Positive Sharing and Abstract Machines

Jui-Hsuan Wu

CNRS, LIP, ENS de Lyon

Joint work with Beniamino Accattoli and Claudio Sacerdoti Coen

GdT Plume, LIP, ENS de Lyon

13 October 2025

Design of λ-calculi with sharing

To introduce sharing in the λ-calculus is, one can simply add let expressions (or explicit substitutions).

$$t, u ::= x \mid tu \mid \lambda x.t \mid t[x \leftarrow u]$$

There are some flexibilities in the design of such a λ-calculus with sharing:

1. restrictions on the syntax
2. small-step substitutions vs. micro-step substitutions

$$(xx)[x \leftarrow t] \rightarrow tt \qquad (xx)[x \leftarrow t] \rightarrow (tx)[x \leftarrow t] \rightarrow (tt)[x \leftarrow t]$$

The positive λ-calculus (λ_{pos}) is a "micro-step" call-by-value λ-calculus with a restricted use of ESs.

Design of λ-calculi with sharing

To introduce sharing in the λ-calculus is, one can simply add let expressions (or explicit substitutions).

$$t, u ::= x \mid tu \mid \lambda x.t \mid t[x \leftarrow u]$$

There are some flexibilities in the design of such a λ-calculus with sharing:

1. restrictions on the syntax
2. small-step substitutions vs. micro-step substitutions

$$(xx)[x \leftarrow t] \rightarrow tt \qquad (xx)[x \leftarrow t] \rightarrow (tx)[x \leftarrow t] \rightarrow (tt)[x \leftarrow t]$$

The positive λ-calculus (λ_{pos}) is a "micro-step" call-by-value λ-calculus with a restricted use of ESs.

Design of λ-calculi with sharing

To introduce sharing in the λ-calculus is, one can simply add let expressions (or explicit substitutions).

$$t, u ::= x \mid tu \mid \lambda x.t \mid t[x \leftarrow u]$$

There are some flexibilities in the design of such a λ-calculus with sharing:

1. restrictions on the syntax
2. small-step substitutions vs. micro-step substitutions

$$(xx)[x \leftarrow t] \rightarrow tt \qquad (xx)[x \leftarrow t] \rightarrow (tx)[x \leftarrow t] \rightarrow (tt)[x \leftarrow t]$$

The positive λ-calculus (λ_{pos}) is a "micro-step" call-by-value λ-calculus with a restricted use of ESs.

Design of λ-calculi with sharing

To introduce sharing in the λ-calculus is, one can simply add let expressions (or explicit substitutions).

$$t, u ::= x \mid tu \mid \lambda x.t \mid t[x \leftarrow u]$$

There are some flexibilities in the design of such a λ-calculus with sharing:

1. restrictions on the syntax
2. small-step substitutions vs. micro-step substitutions

$$(xx)[x \leftarrow t] \rightarrow tt \qquad (xx)[x \leftarrow t] \rightarrow (tx)[x \leftarrow t] \rightarrow (tt)[x \leftarrow t]$$

The positive λ-calculus (λ_{pos}) is a "micro-step" call-by-value λ-calculus with a restricted use of ESs.

Design of λ-calculi with sharing

To introduce sharing in the λ-calculus is, one can simply add let expressions (or explicit substitutions).

$$t, u ::= x \mid tu \mid \lambda x.t \mid t[x \leftarrow u]$$

There are some flexibilities in the design of such a λ-calculus with sharing:

1. restrictions on the syntax
2. small-step substitutions vs. micro-step substitutions

$$(xx)[x \leftarrow t] \rightarrow tt \qquad (xx)[x \leftarrow t] \rightarrow (tx)[x \leftarrow t] \rightarrow (tt)[x \leftarrow t]$$

The positive λ-calculus (λ_{pos}) is a "micro-step" call-by-value λ-calculus with a restricted use of ESs.

Design of λ-calculi with sharing

To introduce sharing in the λ-calculus is, one can simply add let expressions (or explicit substitutions).

$$t, u ::= x \mid tu \mid \lambda x.t \mid t[x \leftarrow u]$$

There are some flexibilities in the design of such a λ-calculus with sharing:

1. restrictions on the syntax
2. small-step substitutions vs. micro-step substitutions

$$(xx)[x \leftarrow t] \rightarrow tt \qquad (xx)[x \leftarrow t] \rightarrow (tx)[x \leftarrow t] \rightarrow (tt)[x \leftarrow t]$$

The positive λ-calculus (λ_{pos}) is a "micro-step" call-by-value λ-calculus with a restricted use of ESs.

Positive λ-calculus λ_{pos}

The syntax of λ_{pos} comes from a proofs-as-terms interpretation of focused proofs, leading to its minimalistic form (See [Miller and Wu, 2023] and [Wu, 2023] for more).

λ_{pos} captures the notion of *useful sharing*, a key concept in the study of reasonable cost models of λ-calculus (See [Accattoli and Wu, 2024])

Bites:

$$b, b' ::= yz \mid \lambda y.u \mid (\lambda y.u)z$$

Terms:

$$t, u ::= x \mid t[x \leftarrow b]$$

Evaluation contexts:

$$O ::= \langle\cdot\rangle \mid O[x \leftarrow b]$$

Reduction rules:

$$\begin{array}{rcl} t[x \leftarrow (\lambda y.\overbrace{O\langle z\rangle}^{u})w] & \mapsto_{\mathsf{m}} & O\langle t\{x \leftarrow z\}\rangle\{y \leftarrow w\} \\ O\langle t[x \leftarrow yz]\rangle[y \leftarrow \lambda w.u] & \mapsto_{\mathsf{e}} & O\langle t[x \leftarrow (\lambda w.u)z]\rangle[y \leftarrow \lambda w.u] \end{array}$$

Positive λ-calculus $\lambda_{\mathtt{pos}}$

The syntax of $\lambda_{\mathtt{pos}}$ comes from a proofs-as-terms interpretation of focused proofs, leading to its minimalistic form (See [Miller and Wu, 2023] and [Wu, 2023] for more).

$\lambda_{\mathtt{pos}}$ captures the notion of *useful sharing*, a key concept in the study of reasonable cost models of λ-calculus (See [Accattoli and Wu, 2024])

Bites:

$$b, b' ::= yz \mid \lambda y.u \mid (\lambda y.u)z$$

Terms:

$$t, u ::= x \mid t[x \leftarrow b]$$

Evaluation contexts:

$$O ::= \langle \cdot \rangle \mid O[x \leftarrow b]$$

Reduction rules:

$$\begin{array}{rcl} t[x \leftarrow (\lambda y. \overbrace{O\langle z \rangle}^{u})w] & \mapsto_{\mathsf{m}} & O\langle t\{x \leftarrow z\}\rangle\{y \leftarrow w\} \\ O\langle t[x \leftarrow yz]\rangle[y \leftarrow \lambda w.u] & \mapsto_{\mathsf{e}} & O\langle t[x \leftarrow (\lambda w.u)z]\rangle[y \leftarrow \lambda w.u] \end{array}$$

Positive λ-calculus λ_{pos}

The syntax of λ_{pos} comes from a proofs-as-terms interpretation of focused proofs, leading to its minimalistic form (See [Miller and Wu, 2023] and [Wu, 2023] for more).

λ_{pos} captures the notion of *useful sharing*, a key concept in the study of reasonable cost models of λ-calculus (See [Accattoli and Wu, 2024])

Bites:

$$b, b' ::= yz \mid \lambda y.u \mid (\lambda y.u)z$$

Terms:

$$t, u ::= x \mid t[x \leftarrow b]$$

Evaluation contexts:

$$O ::= \langle \cdot \rangle \mid O[x \leftarrow b]$$

Reduction rules:

$$t[x \leftarrow (\lambda y.\overbrace{O\langle z\rangle}^{u})w] \quad \mapsto_{\mathsf{m}} \quad O\langle t\{x \leftarrow z\}\rangle\{y \leftarrow w\}$$

$$O\langle t[x \leftarrow yz]\rangle[y \leftarrow \lambda w.u] \quad \mapsto_{\mathsf{e}} \quad O\langle t[x \leftarrow (\lambda w.u)z]\rangle[y \leftarrow \lambda w.u]$$

Positive λ-calculus λ_{pos}

The syntax of λ_{pos} comes from a proofs-as-terms interpretation of focused proofs, leading to its minimalistic form (See [Miller and Wu, 2023] and [Wu, 2023] for more).

λ_{pos} captures the notion of *useful sharing*, a key concept in the study of reasonable cost models of λ-calculus (See [Accattoli and Wu, 2024])

Bites:

$$b, b' ::= yz \mid \lambda y.u \mid (\lambda y.u)z$$

Terms:

$$t, u ::= x \mid t[x \leftarrow b]$$

Evaluation contexts:

$$O ::= \langle \cdot \rangle \mid O[x \leftarrow b]$$

Reduction rules:

$$\begin{array}{rcl} t[x \leftarrow (\lambda y. \overbrace{O\langle z \rangle}^{u})w] & \mapsto_{\mathsf{m}} & O\langle t\{x \leftarrow z\}\rangle\{y \leftarrow w\} \\ O\langle t[x \leftarrow yz]\rangle[y \leftarrow \lambda w.u] & \mapsto_{\mathsf{e}} & O\langle t[x \leftarrow (\lambda w.u)z]\rangle[y \leftarrow \lambda w.u] \end{array}$$

Example

$$t[x{\leftarrow}(\lambda y.\ \overbrace{O\langle z\rangle}^{u})w] \quad \mapsto_{\mathsf{m}} \quad O\langle t\{x{\leftarrow}z\}\rangle\{y{\leftarrow}w\}$$
$$O\langle t[x{\leftarrow}yz]\rangle[y{\leftarrow}\lambda w.u] \quad \mapsto_{\mathsf{e}} \quad O\langle t[x{\leftarrow}(\lambda w.u)z]\rangle[y{\leftarrow}\lambda w.u]$$

Example of reduction:

$$\begin{aligned} & x[x{\leftarrow}yy][y{\leftarrow}zz'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ \to_{\mathsf{e}} \quad & x[x{\leftarrow}yy][y{\leftarrow}(\lambda w.w'[w'{\leftarrow}ww])z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ & x[x{\leftarrow}w_1'w_1'][w_1'{\leftarrow}z'z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \end{aligned}$$

Example

$$t[x{\leftarrow}(\lambda y.\ \overbrace{O\langle z\rangle}^{u})w] \quad \mapsto_{\mathsf{m}} \quad O\langle t\{x{\leftarrow}z\}\rangle\{y{\leftarrow}w\}$$
$$O\langle t[x{\leftarrow}yz]\rangle[y{\leftarrow}\lambda w.u] \quad \mapsto_{\mathsf{e}} \quad O\langle t[x{\leftarrow}(\lambda w.u)z]\rangle[y{\leftarrow}\lambda w.u]$$

Example of reduction:

$$\begin{aligned} & x[x{\leftarrow}yy][y{\leftarrow}zz'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ \to_{\mathsf{e}} \quad & x[x{\leftarrow}yy][y{\leftarrow}(\lambda w.w'[w'{\leftarrow}ww])z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ \to_{\mathsf{m}} \quad & x[x{\leftarrow}w_1'w_1'][w_1'{\leftarrow}z'z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \end{aligned}$$

Example

$$t[x{\leftarrow}(\lambda y.\ \overbrace{O\langle z\rangle}^{u})w] \quad \mapsto_{\mathsf{m}} \quad O\langle t\{x{\leftarrow}z\}\rangle\{y{\leftarrow}w\}$$
$$O\langle t[x{\leftarrow}yz]\rangle[y{\leftarrow}\lambda w.u] \quad \mapsto_{\mathsf{e}} \quad O\langle t[x{\leftarrow}(\lambda w.u)z]\rangle[y{\leftarrow}\lambda w.u]$$

Example of reduction:

$$\begin{aligned} & x[x{\leftarrow}yy][y{\leftarrow}zz'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ \rightarrow_{\mathsf{e}} \quad & x[x{\leftarrow}yy][y{\leftarrow}(\lambda w.w'[w'{\leftarrow}ww])z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ \rightarrow_{\mathsf{m}} \quad & x[x{\leftarrow}w_1'w_1'][w_1'{\leftarrow}z'z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \end{aligned}$$

Example

$$t[x{\leftarrow}(\lambda y.\ \overbrace{O\langle z\rangle}^{u})w] \quad \mapsto_{\mathsf{m}} \quad O\langle t\{x{\leftarrow}z\}\rangle\{y{\leftarrow}w\}$$
$$O\langle t[x{\leftarrow}yz]\rangle[y{\leftarrow}\lambda w.u] \quad \mapsto_{\mathsf{e}} \quad O\langle t[x{\leftarrow}(\lambda w.u)z]\rangle[y{\leftarrow}\lambda w.u]$$

Example of reduction:

$$\begin{aligned}
& x[x{\leftarrow}yy][y{\leftarrow}zz'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\
\rightarrow_{\mathsf{e}} \quad & x[x{\leftarrow}yy][y{\leftarrow}(\lambda w.w'[w'{\leftarrow}ww])z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\
\rightarrow_{\mathsf{m}} \quad & x[x{\leftarrow}w_1'w_1'][w_1'{\leftarrow}z'z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]]
\end{aligned}$$

Example

$$t[x{\leftarrow}(\lambda y.\ \overbrace{O\langle z\rangle}^{u})w] \quad \mapsto_{\mathsf{m}} \quad O\langle t\{x{\leftarrow}z\}\rangle\{y{\leftarrow}w\}$$
$$O\langle t[x{\leftarrow}yz]\rangle[y{\leftarrow}\lambda w.u] \quad \mapsto_{\mathsf{e}} \quad O\langle t[x{\leftarrow}(\lambda w.u)z]\rangle[y{\leftarrow}\lambda w.u]$$

Example of reduction:

$$\begin{aligned}
& x[x{\leftarrow}yy][y{\leftarrow}zz'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\
\to_{\mathsf{e}} \quad & x[x{\leftarrow}yy]\underline{[y{\leftarrow}(\lambda w.w'[w'{\leftarrow}ww])z']}[z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\
\to_{\mathsf{m}} \quad & x[x{\leftarrow}w_1'w_1'][w_1'{\leftarrow}z'z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]]
\end{aligned}$$

Example

$$
\begin{array}{rcl}
t[x{\leftarrow}(\lambda y.\ \overbrace{O\langle z\rangle}^{u})w] & \mapsto_{\mathsf{m}} & O\langle t\{x{\leftarrow}z\}\rangle\{y{\leftarrow}w\} \\
O\langle t[x{\leftarrow}yz]\rangle[y{\leftarrow}\lambda w.u] & \mapsto_{\mathsf{e}} & O\langle t[x{\leftarrow}(\lambda w.u)z]\rangle[y{\leftarrow}\lambda w.u]
\end{array}
$$

Example of reduction:

$$
\begin{array}{rl}
 & x[x{\leftarrow}yy][y{\leftarrow}zz'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\
\to_{\mathsf{e}} & x[x{\leftarrow}yy][y{\leftarrow}(\lambda w.w'[w'{\leftarrow}ww])z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\
\to_{\mathsf{m}} & x[x{\leftarrow}w_1'w_1'][w_1'{\leftarrow}z'z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]]
\end{array}
$$

Right strategy

Just reduce the redexes from right to left!

This can be formally defined using *right contexts* R.

Example:

- $\langle\cdot\rangle[x{\leftarrow}yy][z{\leftarrow}\lambda w.t]$ is a right context
- $\langle\cdot\rangle[x{\leftarrow}yy][y{\leftarrow}\lambda w.t]$ is not a right context

Lemma (Basic properties of the right strategy)

1. *Determinism: if* $t \to_{\mathtt{r}} t_1$ *and* $t \to_{\mathtt{r}} t_2$ *then* $t_1 = t_2$;
2. *No premature stops: if* $t \to_{\mathsf{pos}} u$ *then* $t \to_{\mathtt{r}} r$ *for some* r.

Right strategy

Just reduce the redexes from right to left!

This can be formally defined using *right contexts* R.

Example:

- $\langle\cdot\rangle[x{\leftarrow}yy][z{\leftarrow}\lambda w.t]$ is a right context
- $\langle\cdot\rangle[x{\leftarrow}yy][y{\leftarrow}\lambda w.t]$ is not a right context

Lemma (Basic properties of the right strategy)

1. *Determinism: if* $t \to_{\mathtt{r}} t_1$ *and* $t \to_{\mathtt{r}} t_2$ *then* $t_1 = t_2$;
2. *No premature stops: if* $t \to_{\mathsf{pos}} u$ *then* $t \to_{\mathtt{r}} r$ *for some* r.

Right strategy

Just reduce the redexes from right to left!

This can be formally defined using *right contexts* R.

Example:

- $\langle\cdot\rangle[x{\leftarrow}yy][z{\leftarrow}\lambda w.t]$ is a right context
- $\langle\cdot\rangle[x{\leftarrow}yy][y{\leftarrow}\lambda w.t]$ is not a right context

Lemma (Basic properties of the right strategy)

1. *Determinism: if* $t \to_{\mathtt{r}} t_1$ *and* $t \to_{\mathtt{r}} t_2$ *then* $t_1 = t_2$;
2. *No premature stops: if* $t \to_{\mathsf{pos}} u$ *then* $t \to_{\mathtt{r}} r$ *for some* r.

Natural POsitive Machine (Natural POM)

States are pairs $t \triangleleft R$.

Transitions:

$$
\begin{array}{llll|c|llll}
t[x{\leftarrow}\lambda y.u] & \triangleleft & R & & \rightsquigarrow_{\mathsf{sea}_1} & t & \triangleleft & R\langle\langle\cdot\rangle[x{\leftarrow}\lambda y.u]\rangle \\
t[x{\leftarrow}yz] & \triangleleft & R & & \rightsquigarrow_{\mathsf{sea}_2} & t & \triangleleft & R\langle\langle\cdot\rangle[x{\leftarrow}yz]\rangle \\
t[x{\leftarrow}yz] & \triangleleft & R & & \rightsquigarrow_{\mathsf{e}} & t[x{\leftarrow}(\lambda w.u)^{\alpha}z] & \triangleleft & R \\
t[x{\leftarrow}(\lambda y.O\langle z\rangle)w] & \triangleleft & R & & \rightsquigarrow_{\mathsf{m}} & O\langle t\{x{\leftarrow}z\}\rangle\{y{\leftarrow}w\} & \triangleleft & R
\end{array}
$$

Natural POsitive Machine (Natural POM)

States are pairs $t \triangleleft R$.

Transitions:

$$
\begin{array}{llll|c|llll}
t[x{\leftarrow}\lambda y.u] & \triangleleft & R & & \rightsquigarrow_{\mathtt{sea}_1} & t & \triangleleft & R\langle\langle\cdot\rangle[x{\leftarrow}\lambda y.u]\rangle \\
t[x{\leftarrow}yz] & \triangleleft & R & & \rightsquigarrow_{\mathtt{sea}_2} & t & \triangleleft & R\langle\langle\cdot\rangle[x{\leftarrow}yz]\rangle \\
t[x{\leftarrow}yz] & \triangleleft & R & & \rightsquigarrow_{\mathtt{e}} & t[x{\leftarrow}(\lambda w.u)^{\alpha}z] & \triangleleft & R \\
t[x{\leftarrow}(\lambda y.O\langle z\rangle)w] & \triangleleft & R & & \rightsquigarrow_{\mathtt{m}} & O\langle t\{x{\leftarrow}z\}\rangle\{y{\leftarrow}w\} & \triangleleft & R
\end{array}
$$

Complexity of abstract machines

We are interested in the overhead of abstract machines with respect to its underlying calculus/strategy, that is, to the following two parameters:

- the size $|t|$ of the initial term t
- the number of steps of the calculus

Many abstract machines in the literature have been shown to be bi-linear, that is, linear in both parameters.

Unfortunately, Natural POM is not one of them.

Complexity of abstract machines

We are interested in the overhead of abstract machines with respect to its underlying calculus/strategy, that is, to the following two parameters:

- the size $|t|$ of the initial term t
- the number of steps of the calculus

Many abstract machines in the literature have been shown to be bi-linear, that is, linear in both parameters.

Unfortunately, Natural POM is not one of them.

Complexity of abstract machines

We are interested in the overhead of abstract machines with respect to its underlying calculus/strategy, that is, to the following two parameters:

- the size $|t|$ of the initial term t
- the number of steps of the calculus

Many abstract machines in the literature have been shown to be bi-linear, that is, linear in both parameters.

Unfortunately, Natural POM is not one of them.

Complexity of abstract machines

We are interested in the overhead of abstract machines with respect to its underlying calculus/strategy, that is, to the following two parameters:

- the size $|t|$ of the initial term t
- the number of steps of the calculus

Many abstract machines in the literature have been shown to be bi-linear, that is, linear in both parameters.

Unfortunately, Natural POM is not one of them.

Complexity of abstract machines

We are interested in the overhead of abstract machines with respect to its underlying calculus/strategy, that is, to the following two parameters:

- the size $|t|$ of the initial term t
- the number of steps of the calculus

Many abstract machines in the literature have been shown to be bi-linear, that is, linear in both parameters.

Unfortunately, Natural POM is not one of them.

Inefficiency of Natural POM

The main problem with inefficiency of Natural POM comes from the following transition:

$$t[x{\leftarrow}(\lambda y.O\langle z\rangle)w] \triangleleft R \rightsquigarrow_{\mathsf{m}} O\langle t\{x{\leftarrow}z\}\rangle\{y{\leftarrow}w\} \triangleleft R$$

The meta-level renaming $O\langle t\{x{\leftarrow}z\}\rangle\{y{\leftarrow}w\}$ is not an issue since y can only appear free in $O\langle z\rangle$, which is always a subterm (up to α-renaming) of the initial term.

However, the renaming $t\{x{\leftarrow}z\}$ is problematic: t is not always a subterm of the initial term!

Inefficiency of Natural POM

The main problem with inefficiency of Natural POM comes from the following transition:

$$t[x{\leftarrow}(\lambda y.O\langle z\rangle)w] \triangleleft R \rightsquigarrow_{\mathsf{m}} O\langle t\{x{\leftarrow}z\}\rangle\{y{\leftarrow}w\} \triangleleft R$$

The meta-level renaming $O\langle t\{x{\leftarrow}z\}\rangle\{y{\leftarrow}w\}$ is not an issue since y can only appear free in $O\langle z\rangle$, which is always a subterm (up to α-renaming) of the initial term.

However, the renaming $t\{x{\leftarrow}z\}$ is problematic: t is not always a subterm of the initial term!

Inefficiency of Natural POM

The main problem with inefficiency of Natural POM comes from the following transition:

$$t[x{\leftarrow}(\lambda y.O\langle z\rangle)w] \triangleleft R \rightsquigarrow_{\mathsf{m}} O\langle t\{x{\leftarrow}z\}\rangle\{y{\leftarrow}w\} \triangleleft R$$

The meta-level renaming $O\langle t\{x{\leftarrow}z\}\rangle\{y{\leftarrow}w\}$ is not an issue since y can only appear free in $O\langle z\rangle$, which is always a subterm (up to α-renaming) of the initial term.

However, the renaming $t\{x{\leftarrow}z\}$ is problematic: t is not always a subterm of the initial term!

How big is the inefficiency?

Let $\tau_3 \coloneqq x[x{\leftarrow}yz][z{\leftarrow}yy]$ and $\tau_3' \coloneqq x'[x'{\leftarrow}y'z'][z'{\leftarrow}y'y']$ and so on.

Active code		Right ctx	
$x[x{\leftarrow}yz][z{\leftarrow}yy][y{\leftarrow}\lambda y'.\tau_3']$	$\triangleleft$	$\langle\cdot\rangle$	$\rightsquigarrow_{\mathsf{sea}_1}$
$x[x{\leftarrow}yz][z{\leftarrow}yy]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{e}}$
$x[x{\leftarrow}yz][z{\leftarrow}(\lambda y''.\tau_3'')y]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$=$
$x[x{\leftarrow}yz][z{\leftarrow}(\lambda y''.x''[x''{\leftarrow}y''z''][z''{\leftarrow}y''y''])y]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{m}}$
$x[x{\leftarrow}yx''][x''{\leftarrow}yz''][z''{\leftarrow}yy]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{e}}$
$x[x{\leftarrow}yx''][x''{\leftarrow}yz''][z''{\leftarrow}(\lambda y'''.\tau_3''')y]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{m}}$
$x[x{\leftarrow}yx''][x''{\leftarrow}yx'''][x'''{\leftarrow}yz'''][z'''{\leftarrow}yy]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{e}}$
$\vdots$			

How big is the inefficiency?

Let $\tau_3 \coloneqq x[x{\leftarrow}yz][z{\leftarrow}yy]$ and $\tau_3' \coloneqq x'[x'{\leftarrow}y'z'][z'{\leftarrow}y'y']$ and so on.

Active code		Right ctx	
$x[x{\leftarrow}yz][z{\leftarrow}yy][y{\leftarrow}\lambda y'.\tau_3']$	$\triangleleft$	$\langle\cdot\rangle$	$\rightsquigarrow_{\mathsf{sea}_1}$
$x[x{\leftarrow}yz][z{\leftarrow}yy]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{e}}$
$x[x{\leftarrow}yz][z{\leftarrow}(\lambda y''.\tau_3'')y]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$=$
$x[x{\leftarrow}yz][z{\leftarrow}(\lambda y''.x''[x''{\leftarrow}y''z''][z''{\leftarrow}y''y''])y]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{m}}$
$x[x{\leftarrow}yx''][x''{\leftarrow}yz''][z''{\leftarrow}yy]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{e}}$
$x[x{\leftarrow}yx''][x''{\leftarrow}yz''][z''{\leftarrow}(\lambda y'''.\tau_3''')y]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{m}}$
$x[x{\leftarrow}yx''][x''{\leftarrow}yx'''][x'''{\leftarrow}yz'''][z'''{\leftarrow}yy]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{e}}$
$\vdots$			

How big is the inefficiency?

Let $\tau_3 \coloneqq x[x{\leftarrow}yz][z{\leftarrow}yy]$ and $\tau_3' \coloneqq x'[x'{\leftarrow}y'z'][z'{\leftarrow}y'y']$ and so on.

Active code		Right ctx	
$x[x{\leftarrow}yz][z{\leftarrow}yy][y{\leftarrow}\lambda y'.\tau_3']$	$\triangleleft$	$\langle\cdot\rangle$	$\rightsquigarrow_{\mathsf{sea}_1}$
$x[x{\leftarrow}yz][z{\leftarrow}yy]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{e}}$
$x[x{\leftarrow}yz][z{\leftarrow}(\lambda y''.\tau_3'')y]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$=$
$x[x{\leftarrow}yz][z{\leftarrow}(\lambda y''.x''[x''{\leftarrow}y''z''][z''{\leftarrow}y''y''])y]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{m}}$
$x[x{\leftarrow}yx''][x''{\leftarrow}yz''][z''{\leftarrow}yy]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{e}}$
$x[x{\leftarrow}yx''][x''{\leftarrow}yz''][z''{\leftarrow}(\lambda y'''.\tau_3''')y]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{m}}$
$x[x{\leftarrow}yx''][x''{\leftarrow}yx'''][x'''{\leftarrow}yz'''][z'''{\leftarrow}yy]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{e}}$
$\vdots$			

How big is the inefficiency?

Let $\tau_3 \coloneqq x[x{\leftarrow}yz][z{\leftarrow}yy]$ and $\tau_3' \coloneqq x'[x'{\leftarrow}y'z'][z'{\leftarrow}y'y']$ and so on.

Active code		Right ctx	
$x[x{\leftarrow}yz][z{\leftarrow}yy][y{\leftarrow}\lambda y'.\tau_3']$	$\triangleleft$	$\langle\cdot\rangle$	$\rightsquigarrow_{\mathsf{sea}_1}$
$x[x{\leftarrow}yz][z{\leftarrow}yy]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{e}}$
$x[x{\leftarrow}yz][z{\leftarrow}(\lambda y''.\tau_3'')y]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$=$
$x[x{\leftarrow}yz][z{\leftarrow}(\lambda y''.x''[x''{\leftarrow}y''z''][z''{\leftarrow}y''y''])y]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{m}}$
$x[x{\leftarrow}yx''][x''{\leftarrow}yz''][z''{\leftarrow}yy]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{e}}$
$x[x{\leftarrow}yx''][x''{\leftarrow}yz''][z''{\leftarrow}(\lambda y'''.\tau_3''')y]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{m}}$
$x[x{\leftarrow}yx''][x''{\leftarrow}yx'''][x'''{\leftarrow}yz'''][z'''{\leftarrow}yy]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{e}}$
$\vdots$			

How big is the inefficiency?

Let $\tau_3 \coloneqq x[x{\leftarrow}yz][z{\leftarrow}yy]$ and $\tau_3' \coloneqq x'[x'{\leftarrow}y'z'][z'{\leftarrow}y'y']$ and so on.

Active code		Right ctx	
$x[x{\leftarrow}yz][z{\leftarrow}yy][y{\leftarrow}\lambda y'.\tau_3']$	$\triangleleft$	$\langle\cdot\rangle$	$\rightsquigarrow_{\mathsf{sea}_1}$
$x[x{\leftarrow}yz][z{\leftarrow}yy]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{e}}$
$x[x{\leftarrow}yz][z{\leftarrow}(\lambda y''.\tau_3'')y]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$=$
$x[x{\leftarrow}yz][z{\leftarrow}(\lambda y''.x''[x''{\leftarrow}y''z''][z''{\leftarrow}y''y''])y]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{m}}$
$x[x{\leftarrow}yx''][x''{\leftarrow}yz''][z''{\leftarrow}yy]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{e}}$
$x[x{\leftarrow}yx''][x''{\leftarrow}yz''][z''{\leftarrow}(\lambda y'''.\tau_3''')y]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{m}}$
$x[x{\leftarrow}yx''][x''{\leftarrow}yx'''][x'''{\leftarrow}yz'''][z'''{\leftarrow}yy]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{e}}$
$\vdots$			

How big is the inefficiency?

Let $\tau_3 \coloneqq x[x{\leftarrow}yz][z{\leftarrow}yy]$ and $\tau_3' \coloneqq x'[x'{\leftarrow}y'z'][z'{\leftarrow}y'y']$ and so on.

Active code		Right ctx	
$x[x{\leftarrow}yz][z{\leftarrow}yy][y{\leftarrow}\lambda y'.\tau_3']$	$\triangleleft$	$\langle\cdot\rangle$	$\rightsquigarrow_{\mathsf{sea}_1}$
$x[x{\leftarrow}yz][z{\leftarrow}yy]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{e}}$
$x[x{\leftarrow}yz][z{\leftarrow}(\lambda y''.\tau_3'')y]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$=$
$x[x{\leftarrow}yz][z{\leftarrow}(\lambda y''.x''[x''{\leftarrow}y''z''][z''{\leftarrow}y''y''])y]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{m}}$
$x[x{\leftarrow}yx''][x''{\leftarrow}yz''][z''{\leftarrow}yy]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{e}}$
$x[x{\leftarrow}yx''][x''{\leftarrow}yz''][z''{\leftarrow}(\lambda y'''.\tau_3''')y]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{m}}$
$x[x{\leftarrow}yx''][x''{\leftarrow}yx'''][x'''{\leftarrow}yz'''][z'''{\leftarrow}yy]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{e}}$
$\vdots$			

How big is the inefficiency?

Let $\tau_3 \coloneqq x[x{\leftarrow}yz][z{\leftarrow}yy]$ and $\tau_3' \coloneqq x'[x'{\leftarrow}y'z'][z'{\leftarrow}y'y']$ and so on.

Active code		Right ctx	
$x[x{\leftarrow}yz][z{\leftarrow}yy][y{\leftarrow}\lambda y'.\tau_3']$	$\triangleleft$	$\langle\cdot\rangle$	$\rightsquigarrow_{\mathsf{sea}_1}$
$x[x{\leftarrow}yz][z{\leftarrow}yy]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{e}}$
$x[x{\leftarrow}yz][z{\leftarrow}(\lambda y''.\tau_3'')y]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$=$
$x[x{\leftarrow}yz][z{\leftarrow}(\lambda y''.x''[x''{\leftarrow}y''z''][z''{\leftarrow}y''y''])y]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{m}}$
$x[x{\leftarrow}yx''][x''{\leftarrow}yz''][z''{\leftarrow}yy]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{e}}$
$x[x{\leftarrow}yx''][x''{\leftarrow}yz''][z''{\leftarrow}(\lambda y'''.\tau_3''')y]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{m}}$
$x[x{\leftarrow}yx''][x''{\leftarrow}yx'''][x'''{\leftarrow}yz'''][z'''{\leftarrow}yy]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{e}}$
$\vdots$			

How big is the inefficiency?

Let $\tau_3 \coloneqq x[x{\leftarrow}yz][z{\leftarrow}yy]$ and $\tau_3' \coloneqq x'[x'{\leftarrow}y'z'][z'{\leftarrow}y'y']$ and so on.

Active code		Right ctx	
$x[x{\leftarrow}yz][z{\leftarrow}yy][y{\leftarrow}\lambda y'.\tau_3']$	$\triangleleft$	$\langle\cdot\rangle$	$\rightsquigarrow_{\mathsf{sea}_1}$
$x[x{\leftarrow}yz][z{\leftarrow}yy]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{e}}$
$x[x{\leftarrow}yz][z{\leftarrow}(\lambda y''.\tau_3'')y]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$=$
$x[x{\leftarrow}yz][z{\leftarrow}(\lambda y''.x''[x''{\leftarrow}y''z''][z''{\leftarrow}y''y''])y]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{m}}$
$x[x{\leftarrow}yx''][x''{\leftarrow}yz''][z''{\leftarrow}yy]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{e}}$
$x[x{\leftarrow}yx''][x''{\leftarrow}yz''][z''{\leftarrow}(\lambda y'''.\tau_3''')y]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{m}}$
$x[x{\leftarrow}yx''][x''{\leftarrow}yx'''][x'''{\leftarrow}yz'''][z'''{\leftarrow}yy]$	$\triangleleft$	$[y{\leftarrow}\lambda y'.\tau_3']$	$\rightsquigarrow_{\mathsf{e}}$
$\vdots$			

Slicing to the rescue

$$t[x{\leftarrow}(\lambda y.O'\langle z\rangle)w] \triangleleft R \rightsquigarrow_{\mathsf{m}} O'\langle t\{x{\leftarrow}z\}\rangle\{y{\leftarrow}w\} \triangleleft R$$

The idea is to delay the merging of t and $O'\langle z\rangle$ as $O'\langle t\{x{\leftarrow}z\}\rangle$
↪ put the pair (**slice**) (t, x), denoted as $t[x{\leftarrow}\cdot]$, in a new stack storing delayed merges, and keeping $O'\langle z\rangle\{y{\leftarrow}w\}$ as the active code.

This leads to the following **Sliced POsitive Machine (Sliced POM)**.

The $\rightsquigarrow_{\mathsf{m}}$ transition of the natural POM is replaced by:

$$\begin{array}{l|l|l||c||l|l|l} \mathtt{S} & t[x{\leftarrow}(\lambda y.u)w] & \mathtt{E} & \rightsquigarrow_{\mathsf{m}} & \mathtt{S} : t[x{\leftarrow}\cdot] & u\{y{\leftarrow}w\} & \mathtt{E} \\ \mathtt{S} : t[x{\leftarrow}\cdot] & z & \mathtt{E} & \rightsquigarrow_{\mathsf{sea}_3} & \mathtt{S} & t\{x{\leftarrow}z\} & \mathtt{E} \end{array}$$

Slicing to the rescue

$$t[x{\leftarrow}(\lambda y.O'\langle z\rangle)w] \triangleleft R \rightsquigarrow_{\mathsf{m}} O'\langle t\{x{\leftarrow}z\}\rangle\{y{\leftarrow}w\} \triangleleft R$$

The idea is to delay the merging of t and $O'\langle z\rangle$ as $O'\langle t\{x{\leftarrow}z\}\rangle$
$\hookrightarrow$ put the pair (**slice**) (t, x), denoted as $t[x{\leftarrow}\cdot]$, in a new stack storing delayed merges, and keeping $O'\langle z\rangle\{y{\leftarrow}w\}$ as the active code.

This leads to the following **Sliced POsitive Machine (Sliced POM)**.

The $\rightsquigarrow_{\mathsf{m}}$ transition of the natural POM is replaced by:

S	$t[x{\leftarrow}(\lambda y.u)w]$	E	$\rightsquigarrow_{\mathsf{m}}$	S : $t[x{\leftarrow}\cdot]$	$u\{y{\leftarrow}w\}$	E
S : $t[x{\leftarrow}\cdot]$	z	E	$\rightsquigarrow_{\mathsf{sea}_3}$	S	$t\{x{\leftarrow}z\}$	E

Slicing to the rescue

$$t[x{\leftarrow}(\lambda y.O'\langle z\rangle)w] \triangleleft R \rightsquigarrow_{\mathsf{m}} O'\langle t\{x{\leftarrow}z\}\rangle\{y{\leftarrow}w\} \triangleleft R$$

The idea is to delay the merging of t and $O'\langle z\rangle$ as $O'\langle t\{x{\leftarrow}z\}\rangle$
$\hookrightarrow$ put the pair (**slice**) (t, x), denoted as $t[x{\leftarrow}\cdot]$, in a new stack storing delayed merges, and keeping $O'\langle z\rangle\{y{\leftarrow}w\}$ as the active code.

This leads to the following **Sliced POsitive Machine (Sliced POM)**.

The $\rightsquigarrow_{\mathsf{m}}$ transition of the natural POM is replaced by:

$$\begin{array}{l|l|l||c||l|l|l}
\mathtt{S} & t[x{\leftarrow}(\lambda y.u)w] & \mathtt{E} & \rightsquigarrow_{\mathsf{m}} & \mathtt{S} : t[x{\leftarrow}\cdot] & u\{y{\leftarrow}w\} & \mathtt{E} \\
\mathtt{S} : t[x{\leftarrow}\cdot] & z & \mathtt{E} & \rightsquigarrow_{\mathtt{sea}_3} & \mathtt{S} & t\{x{\leftarrow}z\} & \mathtt{E}
\end{array}$$

Sliced POM

$$
\begin{array}{l|l|l||l||l|l|l}
\mathtt{S} & t[x{\leftarrow}\lambda y.u] & \mathtt{E} & \rightsquigarrow_{\mathtt{sea}_1} & \mathtt{S} & t & [x{\leftarrow}\lambda y.u] : \mathtt{E} \\
\mathtt{S} & t[x{\leftarrow}yz] & \mathtt{E} & \rightsquigarrow_{\mathtt{sea}_2} & \mathtt{S} & t & [x{\leftarrow}yz] : \mathtt{E} \\
\mathtt{S} & t[x{\leftarrow}yz] & \mathtt{E} & \rightsquigarrow_{\mathtt{e}} & \mathtt{S} & t[x{\leftarrow}(\lambda w.u)^{\alpha} z] & \mathtt{E} \\
\mathtt{S} & t[x{\leftarrow}(\lambda y.u)w] & \mathtt{E} & \rightsquigarrow_{\mathtt{m}} & \mathtt{S} : t[x{\leftarrow}\cdot] & u\{y{\leftarrow}w\} & \mathtt{E} \\
\mathtt{S} : t[x{\leftarrow}\cdot] & z & \mathtt{E} & \rightsquigarrow_{\mathtt{sea}_3} & \mathtt{S} & t\{x{\leftarrow}z\} & \mathtt{E}
\end{array}
$$

Lemma (Sub-term property)

Let $t \lhd \mathtt{Q} \rightsquigarrow^* \mathtt{Q}' = (\mathtt{S}, u, \mathtt{E})$ be a Sliced POM run. Then:

1. *If $\mathtt{Q}'$ is not the target of an $\mathtt{e}$-transition then $|r| \leq |t|$ for any term r in $\mathtt{Q}'$;*
2. *Otherwise, $|r| \leq |t|$ for any term r in $\mathtt{Q}'$ except u.*

Lemma (Cost of single transitions)

Let $t \lhd \mathtt{Q} \rightsquigarrow^* \mathtt{Q}'$ be a Sliced POM run. Implementing $\mathtt{sea}_1$ and $\mathtt{sea}_2$ transitions from $\mathtt{Q}'$ costs $\mathcal{O}(1)$ each while implementing $\mathtt{e}$, $\mathtt{m}$, and $\mathtt{sea}_3$ transitions costs $\mathcal{O}(|t|)$ each.

Sliced POM

$$
\begin{array}{l|l|l||l||l|l|l}
\mathtt{S} & t[x{\leftarrow}\lambda y.u] & \mathtt{E} & \rightsquigarrow_{\mathtt{sea}_1} & \mathtt{S} & t & [x{\leftarrow}\lambda y.u] : \mathtt{E} \\
\mathtt{S} & t[x{\leftarrow}yz] & \mathtt{E} & \rightsquigarrow_{\mathtt{sea}_2} & \mathtt{S} & t & [x{\leftarrow}yz] : \mathtt{E} \\
\mathtt{S} & t[x{\leftarrow}yz] & \mathtt{E} & \rightsquigarrow_{\mathtt{e}} & \mathtt{S} & t[x{\leftarrow}(\lambda w.u)^{\alpha} z] & \mathtt{E} \\
\mathtt{S} & t[x{\leftarrow}(\lambda y.u)w] & \mathtt{E} & \rightsquigarrow_{\mathtt{m}} & \mathtt{S} : t[x{\leftarrow}\cdot] & u\{y{\leftarrow}w\} & \mathtt{E} \\
\mathtt{S} : t[x{\leftarrow}\cdot] & z & \mathtt{E} & \rightsquigarrow_{\mathtt{sea}_3} & \mathtt{S} & t\{x{\leftarrow}z\} & \mathtt{E}
\end{array}
$$

Lemma (Sub-term property)

Let $t \lhd \mathtt{Q} \rightsquigarrow^* \mathtt{Q}' = (\mathtt{S}, u, \mathtt{E})$ be a Sliced POM run. Then:

1. *If $\mathtt{Q}'$ is not the target of an* e*-transition then $|r| \leq |t|$ for any term r in $\mathtt{Q}'$;*
2. *Otherwise, $|r| \leq |t|$ for any term r in $\mathtt{Q}'$ except u.*

Lemma (Cost of single transitions)

Let $t \lhd \mathtt{Q} \rightsquigarrow^* \mathtt{Q}'$ be a Sliced POM run. Implementing $\mathtt{sea}_1$ *and* $\mathtt{sea}_2$ *transitions from $\mathtt{Q}'$ costs $\mathcal{O}(1)$ each while implementing* e, m, *and* $\mathtt{sea}_3$ *transitions costs $\mathcal{O}(|t|)$ each.*

Sliced POM

$$
\begin{array}{l|l|l||l||l|l|l||l}
\mathtt{S} & t[x{\leftarrow}\lambda y.u] & \mathtt{E} & \rightsquigarrow_{\mathtt{sea}_1} & \mathtt{S} & t & [x{\leftarrow}\lambda y.u] : \mathtt{E} \\
\mathtt{S} & t[x{\leftarrow}yz] & \mathtt{E} & \rightsquigarrow_{\mathtt{sea}_2} & \mathtt{S} & t & [x{\leftarrow}yz] : \mathtt{E} \\
\mathtt{S} & t[x{\leftarrow}yz] & \mathtt{E} & \rightsquigarrow_{\mathtt{e}} & \mathtt{S} & t[x{\leftarrow}(\lambda w.u)^{\alpha} z] & \mathtt{E} \\
\mathtt{S} & t[x{\leftarrow}(\lambda y.u)w] & \mathtt{E} & \rightsquigarrow_{\mathtt{m}} & \mathtt{S} : t[x{\leftarrow}\cdot] & u\{y{\leftarrow}w\} & \mathtt{E} \\
\mathtt{S} : t[x{\leftarrow}\cdot] & z & \mathtt{E} & \rightsquigarrow_{\mathtt{sea}_3} & \mathtt{S} & t\{x{\leftarrow}z\} & \mathtt{E}
\end{array}
$$

Lemma (Sub-term property)

Let $t \lhd \mathtt{Q} \rightsquigarrow^* \mathtt{Q}' = (\mathtt{S}, u, \mathtt{E})$ *be a Sliced POM run. Then:*

1. *If* $\mathtt{Q}'$ *is not the target of an* e*-transition then* $|r| \leq |t|$ *for any term* r *in* $\mathtt{Q}'$*;*
2. *Otherwise,* $|r| \leq |t|$ *for any term* r *in* $\mathtt{Q}'$ *except* u*.*

Lemma (Cost of single transitions)

Let $t \lhd \mathtt{Q} \rightsquigarrow^* \mathtt{Q}'$ *be a Sliced POM run. Implementing* $\mathtt{sea}_1$ *and* $\mathtt{sea}_2$ *transitions from* $\mathtt{Q}'$ *costs* $\mathcal{O}(1)$ *each while implementing* e, m, *and* $\mathtt{sea}_3$ *transitions costs* $\mathcal{O}(|t|)$ *each.*

Sliced POM

$$
\begin{array}{l|l|l||l||l|l|l}
\mathtt{S} & t[x{\leftarrow}\lambda y.u] & \mathtt{E} & \rightsquigarrow_{\mathtt{sea}_1} & \mathtt{S} & t & [x{\leftarrow}\lambda y.u] : \mathtt{E} \\
\mathtt{S} & t[x{\leftarrow}yz] & \mathtt{E} & \rightsquigarrow_{\mathtt{sea}_2} & \mathtt{S} & t & [x{\leftarrow}yz] : \mathtt{E} \\
\mathtt{S} & t[x{\leftarrow}yz] & \mathtt{E} & \rightsquigarrow_{\mathtt{e}} & \mathtt{S} & t[x{\leftarrow}(\lambda w.u)^{\alpha} z] & \mathtt{E} \\
\mathtt{S} & t[x{\leftarrow}(\lambda y.u)w] & \mathtt{E} & \rightsquigarrow_{\mathtt{m}} & \mathtt{S} : t[x{\leftarrow}\cdot] & u\{y{\leftarrow}w\} & \mathtt{E} \\
\mathtt{S} : t[x{\leftarrow}\cdot] & z & \mathtt{E} & \rightsquigarrow_{\mathtt{sea}_3} & \mathtt{S} & t\{x{\leftarrow}z\} & \mathtt{E}
\end{array}
$$

Lemma (Number of transitions)

Let $r : t \lhd \mathtt{Q} \rightsquigarrow^* \mathtt{Q}'$ be a Sliced POM run.

1. $|r|_{\mathtt{e},\mathtt{sea}_3} \in \mathcal{O}(|r|_{\mathtt{m}})$;
2. $|r|_{\mathtt{sea}_1,\mathtt{sea}_2} \in \mathcal{O}(|t| \cdot (|r|_{\mathtt{m}} + 1))$.

Lemma (Cost of single transitions)

Let $t \lhd \mathtt{Q} \rightsquigarrow^* \mathtt{Q}'$ be a Sliced POM run. Implementing $\mathtt{sea}_1$ and $\mathtt{sea}_2$ transitions from $\mathtt{Q}'$ costs $\mathcal{O}(1)$ each while implementing $\mathtt{e}$, $\mathtt{m}$, and $\mathtt{sea}_3$ transitions costs $\mathcal{O}(|t|)$ each.

Sliced POM

$$
\begin{array}{l|l|l||c||l|l||l}
\mathtt{S} & t[x{\leftarrow}\lambda y.u] & \mathtt{E} & \rightsquigarrow_{\mathsf{sea}_1} & \mathtt{S} & t & [x{\leftarrow}\lambda y.u] : \mathtt{E} \\
\mathtt{S} & t[x{\leftarrow}yz] & \mathtt{E} & \rightsquigarrow_{\mathsf{sea}_2} & \mathtt{S} & t & [x{\leftarrow}yz] : \mathtt{E} \\
\mathtt{S} & t[x{\leftarrow}yz] & \mathtt{E} & \rightsquigarrow_{\mathsf{e}} & \mathtt{S} & t[x{\leftarrow}(\lambda w.u)^{\alpha} z] & \mathtt{E} \\
\mathtt{S} & t[x{\leftarrow}(\lambda y.u)w] & \mathtt{E} & \rightsquigarrow_{\mathsf{m}} & \mathtt{S} : t[x{\leftarrow}\cdot] & u\{y{\leftarrow}w\} & \mathtt{E} \\
\mathtt{S} : t[x{\leftarrow}\cdot] & z & \mathtt{E} & \rightsquigarrow_{\mathsf{sea}_3} & \mathtt{S} & t\{x{\leftarrow}z\} & \mathtt{E}
\end{array}
$$

Theorem

Let $r : t \lhd \mathtt{Q} \rightsquigarrow^* \mathtt{Q}'$ be a Sliced POM run. Then r can be implemented on random access machines in $\mathcal{O}(|t| \cdot (|r|_{\mathsf{m}} + 1))$.

Conclusions and future work

We have provided an efficient implementation of λ_{pos} using slice stacks that are dual to the usual structure of machine environments, justifying its value as a tool for the study of sharing.

Some future directions:

- Adapt λ_{pos} and the sliced POM to call-by-need
- Combining the slicing technique to other implementation techniques on abstract machines

Conclusions and future work

We have provided an efficient implementation of $\lambda_{\mathtt{pos}}$ using slice stacks that are dual to the usual structure of machine environments, justifying its value as a tool for the study of sharing.

Some future directions:

- Adapt $\lambda_{\mathtt{pos}}$ and the sliced POM to call-by-need
- Combining the slicing technique to other implementation techniques on abstract machines

Thank you for your listening!