential reduction steps
  (label \(a\) marks the sharing)
Motivation: Reasoning including Resources

Equational reasoning w.r.t. cost equivalence:

\[ s_1 \approx 0 : 0 : (\text{letrec } xs = 0 : xs \text{ in } xs) \]
\[ s_2 \approx 0^{[a \mapsto 1]} : 0^{[a \mapsto 1]} : (\text{letrec } xs = 0^{[a \mapsto 1]} : 0^{[a \mapsto 1]} : xs \text{ in } xs) \]
\[ s_3 := \text{letrec } y = ((\lambda x.x) 0), xs = y : ((\lambda x.x) y) : xs \text{ in } xs \]
\[ s_4 := \text{letrec } xs = \lambda y.y : (xs y) \text{ in } (xs 0) \]

Work-decorations to keep track of rln-work:

- \([n]\) := \(n\) essential reduction steps
- \([a \mapsto n]\) := \(n\) shared essential reduction steps
  (label \(a\) marks the sharing)
Equational reasoning w.r.t. cost equivalence:

\[
\begin{align*}
s_1 & \approx 0 : 0 : (\text{letrec } x s = 0 : x s \text{ in } x s) \\
s_2 & \approx 0[^{a \rightarrow 1}] : 0[^{a \rightarrow 1}] : (\text{letrec } x s = 0[^{a \rightarrow 1}] : 0[^{a \rightarrow 1}] : x s \text{ in } x s) \\
s_3 & := \text{letrec } y = ((\lambda x . x) 0), x s = y : ((\lambda x . x) y) : x s \text{ in } x s \\
s_4 & := \text{letrec } x s = \lambda y . y : (x s y) \text{ in } (x s 0)
\end{align*}
\]

Work-decorations to keep track of rln-work:

- \([n] := n\) essential reduction steps
- \([a \rightarrow n] := n\) shared essential reduction steps (label \(a\) marks the sharing)
Equational reasoning w.r.t. cost equivalence:

\[ s_1 \approx 0 : 0 : (\text{letrec } xs=0 : xs \text{ in } xs) \]
\[ s_2 \approx 0[a\mapsto 1] : 0[a\mapsto 1] : (\text{letrec } xs=0[a\mapsto 1] : 0[a\mapsto 1] : xs \text{ in } xs) \]
\[ s_3 \approx \text{letrec } y=0[1], xs=y : ((\lambda x.x) y) : xs \text{ in } xs \]
\[ s_4 := \text{letrec } xs=\lambda y.y : (xs y) \text{ in } (xs 0) \]

Work-decorations to keep track of rln-work:

- \([n] := n\) essential reduction steps
- \([a\mapsto n] := n\) shared essential reduction steps
  (label \(a\) marks the sharing)
Motivation: Reasoning including Resources

Equational reasoning w.r.t. cost equivalence:

\[ s_1 \approx 0 : 0 : (\text{letrec } x s = 0 : x s \text{ in } x s) \]
\[ s_2 \approx 0^{[a \mapsto 1]} : 0^{[a \mapsto 1]} : (\text{letrec } x s = 0^{[a \mapsto 1]} : 0^{[a \mapsto 1]} : x s \text{ in } x s) \]
\[ s_3 \approx \text{letrec } x s = 0^{[a \mapsto 1]} : ((\lambda x . x) 0^{[a \mapsto 1]} ) : x s \text{ in } x s \]
\[ s_4 := \text{letrec } x s = \lambda y . y : (x s y) \text{ in } (x s 0) \]

Work-decorations to keep track of \( r \ln \)-work:

- \([n] := n\) essential reduction steps
- \([a \mapsto n] := n\) shared essential reduction steps
  (label \( a \) marks the sharing)
Equational reasoning w.r.t. cost equivalence:

\[ s_1 \approx 0 : 0 : (\text{letrec } x s = 0 : x s \text{ in } x s) \]
\[ s_2 \approx 0[a \mapsto 1] : 0[a \mapsto 1] : (\text{letrec } x s = 0[a \mapsto 1] : 0[a \mapsto 1] : x s \text{ in } x s) \]
\[ s_3 \approx \text{letrec } x s = 0[a \mapsto 1] : (0[a \mapsto 1])[1] : x s \text{ in } x s \]
\[ s_4 := \text{letrec } x s = \lambda y. y : (x s y) \text{ in } (x s 0) \]

Work-decorations to keep track of rln-work:

- \([n] := n\) essential reduction steps
- \([a \mapsto n] := n\) shared essential reduction steps
  (label \(a\) marks the sharing)
Equational reasoning w.r.t. cost equivalence:

\[ s_1 \approx 0 : 0 : (\text{letrec } x s = 0 : x s \text{ in } x s) \]
\[ s_2 \approx 0^{[a \mapsto 1]} : 0^{[a \mapsto 1]} : (\text{letrec } x s = 0^{[a \mapsto 1]} : 0^{[a \mapsto 1]} : x s \text{ in } x s) \]
\[ s_3 \approx 0^{[a \mapsto 1]} : 0^{[a \mapsto 1, b \mapsto 1]} : (\text{letrec } x s = 0^{[a \mapsto 1]} : 0^{[a \mapsto 1, b \mapsto 1]} : x s \text{ in } x s) \]
\[ s_4 := \text{letrec } x s = \lambda y. y : (x s y) \text{ in } (x s 0) \]

Work-decorations to keep track of rln-work:

- \([n] := n\) essential reduction steps
- \([a \mapsto n] := n\) shared essential reduction steps
  (label \(a\) marks the sharing)
Motivation: Reasoning including Resources

Equational reasoning w.r.t. cost equivalence:

\[
\begin{align*}
  s_1 & \approx 0 : 0 : (\text{letrec } x s=0 : x s \text{ in } x s) \\
  s_2 & \approx 0^{[a \rightarrow 1]} : 0^{[a \rightarrow 1]} : (\text{letrec } x s=0^{[a \rightarrow 1]} : 0^{[a \rightarrow 1]} : x s \text{ in } x s) \\
  s_3 & \approx 0^{[a \rightarrow 1]} : (0^{[a \rightarrow 1], b \rightarrow 1}) : (\text{letrec } x s=0^{[a \rightarrow 1]} : 0^{[a \rightarrow 1], b \rightarrow 1} : x s \text{ in } x s) \\
  s_4 & : = \text{letrec } x s=\lambda y.y : (x s y) \text{ in } (x s 0)
\end{align*}
\]

Work-decorations to keep track of r1n-work:

- \([n] : = n \text{ essential reduction steps}
- \([a \rightarrow n] : = n \text{ shared essential reduction steps}
  \text{ (label } a \text{ marks the sharing)}
Motivation: Reasoning including Resources

Equational reasoning w.r.t. cost equivalence:

\[ s_1 \approx 0 : 0 : (\text{letrec } xs = 0 : xs \ \text{in} \ xs) \]
\[ s_2 \approx 0^{[a \mapsto 1]} : 0^{[a \mapsto 1]} : (\text{letrec } xs = 0^{[a \mapsto 1]} : 0^{[a \mapsto 1]} : xs \ \text{in} \ xs) \]
\[ s_3 \approx 0^{[a \mapsto 1]} : (0^{[a \mapsto 1, b \mapsto 1]} : (\text{letrec } xs = 0^{[a \mapsto 1]} : 0^{[a \mapsto 1, b \mapsto 1]} : xs \ \text{in} \ xs) \]
\[ s_4 \approx \text{letrec } xs = \lambda y. y : (xs \ y) \ \text{in} \ ((\lambda y. y : (xs \ y)) \ 0) \]

Work-decorations to keep track of rln-work:

- \([n] := n \) essential reduction steps
- \([a \mapsto n] := n \) shared essential reduction steps
  (label \( a \) marks the sharing)
Motivation: Reasoning including Resources

Equational reasoning w.r.t. cost equivalence:

\[ s_1 \approx 0 : (\text{letrec } x s = 0 : x s \text{ in } x s) \]
\[ s_2 \approx 0[a \mapsto 1] : 0[a \mapsto 1] : (\text{letrec } x s = 0[a \mapsto 1] : 0[a \mapsto 1] : x s \text{ in } x s) \]
\[ s_3 \approx 0[a \mapsto 1] : (0[a \mapsto 1, b \mapsto 1]) : (\text{letrec } x s = 0[a \mapsto 1] : 0[a \mapsto 1, b \mapsto 1] : x s \text{ in } x s) \]
\[ s_4 \approx \text{letrec } x s = \lambda y. y : (x s y) \text{ in } (0 : (x s 0))[1] \]

Work-decorations to keep track of rln-work:

- \([n] := n\) essential reduction steps
- \([a \mapsto n] := n\) shared essential reduction steps
  (label \(a\) marks the sharing)
Equational reasoning w.r.t. cost equivalence:

\[
\begin{align*}
    s_1 & \approx 0 : 0 : (\text{letrec } xs = 0 : xs \text{ in } xs) \\
    s_2 & \approx 0[a\mapsto 1] : 0[a\mapsto 1] : (\text{letrec } xs = 0[a\mapsto 1] : 0[a\mapsto 1] : xs \text{ in } xs) \\
    s_3 & \approx 0[a\mapsto 1] : (0[a\mapsto 1,b\mapsto 1]) : (\text{letrec } xs = 0[a\mapsto 1] : 0[a\mapsto 1,b\mapsto 1] : xs \text{ in } xs) \\
    s_4 & \approx \text{letrec } xs = \lambda y. y : (xs y) \text{ in } (0 : ((\lambda y. y : (xs y)) 0))^{[1]}
\end{align*}
\]

Work-decorations to keep track of rln-work:

- \([n] \;:=\; n\) essential reduction steps
- \([a\mapsto n] \;:=\; n\) shared essential reduction steps
  (label \(a\) marks the sharing)
Motivation: Reasoning including Resources

Equational reasoning w.r.t. cost equivalence:

\[ s_1 \approx 0 : 0 : (\text{letrec } xs = 0 : xs \text{ in } xs) \]
\[ s_2 \approx 0^{[a \mapsto 1]} : 0^{[a \mapsto 1]} : (\text{letrec } xs = 0^{[a \mapsto 1]} : 0^{[a \mapsto 1]} : xs \text{ in } xs) \]
\[ s_3 \approx 0^{[a \mapsto 1]} : (0^{[a \mapsto 1, b \mapsto 1]} : (\text{letrec } xs = 0^{[a \mapsto 1]} : 0^{[a \mapsto 1, b \mapsto 1]} : xs \text{ in } xs) \]
\[ s_4 \approx \text{letrec } xs = \lambda y.y : (xs \ y) \text{ in } (0 : (0 : (xs \ 0))^{[1]}))^{[1]} \]

Work-decorations to keep track of rln-work:

- \([n] : \approx n \text{ essential reduction steps}\)
- \([a \mapsto n] : \approx n \text{ shared essential reduction steps}\)
  (label \(a\) marks the sharing)
Motivation: Reasoning including Resources

Equational reasoning w.r.t. cost equivalence:

\[ s_1 \approx 0 : 0 : (\text{letrec } xs=0 : xs \text{ in } xs) \]
\[ s_2 \approx 0^{[a \mapsto 1]} : 0^{[a \mapsto 1]} : (\text{letrec } xs=0^{[a \mapsto 1]} : 0^{[a \mapsto 1]} : xs \text{ in } xs) \]
\[ s_3 \approx 0^{[a \mapsto 1]} : (0^{[a \mapsto 1, b \mapsto 1]} ) : (\text{letrec } xs=0^{[a \mapsto 1]} : 0^{[a \mapsto 1, b \mapsto 1]} : xs \text{ in } xs) \]
\[ s_4 \approx (0 : (0 : \text{letrec } xs=\lambda y. y : (xs \ y) \text{ in } (xs \ 0))^{[1]})^{[1]} \]

Work-decorations to keep track of rln-work:

- \([n] := n\) essential reduction steps
- \([a \mapsto n] := n\) shared essential reduction steps
  (label \(a\) marks the sharing)
Motivation: Reasoning including Resources

Equational reasoning w.r.t. cost equivalence:

\[
\begin{align*}
    s_1 & \approx 0 : 0 : (\text{letrec } xs = 0 : xs \text{ in } xs) \\
    s_2 & \approx 0^{[a \mapsto 1]} : 0^{[a \mapsto 1]} : (\text{letrec } xs = 0^{[a \mapsto 1]} : 0^{[a \mapsto 1]} : xs \text{ in } xs) \\
    s_3 & \approx 0^{[a \mapsto 1]} : (0^{[a \mapsto 1, b \mapsto 1]} : (\text{letrec } xs = 0^{[a \mapsto 1]} : 0^{[a \mapsto 1, b \mapsto 1]} : xs \text{ in } xs) \\
    s_4 & \approx (0 : (0 : \text{letrec } xs = \lambda y. y : (xs y) \text{ in } (xs 0))^{[1]}{[1]}
\end{align*}
\]

Further processing shows \( s_1 \preceq s_2 \preceq s_3 \preceq s_4 \)

Work-decorations to keep track of rln-work:

- \([n] := n\) essential reduction steps
- \([a \mapsto n] := n\) shared essential reduction steps
  (label \(a\) marks the sharing)
Motivation: Reasoning including Resources

Equational reasoning w.r.t. cost equivalence:

\[
\begin{align*}
s_1 & \approx 0 : 0 : (\text{letrec } xs = 0 : xs \text{ in } xs) \\
s_2 & \approx 0^{[a\mapsto1]} : 0^{[a\mapsto1]} : (\text{letrec } xs = 0^{[a\mapsto1]} : 0^{[a\mapsto1]} : xs \text{ in } xs) \\
s_3 & \approx 0^{[a\mapsto1]} : (0^{[a\mapsto1, b\mapsto1]} : (\text{letrec } xs = 0^{[a\mapsto1]} : 0^{[a\mapsto1, b\mapsto1]} : xs \text{ in } xs) \\
s_4 & \approx (0 : (0 : \text{letrec } xs = \lambda y. y : (xs y) \text{ in } (xs 0))^{[1]} \cdot^{[1]}
\end{align*}
\]

Further processing shows \( s_1 \preceq s_2 \preceq s_3 \preceq s_4 \)

Work-decorations to keep track of \texttt{rln}-work:

- \([n] := n\) essential reduction steps
- \([a\mapsto n] := n\) shared essential reduction steps
  (label \(a\) marks the sharing)

Goal of current paper:

Define and analyze the exact semantics of \([a\mapsto n]\)
Our Contribution

- Exact semantics of (shared) work-decorations $[a\mapsto n]$ (and $[n]$)
- Prove **computation rules**, like $S[s[a\mapsto n], t[a\mapsto n]] \preceq S[s, t][n]$
- The notation $[a\mapsto n]$ is ambiguous, e.g. in
  \begin{verbatim}
  letrec x=\lambda y.s[a\mapsto n] in C[x] when inlining the binding for x:
  \end{verbatim}
  Possibilities:
  - letrec $x=\lambda y.s[a\mapsto n]$ in $C[\lambda y.s[a\mapsto n]]$
  - letrec $x=\lambda y.s[a\mapsto n]$ in $C[\lambda y.s[b\mapsto n]]$ (where $b$ is fresh)
Our Contribution

- Exact semantics of (shared) work-decorations $[a \mapsto n]$ (and $[n]$)
- Prove **computation rules**, like $S[s[a \mapsto n], t[a \mapsto n]] \preceq S[s, t][n]$
- The notation $[a \mapsto n]$ is **ambiguous**, e.g. in
  letrec $x = \lambda y.s[a \mapsto n]$ in $C[x]$ when inlining the binding for $x$:

  Possibilities:
  - letrec $x = \lambda y.s[a \mapsto n]$ in $C[\lambda y.s[a \mapsto n]]$
  - letrec $x = \lambda y.s[a \mapsto n]$ in $C[\lambda y.s[b \mapsto n]]$ (where $b$ is fresh)

- We **change the notation** to add a **scoping** for work-decorations:

  Instead of $[a \mapsto n]$ we use a **binding** $a := n$ and a **label** $[a]$
Our Contribution

- Exact semantics of (shared) work-decorations $[a \mapsto n]$ (and $[n]$)
- Prove **computation rules**, like $S[s^{[a \mapsto n]}, t^{[a \mapsto n]}] \preceq S[s, t]^{[n]}$
- The notation $[a \mapsto n]$ is **ambiguous**, e.g. in
  letrec $x = \lambda y. s^{[a \mapsto n]}$ in $C[x]$ when inlining the binding for $x$:
  Possibilities:
  - letrec $x = \lambda y. s^{[a \mapsto n]}$ in $C[\lambda y. s^{[a \mapsto n]}]$
  - letrec $x = \lambda y. s^{[a \mapsto n]}$ in $C[\lambda y. s^{[b \mapsto n]}]$ (where $b$ is fresh)
- We **change the notation** to add a **scoping** for work-decorations:

  Instead of $[a \mapsto n]$ we use a **binding** $a := n$ and a **label** $[a]$

- Examples:
  - letrec $a := n, x = \lambda y. s^{[a]}$ in $C[x]$
  - letrec $x = \lambda y. (\text{letrec } a := n \text{ in } s^{[a]})$ in $C[x]
The Calculus LRPw

LRPw extends LRP by work decorations

Types:

\[ \tau \in Typ \quad ::= \quad A \mid (\tau_1 \to \tau_2) \mid K \tau_1 \ldots \tau_{\text{ar}(K)} \]

\[ \rho \in PTyp \quad ::= \quad \tau \mid \lambda A.\rho \]

Expressions:

\[ u \in PExpr_F \quad ::= \quad \Lambda A_1 \ldots \Lambda A_k.\lambda x.s \]

\[ s, t \in Expr_F \quad ::= \quad u \mid x :: \rho \mid (s \tau) \mid (s \cdot t) \mid (\text{seq} \ s \ t) \]

\[ \quad \mid \text{letrec} \ \text{bind}_1, \ldots, \text{bind}_m \ \text{in} \ t \]

\[ \quad \mid (c_{K,i} :: \tau \ s_1 \ldots \ s_{\text{ar}(c_{K,i})}) \]

\[ \quad \mid (\text{case}_K \ s \ \text{of} \ (\text{pat}_{K,1} \to t_1) \ldots (\text{pat}_{K,|D_K|} \to t_{|D_K|})) \]

\[ \quad \mid s[a], \text{where} \ a \ \text{is a label} \]

\[ \text{pat}_{K,i} \quad ::= \quad (c_{K,i} :: \tau \ x_1 :: \tau_1 \ldots x_{\text{ar}(c_{K,i})} :: \tau_{\text{ar}(c_{K,i})}) \]

\[ \text{bind}_i \quad ::= \quad x_i :: \rho_i = s_i \mid a := n, \text{where} \ n \in \mathbb{N} \ \text{and} \ a \ \text{is a label} \]
The Calculus LRPw: Operational Semantics

Normal Order Reduction $\xrightarrow{\text{LRPw}}$

- Small-step reduction relation
- Call-by-need strategy using reduction contexts $R$
- Several reduction rules, e.g.

(lbeta) $((\lambda x.s)\,t) \rightarrow \text{letrec } x = t \text{ in } s$

(cp-in) $\text{letrec } x_1 = (\lambda y.t), \{x_i = x_{i-1}\}_{i=2}^{m}, \text{Env in } C[x_m]$

$\rightarrow \text{letrec } x_1 = (\lambda y.t), \{x_i = x_{i-1}\}_{i=2}^{m}, \text{Env in } C[(\lambda y.t)]$

(seq-c) $\text{seq } v \, t \rightarrow t \text{ if } v \text{ is a value}$

(case-c) $\text{case}_{K} (c \, t_1 \ldots t_n) \ldots ((c \, y_1 \ldots y_n) \rightarrow s) \ldots$

$\rightarrow \text{letrec } \{y_i = t_i\}_{i=1}^{n} \text{ in } s$

...$

(\text{letwn}) \text{letrec } \ldots a := n \ldots C[(s^{[a]})] \rightarrow \text{letrec } \ldots a := n-1 \ldots C[s^{[a]}]$

(\text{letw0}) \text{letrec } \ldots a := 0 \ldots C[(s^{[a]})] \rightarrow \text{letrec } \ldots a := 0 \ldots C[s]
Contextual Equivalence in LRPw

**Convergence**

A weak head normal form (WHNF) is

- a value: \( \lambda x.s, \Lambda A.u, \) or \( c \rightarrow s \).
- \texttt{letrec} \( \text{Env in } v \), where \( v \) is a value
- \texttt{letrec} \( x_1 = c \rightarrow s, \{ x_i = x_{i-1} \}_{i=2}^m, \text{Env in } x_m \)

Convergence:

- \( s \downarrow t \) iff \( s \xrightarrow{\text{LRPw,*}} t \) \( \land t \) is a WHNF
- \( s \downarrow \) iff \( \exists t : s \downarrow t \).

**Contextual Equivalence**

For \( s, t :: \rho \), \( s \sim_c t \) iff for all contexts \( C[\cdot :: \rho] : C[s] \downarrow \iff C[t] \downarrow \)

Program transformation \( P \) is correct iff \( s \xrightarrow{P} t \implies s \sim_c t \)
Improvement in LRPw

Counting Essential Reductions

For \( \{\text{letw}, \text{letw}n\} = A_0 \subseteq A \subseteq \mathcal{A} = \{\text{letw}, \text{case}, \text{seq}, \text{letw}\} \):

\[
\text{rln}_A(t) := \begin{cases} 
\text{number of } A\text{-reductions in } t \xrightarrow{\text{LRPw,}^*} t', & \text{if } t \Downarrow t' \\
\infty, & \text{otherwise}
\end{cases}
\]

Improvement Relation

For \( s, t :: \rho \), \( s \) improves \( t \) (written \( s \preceq_A t \)) iff

- \( s \sim_C t \), and
- for all \( C[\cdot :: \rho] \) s.t. \( C[s], C[t] \) are closed: \( \text{rln}_A(C[s]) \leq \text{rln}_A(C[t]) \).

We write \( s \equiv_A t \iff s \preceq_A t \land t \preceq_A s \) (**cost-equivalence**)

Program transformation \( P \) is an improvement iff \( s \xrightarrow{P} t \implies t \preceq_A s \)
Do Work-Decorations Change the Semantics?

LRP

LRPw = LRP + a := n + [a]

Questions:
- Is there a change w.r.t. contextual equivalence?
- Is there a change w.r.t. improvement and cost-equivalence?
Q1: Is there a change w.r.t. contextual equivalence?

No, since:

**Theorem**

The embedding of LRP into LRP\(_w\) w.r.t. \(\sim_c\) is conservative and the calculi LRP and LRP\(_w\) are isomorphic: The isomorphism is 

\[[s]_{\sim_c,LRP_w} = [\text{rmw}(s)]_{\sim_c,LRP}\]

where \(\text{rmw}(\cdot)\) removes the decorations.
Q2: Is there a change w.r.t. cost-equivalence?

No, if (seq)-reductions do not count for rln:

**Theorem**

Let \( A_0 \subseteq A \subseteq \mathbb{A} \), such that \( \text{seq} \notin A \).

Then LRP and LRPw are isomorphic w.r.t. \( \approx_A \).

Encode letrec \( a := n, Env \) in \( s \) as

letrec \( x_a := \text{id}^{n+1}, Env[\text{seq} x_a \; t/t^{[a]}] \) in \( s[\text{seq} x_a \; t/t^{[a]}] \)
Q2: Is there a change w.r.t. cost-equivalence?

No, if (seq)-reductions do not count for $rln$:

**Theorem**

Let $A_0 \subseteq A \subset \mathbb{A}$, such that $\text{seq} \notin A$.
Then LRP and LRP$_w$ are **isomorphic** w.r.t. $\approx_A$.

Encode letrec $a := n, \text{Env}$ in $s$ as
letrec $x_a := \text{id}^{n+1}, \text{Env}[\text{seq } x_a \ t/t[a]]$ in $s[\text{seq } x_a \ t/t[a]]$

Yes, for the isomorphism property if $A = \mathbb{A}$

**Proposition**

Let $A = \mathbb{A}$ and let $c_1$ and $c_2$ be different constants. Then
letrec $a := 1$ in $(\text{Pair } c_1^{[a]} c_2^{[a]})$

is **not equivalent** w.r.t. $\approx_A$ to any LRP-expression.
Q2: Is there a change w.r.t. cost-equivalence?

No, if (seq)-reductions do not count for rln:

**Theorem**
Let $A_0 \subseteq A \subseteq \mathcal{A}$, such that $seq \notin A$.
Then LRP and LRP\textsubscript{w} are isomorphic w.r.t. $\approx_A$.

Encode letrec $a ::= n, Env$ in $s$ as
letrec $x_a ::= \text{id}^{n+1}, Env[seq\ x_a\ t/t[a]]$ in $s[seq\ x_a\ t/t[a]]$

Yes, for the isomorphism property if $A = \mathcal{A}$

**Proposition**
Let $A = \mathcal{A}$ and let $c_1$ and $c_2$ be different constants. Then
letrec $a ::= 1$ in $(\text{Pair}\ c_1^{[a]}\ c_2^{[a]})$

is not equivalent w.r.t. $\approx_A$ to any LRP-expression.

Open for conservativity: $s \approx_{\mathcal{A},\text{LRP}} t \implies s \approx_{\mathcal{A},\text{LRPw}} t$?
Theorem

Let $A_0 \subseteq A \subseteq \mathbb{A}$.

- If $s \xrightarrow{\text{LRPw},a} t$ where $a \in A$ then $s \approx_A t^{[1]}$
- If $s \xrightarrow{C,a} t$ where $a \in A$ then $t \preceq_A s$
- If $s \xrightarrow{C,a} t$, $a$ is (III), (cp), (letw0), (cpx), (cpcx), (abs), (abse), (lwas), (ucp), (gc), (gcW), then $s \approx_A t$

(III) $\text{letrec } Env_1 \text{ in letrec } Env_2 \text{ in } s \rightarrow \text{letrec } Env_1, Env_2 \text{ in } s$

(III) $\text{letrec } Env_1, x=(\text{letrec } Env_2 \text{ in } s) \text{ in } t \rightarrow \text{letrec } Env_1, Env_2, x=s \text{ in } t$

(III) $(\text{letrec } Env \text{ in } s) \ t \rightarrow \text{letrec } Env \text{ in } (s \ t)$

(gc) $\text{letrec } \{x_i=s_i\}_{i=1}^n, Env \text{ in } t \rightarrow \text{letrec } Env \text{ in } t$, if $\forall i : x_i \not\in FV(t, Env)$

(gc) $\text{letrec } \{x_i=s_i\}_{i=1}^n \text{ in } t \rightarrow t$, if for all $i : x_i \not\in FV(t)$

(gcW) $\text{letrec } \{a_i := n_i\}_{i=1}^m, Env \text{ in } s \rightarrow \text{letrec } Env \text{ in } s$, if all $a_i$ do not occur in $Env, s$

(gcW) $\text{letrec } \{a_i := n_i\}_{i=1}^m \text{ in } s \rightarrow s$, if $a_1, \ldots, a_m$ do not occur in $s$

(cpx) $\text{letrec } x=y, \ldots C[x] \ldots \rightarrow \text{letrec } x=y, \ldots C[y] \ldots$

(cpcx) $\text{letrec } x=(ct_1 \ldots t_n) \ldots C[x] \ldots \rightarrow \text{letrec } x=(cy_1 \ldots y_n), \{y_i=t_i\}_{i=1}^n \ldots C[c\ y_1 \ldots y_n] \ldots$
Theorem

Let $A_0 \subseteq A \subseteq \mathcal{A}$ and $S,T$ be surface contexts

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

2. letrec $a := n$ in $(s[a])[a] \approx_A$ letrec $a := n$ in $s[a]$

3. $S[\text{letrec } a := n \text{ in } T[s[a]]] \preceq_A$ letrec $a := n$ in $S[T[s]]^{[a]}$

4. $S[\text{letrec } a := n \text{ in } T[s[a]]] \approx_A$ letrec $a := n$ in $S[T[s]]^{[a]}$, if $S[T]$ is strict.

5. letrec $a := n, b := m$ in $(s[a])[b] \approx_A$ letrec $a := n, b := m$ in $(s[b])[a]$

6. letrec $a := n$ in $S[s_1^{[a]}, \ldots, s_n^{[a]}] \preceq_A$ letrec $a := n$ in $S[s_1, \ldots, s_n]^{[a]}$.

7. letrec $a := n$ in $S[s_1^{[a]}, \ldots, s_n^{[a]}] \approx_A$ letrec $a := n$ in $S[s_1, \ldots, s_n]^{[a]}$, if some hole in $S$ is in strict position.
Conclusion

- \( \text{LRP}_w = \text{Call-by-need calculus with scoped work-decorations} \)
- \( \text{LRP}_w \) **not obviously encodable** in \( \text{LRP} \)
- Several **improvements and cost-equivalences** hold in \( \text{LRP}_w \)
- Expected **computation rules** hold in \( \text{LRP}_w \)
Further Work

- Apply the results to prove **further improvements and cost-equivalences**
- **Automation** of **program optimization**
- **Automation** of **proving** improvement
- Space-improvements