Positive Focusing is Directly Useful

Jui-Hsuan Wu (Ray) and Beniamino Accattoli

LIX, Ecole Polytechnique & Inria Saclay

MFPS 2024
University of Oxford, UK

21 June 2024

Background

Sharing is important.

But there is no sharing in the λ-calculus.

The simplest way to introduce sharing in the λ-calculus is *subterm* sharing.

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

In a call-by-value setting, general applications tu become somewhat redundant.
$\hookrightarrow$ It is possible to restrict the shape of applications.

Background

Sharing is important.

But there is no sharing in the λ-calculus.

The simplest way to introduce sharing in the λ-calculus is *subterm* sharing.

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

In a call-by-value setting, general applications tu become somewhat redundant.
$\hookrightarrow$ It is possible to restrict the shape of applications.

Background

Sharing is important.

But there is no sharing in the λ-calculus.

The simplest way to introduce sharing in the λ-calculus is *subterm* sharing.

$$t, u ::= x \mid tu \mid \lambda x.t \mid \texttt{let } x \texttt{ = } u \texttt{ in } t$$

In a call-by-value setting, general applications *tu* become somewhat redundant.
$\hookrightarrow$ It is possible to restrict the shape of applications.

Background

Sharing is important.

But there is no sharing in the λ-calculus.

The simplest way to introduce sharing in the λ-calculus is *subterm* sharing.

$$t, u ::= x \mid tu \mid \lambda x.t \mid t[x \leftarrow u] \text{ (explicit substitution)}$$

In a call-by-value setting, general applications *tu* become somewhat redundant.
$\hookrightarrow$ It is possible to restrict the shape of applications.

Background

Sharing is important.

But there is no sharing in the λ-calculus.

The simplest way to introduce sharing in the λ-calculus is *subterm* sharing.

$$t, u ::= x \mid tu \mid \lambda x.t \mid t[x\leftarrow u] \text{ (explicit substitution)}$$

In a call-by-value setting, general applications tu become somewhat redundant.
$\hookrightarrow$ It is possible to restrict the shape of applications.

Shape of applications in a CbV setting

In CbV, there are many possible ways to restrict the shape of applications:

$$\begin{tikzcd}
 & (x[\![u]\!])[x\leftarrow[\![t]\!]] \\
tu \arrow[ur, bend left, "[\![\cdot]\!]"] \arrow[r, "[\![\cdot]\!]"] \arrow[dr, bend right, "[\![\cdot]\!]"'] & ([\![t]\!]x)[x\leftarrow[\![u]\!]] \\
 & (xy)[x\leftarrow[\![t]\!]][y\leftarrow[\![u]\!]]
\end{tikzcd}$$

These restrictions are typical in a call-by-value setting, as substitutions of applications sometimes are simply blocked by the syntax:

Shape of applications in a CbV setting

In CbV, there are many possible ways to restrict the shape of applications:

$$
\begin{tikzcd}
 & (x\llbracket u\rrbracket)[x\leftarrow\llbracket t\rrbracket] \\
tu \arrow[ur, bend left, "\llbracket\cdot\rrbracket"] \arrow[r, "\llbracket\cdot\rrbracket"] \arrow[dr, bend right, "\llbracket\cdot\rrbracket"'] & (\llbracket t\rrbracket x)[x\leftarrow\llbracket u\rrbracket] \\
 & (xy)[x\leftarrow\llbracket t\rrbracket][y\leftarrow\llbracket u\rrbracket]
\end{tikzcd}
$$

These restrictions are typical in a call-by-value setting, as substitutions of applications sometimes are simply blocked by the syntax:

Shape of applications in a CbV setting

In CbV, there are many possible ways to restrict the shape of applications:

$$\begin{tikzcd}
 & (x\llbracket u\rrbracket)[x\leftarrow\llbracket t\rrbracket] \\
tu \arrow[ur, bend left, "\llbracket\cdot\rrbracket"] \arrow[r, "\llbracket\cdot\rrbracket"] \arrow[dr, bend right, "\llbracket\cdot\rrbracket"'] & (\llbracket t\rrbracket x)[x\leftarrow\llbracket u\rrbracket] \\
 & (xy)[x\leftarrow\llbracket t\rrbracket][y\leftarrow\llbracket u\rrbracket]
\end{tikzcd}$$

These restrictions are typical in a call-by-value setting, as substitutions of applications sometimes are simply blocked by the syntax:

Shape of applications in a CbV setting

In CbV, there are many possible ways to restrict the shape of applications:

$$\begin{tikzcd} & (x\llbracket u\rrbracket)[x\leftarrow\llbracket t\rrbracket] \\ tu \arrow[ur, bend left, "\llbracket\cdot\rrbracket"] \arrow[r, "\llbracket\cdot\rrbracket"] \arrow[dr, bend right, "\llbracket\cdot\rrbracket"'] & (\llbracket t\rrbracket x)[x\leftarrow\llbracket u\rrbracket] \\ & (xy)[x\leftarrow\llbracket t\rrbracket][y\leftarrow\llbracket u\rrbracket] \end{tikzcd}$$

value as the left subterm of an application

These restrictions are typical in a call-by-value setting, as substitutions of applications sometimes are simply blocked by the syntax:

Shape of applications in a CbV setting

In CbV, there are many possible ways to restrict the shape of applications:

$$\begin{tikzcd} & (x\llbracket u\rrbracket)[x\leftarrow\llbracket t\rrbracket] \\ tu \arrow[ur, bend left, "\llbracket\cdot\rrbracket"] \arrow[r, "\llbracket\cdot\rrbracket"] \arrow[dr, bend right, "\llbracket\cdot\rrbracket"'] & (\llbracket t\rrbracket x)[x\leftarrow\llbracket u\rrbracket] \\ & (xy)[x\leftarrow\llbracket t\rrbracket][y\leftarrow\llbracket u\rrbracket] \end{tikzcd}$$

value as the left subterm of an application

These restrictions are typical in a call-by-value setting, as substitutions of applications sometimes are simply blocked by the syntax:

Shape of applications in a CbV setting

In CbV, there are many possible ways to restrict the shape of applications:

$$\begin{tikzcd}
 & (x\llbracket u\rrbracket)[x\leftarrow\llbracket t\rrbracket] \\
tu \arrow[ur, bend left, "\llbracket\cdot\rrbracket"] \arrow[r, "\llbracket\cdot\rrbracket"] \arrow[dr, bend right, "\llbracket\cdot\rrbracket"'] & (\llbracket t\rrbracket x)[x\leftarrow\llbracket u\rrbracket] \\
 & (xy)[x\leftarrow\llbracket t\rrbracket][y\leftarrow\llbracket u\rrbracket]
\end{tikzcd}$$

value as the left subterm of an application

value as the right subterm $\cdots$

These restrictions are typical in a call-by-value setting, as substitutions of applications sometimes are simply blocked by the syntax:

Shape of applications in a CbV setting

In CbV, there are many possible ways to restrict the shape of applications:

$$\begin{tikzcd} & (x\llbracket u\rrbracket)[x\leftarrow\llbracket t\rrbracket] \\ tu \arrow[ur, bend left, "\llbracket\cdot\rrbracket"] \arrow[r, "\llbracket\cdot\rrbracket"] \arrow[dr, bend right, "\llbracket\cdot\rrbracket"'] & (\llbracket t\rrbracket x)[x\leftarrow\llbracket u\rrbracket] \\ & (xy)[x\leftarrow\llbracket t\rrbracket][y\leftarrow\llbracket u\rrbracket] \end{tikzcd}$$

value as the left subterm of an application

value as the right subterm $\cdots$

These restrictions are typical in a call-by-value setting, as substitutions of applications sometimes are simply blocked by the syntax:

Shape of applications in a CbV setting

In CbV, there are many possible ways to restrict the shape of applications:

$$\begin{tikzcd} & (x\llbracket u\rrbracket)[x\leftarrow\llbracket t\rrbracket] \\ tu \arrow[ur, bend left, "\llbracket \cdot \rrbracket"] \arrow[r, "\llbracket \cdot \rrbracket"] \arrow[dr, bend right, "\llbracket \cdot \rrbracket"'] & (\llbracket t\rrbracket x)[x\leftarrow\llbracket u\rrbracket] \\ & (xy)[x\leftarrow\llbracket t\rrbracket][y\leftarrow\llbracket u\rrbracket] \end{tikzcd}$$

$(x\llbracket u\rrbracket)[x\leftarrow\llbracket t\rrbracket]$	value as the left subterm of an application
$(\llbracket t\rrbracket x)[x\leftarrow\llbracket u\rrbracket]$	value as the right subterm $\cdots$
$(xy)[x\leftarrow\llbracket t\rrbracket][y\leftarrow\llbracket u\rrbracket]$	values as both subterms $\cdots$

These restrictions are typical in a call-by-value setting, as substitutions of applications sometimes are simply blocked by the syntax:

Shape of applications in a CbV setting

In CbV, there are many possible ways to restrict the shape of applications:

$$tu \xrightarrow{\llbracket \cdot \rrbracket} (x\llbracket u \rrbracket)[x\leftarrow\llbracket t \rrbracket]$$

value as the left subterm of an application

$$tu \xrightarrow{\llbracket \cdot \rrbracket} (\llbracket t \rrbracket x)[x\leftarrow\llbracket u \rrbracket]$$

value as the right subterm $\cdots$

$$tu \xrightarrow{\llbracket \cdot \rrbracket} (xy)[x\leftarrow\llbracket t \rrbracket][y\leftarrow\llbracket u \rrbracket]$$

values as both subterms $\cdots$

These restrictions are typical in a call-by-value setting, as substitutions of applications sometimes are simply blocked by the syntax:

Shape of applications in a CbV setting

In CbV, there are many possible ways to restrict the shape of applications:

$$
\begin{tikzcd}[column sep=huge]
 & (x\llbracket u\rrbracket)[x\leftarrow\llbracket t\rrbracket] \\
tu \arrow[ur, bend left, "\llbracket\cdot\rrbracket"] \arrow[r, "\llbracket\cdot\rrbracket"] \arrow[dr, bend right, "\llbracket\cdot\rrbracket"'] & (\llbracket t\rrbracket x)[x\leftarrow\llbracket u\rrbracket] \\
 & (xy)[x\leftarrow\llbracket t\rrbracket][y\leftarrow\llbracket u\rrbracket]
\end{tikzcd}
$$

value as the left subterm of an application

value as the right subterm ⋯

values as both subterms ⋯

These restrictions are typical in a call-by-value setting, as substitutions of applications sometimes are simply blocked by the syntax:

$$xy \xrightarrow[\text{substituting } zw \text{ for } x]{} $$

Shape of applications in a CbV setting

In CbV, there are many possible ways to restrict the shape of applications:

$$\begin{tikzcd}[column sep=large] & (x\llbracket u\rrbracket)[x\leftarrow\llbracket t\rrbracket] & \text{value as the left subterm of an application} \\ tu \arrow[ur, bend left, "\llbracket\cdot\rrbracket"] \arrow[r, "\llbracket\cdot\rrbracket"] \arrow[dr, bend right, "\llbracket\cdot\rrbracket"'] & (\llbracket t\rrbracket x)[x\leftarrow\llbracket u\rrbracket] & \text{value as the right subterm} \cdots \\ & (xy)[x\leftarrow\llbracket t\rrbracket][y\leftarrow\llbracket u\rrbracket] & \text{values as both subterms} \cdots \end{tikzcd}$$

These restrictions are typical in a call-by-value setting, as substitutions of applications sometimes are simply blocked by the syntax:

$$xy \xrightarrow[\text{substituting } zw \text{ for } x]{} (zw)y$$

Shape of applications in a CbV setting

In CbV, there are many possible ways to restrict the shape of applications:

$$tu \xrightarrow{[\![\,\cdot\,]\!]} (x[\![u]\!])[x\leftarrow[\![t]\!]]$$

value as the left subterm of an application

$$tu \xrightarrow{[\![\,\cdot\,]\!]} ([\![t]\!]x)[x\leftarrow[\![u]\!]]$$

value as the right subterm $\cdots$

$$tu \xrightarrow{[\![\,\cdot\,]\!]} (xy)[x\leftarrow[\![t]\!]][y\leftarrow[\![u]\!]]$$

values as both subterms $\cdots$

These restrictions are typical in a call-by-value setting, as substitutions of applications sometimes are simply blocked by the syntax:

$$xy \;\cancel{\xrightarrow[\text{substituting } zw \text{ for } x]{\qquad\qquad}}\; (zw)y$$

Classification/design of call-by-value calculi with ESs

It is actually possible to have only variables as immediate sub-terms of applications

Now we have nine different forms of applications:

- the general form tu
- eight crumbled forms vu, xu, tv', vv', xv', ty, vy, and xy.

Some more ways to classify/design call-by-value calculi with ESs.

- Nested or flattened ESs: $t[x{\leftarrow}u[y{\leftarrow}r]]$ vs. $t[x{\leftarrow}u][y{\leftarrow}r]$
- Small-step vs. micro-step substitutions:

$$(xx)[x{\leftarrow}\mathsf{I}] \to \mathsf{II}$$
$$\text{vs.}$$
$$(xx)[x{\leftarrow}\mathsf{I}] \to (\mathsf{I}x)[x{\leftarrow}\mathsf{I}] \to (\mathsf{II})[x{\leftarrow}\mathsf{I}] \to \mathsf{II}$$

- Variables as values?

Classification/design of call-by-value calculi with ESs

It is actually possible to have only variables as immediate sub-terms of applications

Now we have nine different forms of applications:

- the general form tu
- eight crumbled forms vu, xu, tv', vv', xv', ty, vy, and xy.

Some more ways to classify/design call-by-value calculi with ESs.

- Nested or flattened ESs: $t[x{\leftarrow}u[y{\leftarrow}r]]$ vs. $t[x{\leftarrow}u][y{\leftarrow}r]$
- Small-step vs. micro-step substitutions:

$$(xx)[x{\leftarrow}\mathsf{I}] \to \mathsf{I}\mathsf{I}$$

vs.

$$(xx)[x{\leftarrow}\mathsf{I}] \to (\mathsf{I}x)[x{\leftarrow}\mathsf{I}] \to (\mathsf{I}\mathsf{I})[x{\leftarrow}\mathsf{I}] \to \mathsf{I}\mathsf{I}$$

- Variables as values?

Classification/design of call-by-value calculi with ESs

It is actually possible to have only variables as immediate sub-terms of applications

Now we have nine different forms of applications:

- the general form tu
- eight crumbled forms vu, xu, tv', vv', xv', ty, vy, and xy.

Some more ways to classify/design call-by-value calculi with ESs.

- Nested or flattened ESs: $t[x{\leftarrow}u[y{\leftarrow}r]]$ vs. $t[x{\leftarrow}u][y{\leftarrow}r]$
- Small-step vs. micro-step substitutions:

$$(xx)[x{\leftarrow}\mathsf{I}] \to \mathsf{II}$$

vs.

$$(xx)[x{\leftarrow}\mathsf{I}] \to (\mathsf{I}x)[x{\leftarrow}\mathsf{I}] \to (\mathsf{II})[x{\leftarrow}\mathsf{I}] \to \mathsf{II}$$

- Variables as values?

Classification/design of call-by-value calculi with ESs

It is actually possible to have only variables as immediate sub-terms of applications

Now we have nine different forms of applications:

- the general form tu
- eight crumbled forms vu, xu, tv', vv', xv', ty, vy, and xy.

Some more ways to classify/design call-by-value calculi with ESs.

- Nested or flattened ESs: $t[x{\leftarrow}u[y{\leftarrow}r]]$ vs. $t[x{\leftarrow}u][y{\leftarrow}r]$
- Small-step vs. micro-step substitutions:

$$(xx)[x{\leftarrow}\mathsf{I}] \to \mathsf{II}$$

vs.

$$(xx)[x{\leftarrow}\mathsf{I}] \to (\mathsf{I}x)[x{\leftarrow}\mathsf{I}] \to (\mathsf{II})[x{\leftarrow}\mathsf{I}] \to \mathsf{II}$$

- Variables as values?

Classification/design of call-by-value calculi with ESs

It is actually possible to have only variables as immediate sub-terms of applications

Now we have nine different forms of applications:

- the general form tu
- eight crumbled forms vu, xu, tv', vv', xv', ty, vy, and xy.

Some more ways to classify/design call-by-value calculi with ESs.

- Nested or flattened ESs: $t[x{\leftarrow}u[y{\leftarrow}r]]$ vs. $t[x{\leftarrow}u][y{\leftarrow}r]$
- Small-step vs. micro-step substitutions:

$$(xx)[x{\leftarrow}\mathsf{I}] \to \mathsf{II}$$

vs.

$$(xx)[x{\leftarrow}\mathsf{I}] \to (\mathsf{I}x)[x{\leftarrow}\mathsf{I}] \to (\mathsf{II})[x{\leftarrow}\mathsf{I}] \to \mathsf{II}$$

- Variables as values?

Classification/design of call-by-value calculi with ESs

It is actually possible to have only variables as immediate sub-terms of applications

Now we have nine different forms of applications:

- the general form tu
- eight crumbled forms vu, xu, tv', vv', xv', ty, vy, and xy.

Some more ways to classify/design call-by-value calculi with ESs.

- Nested or flattened ESs: $t[x{\leftarrow}u[y{\leftarrow}r]]$ vs. $t[x{\leftarrow}u][y{\leftarrow}r]$
- Small-step vs. micro-step substitutions:

$$(xx)[x{\leftarrow}\mathsf{I}] \to \mathsf{I}\mathsf{I}$$

vs.

$$(xx)[x{\leftarrow}\mathsf{I}] \to (\mathsf{I}x)[x{\leftarrow}\mathsf{I}] \to (\mathsf{I}\mathsf{I})[x{\leftarrow}\mathsf{I}] \to \mathsf{I}\mathsf{I}$$

- Variables as values?

Positive Focusing is Directly **Useful**

Useful substitutions v.s. non-useful substitutions

In micro-step settings, one has the following substitution rule:

$$C\langle x\rangle[x\leftarrow v] \to C\langle v\rangle[x\leftarrow v]$$

What about making a substitution only when it contributes to the creation of some β-redexes?

Consider

$$(yx)[x\leftarrow \mathsf{I}] \to (y\mathsf{I})[x\leftarrow \mathsf{I}]$$

There is no β-redex created after this substitution, and there won't be any β-redex created in the future → non-useful

Some more examples:

- $(xy)[x\leftarrow \mathsf{I}] \to (\mathsf{I}y)[x\leftarrow \mathsf{I}]$ is useful
- $x[x\leftarrow \mathsf{I}] \to \mathsf{I}[x\leftarrow \mathsf{I}]$ is non-useful
- $(xx)[x\leftarrow \mathsf{I}]$

Useful substitutions v.s. non-useful substitutions

In micro-step settings, one has the following substitution rule:

$$C\langle x\rangle[x{\leftarrow}v] \to C\langle v\rangle[x{\leftarrow}v]$$

What about making a substitution only when it contributes to the creation of some β-redexes?

Consider

$$(yx)[x{\leftarrow}\mathsf{I}] \to (y\mathsf{I})[x{\leftarrow}\mathsf{I}]$$

There is no β-redex created after this substitution, and there won't be any β-redex created in the future $\to$ non-useful

Some more examples:

- $(xy)[x{\leftarrow}\mathsf{I}] \to (\mathsf{I}y)[x{\leftarrow}\mathsf{I}]$ is useful
- $x[x{\leftarrow}\mathsf{I}] \to \mathsf{I}[x{\leftarrow}\mathsf{I}]$ is non-useful
- $(xx)[x{\leftarrow}\mathsf{I}]$

Useful substitutions v.s. non-useful substitutions

In micro-step settings, one has the following substitution rule:

$$C\langle x\rangle[x{\leftarrow}v] \to C\langle v\rangle[x{\leftarrow}v]$$

What about making a substitution only when it contributes to the creation of some β-redexes?

Consider

$$(yx)[x{\leftarrow}\mathsf{I}] \to (y\mathsf{I})[x{\leftarrow}\mathsf{I}]$$

There is no β-redex created after this substitution, and there won't be any β-redex created in the future $\to$ non-useful

Some more examples:

- $(xy)[x{\leftarrow}\mathsf{I}] \to (\mathsf{I}y)[x{\leftarrow}\mathsf{I}]$ is useful
- $x[x{\leftarrow}\mathsf{I}] \to \mathsf{I}[x{\leftarrow}\mathsf{I}]$ is non-useful
- $(xx)[x{\leftarrow}\mathsf{I}]$

Useful substitutions v.s. non-useful substitutions

In micro-step settings, one has the following substitution rule:

$$C\langle x\rangle[x\leftarrow v] \to C\langle v\rangle[x\leftarrow v]$$

What about making a substitution only when it contributes to the creation of some β-redexes?

Consider

$$(yx)[x\leftarrow \mathsf{I}] \to (y\mathsf{I})[x\leftarrow \mathsf{I}]$$

There is no β-redex created after this substitution, and there won't be any β-redex created in the future → non-useful

Some more examples:

- $(xy)[x\leftarrow \mathsf{I}] \to (\mathsf{I}y)[x\leftarrow \mathsf{I}]$ is useful
- $x[x\leftarrow \mathsf{I}] \to \mathsf{I}[x\leftarrow \mathsf{I}]$ is non-useful
- $(xx)[x\leftarrow \mathsf{I}]$

Useful substitutions v.s. non-useful substitutions

In micro-step settings, one has the following substitution rule:

$$C\langle x\rangle[x{\leftarrow}v] \to C\langle v\rangle[x{\leftarrow}v]$$

What about making a substitution only when it contributes to the creation of some β-redexes?

Consider

$$(yx)[x{\leftarrow}\mathsf{I}] \to (y\mathsf{I})[x{\leftarrow}\mathsf{I}]$$

There is no β-redex created after this substitution, and there won't be any β-redex created in the future → non-useful

Some more examples:

- $(xy)[x{\leftarrow}\mathsf{I}] \to (\mathsf{I}y)[x{\leftarrow}\mathsf{I}]$ is useful
- $x[x{\leftarrow}\mathsf{I}] \to \mathsf{I}[x{\leftarrow}\mathsf{I}]$ is non-useful
- $(xx)[x{\leftarrow}\mathsf{I}]$

Useful substitutions v.s. non-useful substitutions

In micro-step settings, one has the following substitution rule:

$$C\langle x\rangle[x\leftarrow v] \to C\langle v\rangle[x\leftarrow v]$$

What about making a substitution only when it contributes to the creation of some β-redexes?

Consider

$$(yx)[x\leftarrow \mathsf{I}] \to (y\mathsf{I})[x\leftarrow \mathsf{I}]$$

There is no β-redex created after this substitution, and there won't be any β-redex created in the future → non-useful

Some more examples:

- $(xy)[x\leftarrow \mathsf{I}] \to (\mathsf{I}y)[x\leftarrow \mathsf{I}]$ is (directly) useful
- $x[x\leftarrow \mathsf{I}] \to \mathsf{I}[x\leftarrow \mathsf{I}]$ is non-useful
- $(xx)[x\leftarrow \mathsf{I}]$

Useful substitutions v.s. non-useful substitutions

In micro-step settings, one has the following substitution rule:

$$C\langle x\rangle[x\leftarrow v] \to C\langle v\rangle[x\leftarrow v]$$

What about making a substitution only when it contributes to the creation of some β-redexes?

Consider

$$(yx)[x\leftarrow \mathsf{I}] \to (y\mathsf{I})[x\leftarrow \mathsf{I}]$$

There is no β-redex created after this substitution, and there won't be any β-redex created in the future → non-useful

Some more examples:

- $(xy)[x\leftarrow \mathsf{I}] \to (\mathsf{I}y)[x\leftarrow \mathsf{I}]$ is (directly) useful
- $x[x\leftarrow \mathsf{I}] \to \mathsf{I}[x\leftarrow \mathsf{I}]$ is non-useful
- $(xx)[x\leftarrow \mathsf{I}]$

Usefulness: subtleties

- Contextual closure:
  $x[x{\leftarrow}\mathsf{I}] \to \mathsf{I}[x{\leftarrow}\mathsf{I}]$ is non-useful
  while $x[x{\leftarrow}\mathsf{I}]y \to \mathsf{I}[x{\leftarrow}\mathsf{I}]y$ is useful
- Indirect usefulness:
  $(xy)[x{\leftarrow}z][z{\leftarrow}\mathsf{I}] \to (xy)[x{\leftarrow}\mathsf{I}][z{\leftarrow}\mathsf{I}]$
  $\hookrightarrow$ It is useful!
- Renaming chains:

$$\begin{array}{rl} & (x_0\,t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}x_k][x_k{\leftarrow}\mathsf{I}] \\ \to & (x_0\,t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}] \\ \to^* & (x_0\,t)[x_0{\leftarrow}\mathsf{I}][x_1{\leftarrow}\mathsf{I}]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}] \end{array}$$

Usefulness: subtleties

- Contextual closure:
  $x[x{\leftarrow}\mathsf{I}] \to \mathsf{I}[x{\leftarrow}\mathsf{I}]$ is non-useful
  while $x[x{\leftarrow}\mathsf{I}]y \to \mathsf{I}[x{\leftarrow}\mathsf{I}]y$ is useful

- Indirect usefulness:
  $(xy)[x{\leftarrow}z][z{\leftarrow}\mathsf{I}] \to (xy)[x{\leftarrow}\mathsf{I}][z{\leftarrow}\mathsf{I}]$
  $\hookrightarrow$ It is useful!

- Renaming chains:

$$
\begin{array}{rl}
& (x_0\,t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}x_k][x_k{\leftarrow}\mathsf{I}] \\
\to & (x_0\,t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}] \\
\to^* & (x_0\,t)[x_0{\leftarrow}\mathsf{I}][x_1{\leftarrow}\mathsf{I}]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}]
\end{array}
$$

Usefulness: subtleties

- Contextual closure:
  $x[x{\leftarrow}\mathsf{I}] \to \mathsf{I}[x{\leftarrow}\mathsf{I}]$ is non-useful
  while $x[x{\leftarrow}\mathsf{I}]y \to \mathsf{I}[x{\leftarrow}\mathsf{I}]y$ is useful
- Indirect usefulness:
  $(xy)[x{\leftarrow}z][z{\leftarrow}\mathsf{I}] \to (xy)[x{\leftarrow}\mathsf{I}][z{\leftarrow}\mathsf{I}]$
  $\hookrightarrow$ It is useful!
- Renaming chains:

$$\begin{array}{ll} & (x_0\,t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}x_k][x_k{\leftarrow}\mathsf{I}] \\ \to & (x_0\,t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}] \\ \to^* & (x_0\,t)[x_0{\leftarrow}\mathsf{I}][x_1{\leftarrow}\mathsf{I}]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}] \end{array}$$

Usefulness: subtleties

- Contextual closure:
  $x[x{\leftarrow}\mathsf{I}] \to \mathsf{I}[x{\leftarrow}\mathsf{I}]$ is non-useful
  while $x[x{\leftarrow}\mathsf{I}]y \to \mathsf{I}[x{\leftarrow}\mathsf{I}]y$ is useful

- Indirect usefulness:
  $(xy)[x{\leftarrow}z][z{\leftarrow}\mathsf{I}] \to (xy)[x{\leftarrow}\mathsf{I}][z{\leftarrow}\mathsf{I}]$
  $\hookrightarrow$ It is useful!

- Renaming chains:

$$
\begin{array}{rl}
 & (x_0\,t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}x_k][x_k{\leftarrow}\mathsf{I}] \\
\to & (x_0\,t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}] \\
\to^* & (x_0\,t)[x_0{\leftarrow}\mathsf{I}][x_1{\leftarrow}\mathsf{I}]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}]
\end{array}
$$

Usefulness: subtleties

- Contextual closure:
  $x[x{\leftarrow}\mathsf{I}] \to \mathsf{I}[x{\leftarrow}\mathsf{I}]$ is non-useful
  while $x[x{\leftarrow}\mathsf{I}]y \to \underline{\mathsf{I}}[x{\leftarrow}\mathsf{I}]\underline{y}$ is useful

- Indirect usefulness:
  $(xy)[x{\leftarrow}z][z{\leftarrow}\mathsf{I}] \to (xy)[x{\leftarrow}\mathsf{I}][z{\leftarrow}\mathsf{I}]$
  $\hookrightarrow$ It is useful!

- Renaming chains:

$$\begin{array}{ll} & (x_0\,t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}x_k][x_k{\leftarrow}\mathsf{I}] \\ \to & (x_0\,t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}] \\ \to^* & (x_0\,t)[x_0{\leftarrow}\mathsf{I}][x_1{\leftarrow}\mathsf{I}]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}] \end{array}$$

Usefulness: subtleties

- Contextual closure:
  $x[x{\leftarrow}\mathsf{I}] \to \mathsf{I}[x{\leftarrow}\mathsf{I}]$ is non-useful
  while $x[x{\leftarrow}\mathsf{I}]y \to \underline{\mathsf{I}}[x{\leftarrow}\mathsf{I}]\underline{y}$ is useful

- Indirect usefulness:
  $(xy)[x{\leftarrow}z][z{\leftarrow}\mathsf{I}] \to (xy)[x{\leftarrow}\mathsf{I}][z{\leftarrow}\mathsf{I}]$ is useful or not?
  $\hookrightarrow$ It is useful!

- Renaming chains:

$$
\begin{array}{ll}
 & (x_0 t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}x_k][x_k{\leftarrow}\mathsf{I}] \\
\to & (x_0 t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}] \\
\to^* & (x_0 t)[x_0{\leftarrow}\mathsf{I}][x_1{\leftarrow}\mathsf{I}]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}]
\end{array}
$$

Usefulness: subtleties

- Contextual closure:
  $x[x{\leftarrow}\mathsf{I}] \to \mathsf{I}[x{\leftarrow}\mathsf{I}]$ is non-useful
  while $x[x{\leftarrow}\mathsf{I}]y \to \underline{\mathsf{I}}[x{\leftarrow}\mathsf{I}]\underline{y}$ is useful

- Indirect usefulness:
  $(xy)[x{\leftarrow}z][z{\leftarrow}\mathsf{I}] \to (xy)[x{\leftarrow}\mathsf{I}][z{\leftarrow}\mathsf{I}] \to (\mathsf{I}y)[x{\leftarrow}\mathsf{I}][z{\leftarrow}\mathsf{I}]$
  $\hookrightarrow$ It is useful!

- Renaming chains:

$$\begin{array}{ll} & (x_0\,t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}x_k][x_k{\leftarrow}\mathsf{I}] \\ \to & (x_0\,t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}] \\ \to^* & (x_0\,t)[x_0{\leftarrow}\mathsf{I}][x_1{\leftarrow}\mathsf{I}]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}] \end{array}$$

Usefulness: subtleties

- Contextual closure:
  $x[x{\leftarrow}\mathsf{I}] \to \mathsf{I}[x{\leftarrow}\mathsf{I}]$ is non-useful
  while $x[x{\leftarrow}\mathsf{I}]y \to \underline{\mathsf{I}}[x{\leftarrow}\mathsf{I}]\underline{y}$ is useful

- Indirect usefulness:
  $(xy)[x{\leftarrow}z][z{\leftarrow}\mathsf{I}] \to (xy)[x{\leftarrow}\mathsf{I}][z{\leftarrow}\mathsf{I}] \to (\mathsf{I}y)[x{\leftarrow}\mathsf{I}][z{\leftarrow}\mathsf{I}]$
  $\hookrightarrow$ It is (indirectly) useful!

- Renaming chains:

$$\begin{array}{ll} & (x_0\,t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}x_k][x_k{\leftarrow}\mathsf{I}] \\ \to & (x_0\,t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}] \\ \to^* & (x_0\,t)[x_0{\leftarrow}\mathsf{I}][x_1{\leftarrow}\mathsf{I}]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}] \end{array}$$

Usefulness: subtleties

- Contextual closure:
  $x[x{\leftarrow}\mathsf{I}] \to \mathsf{I}[x{\leftarrow}\mathsf{I}]$ is non-useful
  while $x[x{\leftarrow}\mathsf{I}]y \to \underline{\mathsf{I}}[x{\leftarrow}\mathsf{I}]\underline{y}$ is useful

- Indirect usefulness:
  $(xy)[x{\leftarrow}z][z{\leftarrow}\mathsf{I}] \to (xy)[x{\leftarrow}\mathsf{I}][z{\leftarrow}\mathsf{I}] \to (\mathsf{I}y)[x{\leftarrow}\mathsf{I}][z{\leftarrow}\mathsf{I}]$
  $\hookrightarrow$ It is (indirectly) useful!

- Renaming chains:

$$\begin{array}{ll} & (x_0\,t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}x_k][x_k{\leftarrow}\mathsf{I}] \\ \to & (x_0\,t)[x_0{\leftarrow}x_1][x_1{\leftarrow}x_2]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}] \\ \to^* & (x_0\,t)[x_0{\leftarrow}\mathsf{I}][x_1{\leftarrow}\mathsf{I}]\cdots[x_{k-1}{\leftarrow}\mathsf{I}][x_k{\leftarrow}\mathsf{I}] \end{array}$$

Positive **Focusing** is Directly Useful

Focusing

Focusing is a technique first introduced by Andreoli to reduce non-determinism in *logic programming* (or *proof search*) in linear logic.

It comes from a simple observation:

Rule	invertible	non-invertible
Phase	negative	positive
Connective	negative	positive

Focusing gives more structure to proofs.
$\hookrightarrow$ focused proofs can be seen as a (light) canonical form of proofs.

Focusing

Focusing is a technique first introduced by Andreoli to reduce non-determinism in *logic programming* (or *proof search*) in linear logic.

It comes from a simple observation:

Rule	invertible	non-invertible
Phase	negative	positive
Connective	negative	positive

Focusing gives more structure to proofs.
↪ focused proofs can be seen as a (light) canonical form of proofs.

Focusing

Focusing is a technique first introduced by Andreoli to reduce non-determinism in *logic programming* (or *proof search*) in linear logic.

It comes from a simple observation:

Rule	invertible	non-invertible
Phase	negative	positive
Connective	negative	positive

Focusing gives more structure to proofs.
$\hookrightarrow$ focused proofs can be seen as a (light) canonical form of proofs.

Negative/positive λ-terms

In a previous work with Dale Miller, we use the focused proof system $LJF_{\supset}$ to design term structures.

Formulas are polarized:

- Implications are negative
- Atomic formulas are either negative or postive

We consider the two uniform polarizations δ^- and δ^+:

- δ^- yields the usual tree-like syntax. No sharing within a term.
  $\hookrightarrow$ negative/usual λ-terms
- δ^+ yields a syntax allowing some specific forms of sharing within a term.
  $\hookrightarrow$ *positive* λ-terms

Negative/positive λ-terms

In a previous work with Dale Miller, we use the focused proof system $LJF_{\supset}$ to design term structures.

Formulas are polarized:

- Implications are negative
- Atomic formulas are either negative or postive

We consider the two uniform polarizations δ^{-} and δ^{+}:

- δ^{-} yields the usual tree-like syntax. No sharing within a term.
  $\hookrightarrow$ negative/usual λ-terms
- δ^{+} yields a syntax allowing some specific forms of sharing within a term.
  $\hookrightarrow$ *positive* λ-terms

Negative/positive λ-terms

In a previous work with Dale Miller, we use the focused proof system $LJF_{\supset}$ to design term structures.

Formulas are polarized:

- Implications are negative
- Atomic formulas are either negative or postive

We consider the two uniform polarizations δ^{-} and δ^{+}:

- δ^{-} yields the usual tree-like syntax. No sharing within a term.
  $\hookrightarrow$ negative/usual λ-terms
- δ^{+} yields a syntax allowing some specific forms of sharing within a term.
  $\hookrightarrow$ *positive* λ-terms

Negative/positive λ-terms

In a previous work with Dale Miller, we use the focused proof system $LJF_{\supset}$ to design term structures.

Formulas are polarized:

- Implications are negative
- Atomic formulas are either negative or postive

We consider the two uniform polarizations δ^- and δ^+:

- δ^- yields the usual tree-like syntax. No sharing within a term.
  $\hookrightarrow$ negative/usual λ-terms
- δ^+ yields a syntax allowing some specific forms of sharing within a term.
  $\hookrightarrow$ *positive* λ-terms

Negative/positive λ-terms

In a previous work with Dale Miller, we use the focused proof system $LJF_{\supset}$ to design term structures.

Formulas are polarized:

- Implications are negative
- Atomic formulas are either negative or postive

We consider the two uniform polarizations δ^- and δ^+:

- δ^- yields the usual tree-like syntax. No sharing within a term.
  $\hookrightarrow$ negative/usual λ-terms
- δ^+ yields a syntax allowing some specific forms of sharing within a term.
  $\hookrightarrow$ *positive* λ-terms

Positive Focusing is Directly Useful

Positive λ-calculus λ_{pos}

$$t, u \quad ::= \quad x \mid t[x{\leftarrow}yz] \mid t[x{\leftarrow}\lambda y.u]$$

- ESs are flattened.
- Restricted form of explicit substitutions:
  1. Minimalistic application yz
  2. No ES for variables: variables are not values and renaming chains do not exist!

Example of reduction:

$$\begin{array}{rl} & x[x{\leftarrow}yy][y{\leftarrow}zz'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ \rightarrow_{\text{oe}_+} & x[x{\leftarrow}yy][y{\leftarrow}(\lambda w.w'[w'{\leftarrow}ww])z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ & x[x{\leftarrow}w'_1 w'_1][w'_1{\leftarrow}z'z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \end{array}$$

Key fact: λ_{pos} (or λ_{xpos}) is directly useful by definition!

Positive λ-calculus λ_{pos}

$$t, u \quad ::= \quad x \mid t[x {\leftarrow} yz] \mid t[x {\leftarrow} \lambda y.u]$$

- ESs are flattened.
- Restricted form of explicit substitutions:
  1. Minimalistic application yz
  2. No ES for variables: variables are not values and renaming chains do not exist!

Example of reduction:

$$\begin{array}{rl} & x[x{\leftarrow}yy][y{\leftarrow}zz'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ \to_{\text{oe}_+} & x[x{\leftarrow}yy][y{\leftarrow}(\lambda w.w'[w'{\leftarrow}ww])z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ & x[x{\leftarrow}w_1'w_1'][w_1'{\leftarrow}z'z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \end{array}$$

Key fact: λ_{pos} (or λ_{xpos}) is directly useful by definition!

Positive λ-calculus λ_{pos}

$$t, u \quad ::= \quad x \mid t[x{\leftarrow}yz] \mid t[x{\leftarrow}\lambda y.u]$$

- ESs are flattened.
- Restricted form of explicit substitutions:
  1. Minimalistic application yz
  2. No ES for variables: variables are not values and renaming chains do not exist!

Example of reduction:

$$\begin{array}{rl} & x[x{\leftarrow}yy][y{\leftarrow}zz'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ \to_{\mathrm{oe}_+} & x[x{\leftarrow}yy][y{\leftarrow}(\lambda w.w'[w'{\leftarrow}ww])z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ & x[x{\leftarrow}w_1'w_1'][w_1'{\leftarrow}z'z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \end{array}$$

Key fact: λ_{pos} (or λ_{xpos}) is directly useful by definition!

Positive λ-calculus λ_{pos}

$$t, u \quad ::= \quad x \mid t[x \leftarrow yz] \mid t[x \leftarrow \lambda y.u]$$

- ESs are flattened.
- Restricted form of explicit substitutions:
  1. Minimalistic application yz
  2. No ES for variables: variables are not values and renaming chains do not exist!

Example of reduction:

$$\begin{array}{rl} & x[x \leftarrow yy][y \leftarrow zz'][z \leftarrow \lambda w.w'[w' \leftarrow ww]] \\ \rightarrow_{\text{oe}_+} & x[x \leftarrow yy][y \leftarrow (\lambda w.w'[w' \leftarrow ww])z'][z \leftarrow \lambda w.w'[w' \leftarrow ww]] \\ \rightarrow_{\text{om}_+} & x[x \leftarrow w_1' w_1'][w_1' \leftarrow z'z'][z \leftarrow \lambda w.w'[w' \leftarrow ww]] \end{array}$$

Key fact: λ_{pos} (or λ_{xpos}) is directly useful by definition!

Positive λ-calculus λ_{pos}

$$t, u \quad ::= \quad x \mid t[x{\leftarrow}yz] \mid t[x{\leftarrow}\lambda y.u]$$

- ESs are flattened.
- Restricted form of explicit substitutions:
  1. Minimalistic application yz
  2. No ES for variables: variables are not values and renaming chains do not exist!

Example of reduction:

$$\begin{array}{ll} & x[x{\leftarrow}yy][y{\leftarrow}\underline{z}z'][\underline{z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]}] \\ \to_{\text{oe}_+} & x[x{\leftarrow}yy][y{\leftarrow}(\lambda w.w'[w'{\leftarrow}ww])z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ \to_{\text{om}_+} & x[x{\leftarrow}w_1'w_1'][w_1'{\leftarrow}z'z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \end{array}$$

Key fact: λ_{pos} (or λ_{xpos}) is directly useful by definition!

Positive λ-calculus λ_{pos}

$$t, u \quad ::= \quad x \mid t[x{\leftarrow}yz] \mid t[x{\leftarrow}\lambda y.u]$$

- ESs are flattened.
- Restricted form of explicit substitutions:
  1. Minimalistic application yz
  2. No ES for variables: variables are not values and renaming chains do not exist!

Example of reduction:

$$\begin{array}{ll} & x[x{\leftarrow}yy][y{\leftarrow}zz'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ \to_{\mathsf{oe}_+} & x[x{\leftarrow}yy][y{\leftarrow}(\lambda w.w'[w'{\leftarrow}ww])z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ \to_{\mathsf{om}_+} & x[x{\leftarrow}w_1'w_1'][w_1'{\leftarrow}z'z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \end{array}$$

Key fact: λ_{pos} (or λ_{xpos}) is directly useful by definition!

Positive λ-calculus λ_{pos}

$$t, u \quad ::= \quad x \mid t[x \leftarrow yz] \mid t[x \leftarrow \lambda y.u]$$

- ESs are flattened.
- Restricted form of explicit substitutions:
  1. Minimalistic application yz
  2. No ES for variables: variables are not values and renaming chains do not exist!

Example of reduction:

$$\begin{array}{ll} & x[x \leftarrow yy][y \leftarrow zz'][z \leftarrow \lambda w.w'[w' \leftarrow ww]] \\ \rightarrow_{\mathsf{oe}_+} & x[x \leftarrow yy][y \leftarrow (\lambda w.w'[w' \leftarrow ww])z'][z \leftarrow \lambda w.w'[w' \leftarrow ww]] \\ \rightarrow_{\mathsf{om}_+} & x[x \leftarrow w_1' w_1'][w_1' \leftarrow z'z'][z \leftarrow \lambda w.w'[w' \leftarrow ww]] \end{array}$$

Key fact: λ_{pos} (or λ_{xpos}) is directly useful by definition!

Positive λ-calculus λ_{pos}

$$t, u \quad ::= \quad x \mid t[x{\leftarrow}yz] \mid t[x{\leftarrow}\lambda y.u]$$

- ESs are flattened.
- Restricted form of explicit substitutions:
  1. Minimalistic application yz
  2. No ES for variables: variables are not values and renaming chains do not exist!

Example of reduction:

$$\begin{array}{ll} & x[x{\leftarrow}yy][y{\leftarrow}zz'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ \to_{\mathrm{oe}_+} & x[x{\leftarrow}yy][y{\leftarrow}(\lambda w.w'[w'{\leftarrow}ww])z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ \to_{\mathrm{oem}_+} & x[x{\leftarrow}w'_1w'_1][w'_1{\leftarrow}z'z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \end{array}$$

Key fact: λ_{pos} (or λ_{xpos}) is directly useful by definition!

Explicit positive λ-calculus λ_{xpos}

$$t, u \quad ::= \quad x \mid t[x{\leftarrow}yz] \mid t[x{\leftarrow}\lambda y.u] \mid \color{red}{t[x{\leftarrow}(\lambda y.u)z]}$$

- ESs are flattened.
- Restricted form of explicit substitutions:
  1. Minimalistic application yz
  2. No ES for variables: variables are not values and renaming chains do not exist!

Example of reduction:

$$\begin{array}{ll} & x[x{\leftarrow}yy][y{\leftarrow}zz'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ \to_{\mathrm{oe}_+} & x[x{\leftarrow}yy][y{\leftarrow}(\lambda w.w'[w'{\leftarrow}ww])z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \\ \to_{\mathrm{om}_+} & x[x{\leftarrow}w_1'w_1'][w_1'{\leftarrow}z'z'][z{\leftarrow}\lambda w.w'[w'{\leftarrow}ww]] \end{array}$$

Key fact: λ_{pos} (or λ_{xpos}) is directly useful by definition!

Explicit positive λ-calculus λ_{xpos}

$$t, u \quad ::= \quad x \mid t[x \leftarrow yz] \mid t[x \leftarrow \lambda y.u] \mid t[x \leftarrow (\lambda y.u)z]$$

- ESs are flattened.
- Restricted form of explicit substitutions:
  1. Minimalistic application yz
  2. No ES for variables: variables are not values and renaming chains do not exist!

Example of reduction:

$$\begin{array}{ll} & x[x \leftarrow yy][y \leftarrow zz'][z \leftarrow \lambda w.w'[w' \leftarrow ww]] \\ \to_{\mathsf{oe}_+} & x[x \leftarrow yy][y \leftarrow (\lambda w.w'[w' \leftarrow ww])z'][z \leftarrow \lambda w.w'[w' \leftarrow ww]] \\ \to_{\mathsf{om}_+} & x[x \leftarrow w_1' w_1'][w_1' \leftarrow z'z'][z \leftarrow \lambda w.w'[w' \leftarrow ww]] \end{array}$$

Key fact: λ_{pos} (or λ_{xpos}) is directly useful by definition!

Value substitution calculus $\lambda_{\mathtt{vsc}}$

$$\begin{array}{rcl} t, u & ::= & v \mid tu \mid t[x{\leftarrow}u] \\ v & ::= & x \mid \lambda x.t \end{array}$$

- General applications tu
- Variables are values and ES for variable: renaming chains do exist...

There are two rules in $\lambda_{\mathtt{vsc}}$:

- The m-rule fires a β-redex and creates an ES

$$(\lambda x.t)u \to t[x{\leftarrow}u]$$

- The e-rule fires an ES (of values) and makes a substitution

$$C\langle\langle x\rangle\rangle[x{\leftarrow}v] \to C\langle\langle v\rangle\rangle[x{\leftarrow}v]$$

Known result: the number of m-steps is a reasonable cost model.

Value substitution calculus $\lambda_{\mathtt{vsc}}$

$$
\begin{array}{rcl}
t, u & ::= & v \mid tu \mid t[x{\leftarrow}u] \\
v & ::= & x \mid \lambda x.t
\end{array}
$$

- General applications tu
- Variables are values and ES for variable: renaming chains do exist...

There are two rules in $\lambda_{\mathtt{vsc}}$:

- The m-rule fires a β-redex and creates an ES

$$(\lambda x.t)u \to t[x{\leftarrow}u]$$

- The e-rule fires an ES (of values) and makes a substitution

$$C\langle x\rangle[x{\leftarrow}v] \to C\langle v\rangle[x{\leftarrow}v]$$

Known result: the number of m-steps is a reasonable cost model.

Value substitution calculus $\lambda_{\mathtt{vsc}}$

$$
\begin{array}{rcl}
t, u & ::= & v \mid tu \mid t[x{\leftarrow}u] \\
v & ::= & x \mid \lambda x.t
\end{array}
$$

- General applications tu
- Variables are values and ES for variable: renaming chains do exist...

There are two rules in $\lambda_{\mathtt{vsc}}$:

- The m-rule fires a β-redex and creates an ES

$$(\lambda x.t)u \to t[x{\leftarrow}u]$$

- The e-rule fires an ES (of values) and makes a substitution

Known result: the number of m-steps is a reasonable cost model.

Value substitution calculus $\lambda_{\mathtt{vsc}}$

$$\begin{array}{rcl} t, u & ::= & v \mid tu \mid t[x{\leftarrow}u] \\ v & ::= & x \mid \lambda x.t \end{array}$$

- General applications tu
- Variables are values and ES for variable: renaming chains do exist...

There are two rules in $\lambda_{\mathtt{vsc}}$:

- The m-rule fires a β-redex and creates an ES

$$(\lambda x.t)u \to t[x{\leftarrow}u]$$

- The e-rule fires an ES (of values) and makes a substitution

$$C\langle x\rangle[x{\leftarrow}v] \to C\langle v\rangle[x{\leftarrow}v]$$

Known result: the number of m-steps is a reasonable cost model.

Value substitution calculus $\lambda_{\mathtt{vsc}}$

$$\begin{array}{rcl} t, u & ::= & v \mid tu \mid t[x{\leftarrow}u] \\ v & ::= & x \mid \lambda x.t \end{array}$$

- General applications tu
- Variables are values and ES for variable: renaming chains do exist...

There are two rules in $\lambda_{\mathtt{vsc}}$:

- The m-rule fires a β-redex and creates an ES

$$(\lambda x.t)u \to t[x{\leftarrow}u]$$

- The e-rule fires an ES (of values) and makes a substitution

$$C\langle x\rangle[x{\leftarrow}v] \to C\langle v\rangle[x{\leftarrow}v]$$

Known result: the number of m-steps is a reasonable cost model.

Value substitution calculus $\lambda_{\mathtt{vsc}}$

$$\begin{array}{rcl} t, u & ::= & v \mid tu \mid t[x \leftarrow u] \\ v & ::= & x \mid \lambda x.t \end{array}$$

- General applications tu
- Variables are values and ES for variable: renaming chains do exist...

There are two rules in $\lambda_{\mathtt{vsc}}$:

- The m-rule fires a β-redex and creates an ES

$$(\lambda x.t)u \to t[x \leftarrow u]$$

- The e-rule fires an ES (of values) and makes a substitution

$$C\langle x\rangle[x \leftarrow v] \to C\langle v\rangle[x \leftarrow v]$$

Known result: the number of m-steps is a reasonable cost model.

Value substitution calculus $\lambda_{\mathtt{vsc}}$

$$\begin{array}{rcl} t,u & ::= & v \mid tu \mid t[x \leftarrow u] \\ v & ::= & x \mid \lambda x.t \end{array}$$

- General applications tu
- Variables are values and ES for variable: renaming chains do exist...

There are two rules in $\lambda_{\mathtt{vsc}}$:

- The m-rule fires a β-redex and creates an ES

$$(\lambda x.t)u \to t[x \leftarrow u]$$

- The e-rule fires an ES (of values) and makes a substitution

$$C\langle x \rangle [x \leftarrow v] \to C\langle v \rangle [x \leftarrow v]$$

Known result: the number of m-steps is a reasonable cost model.

Value substitution calculus $\lambda_{\mathtt{vsc}}$

$$\begin{array}{rcl} t, u & ::= & v \mid tu \mid t[x{\leftarrow}u] \\ v & ::= & x \mid \lambda x.t \end{array}$$

- General applications tu
- Variables are values and ES for variable: renaming chains do exist...

There are two rules in $\lambda_{\mathtt{vsc}}$:

- The m-rule fires a β-redex and creates an ES

$$(\lambda x.t)u \to t[x{\leftarrow}u]$$

- The e-rule fires an ES (of values) and makes a substitution

$$C\langle x\rangle[x{\leftarrow}v] \to C\langle v\rangle[x{\leftarrow}v]$$

Known result: the number of m-steps is a reasonable cost model.

Positive Focusing is Directly Useful

Dissecting $\lambda_{\mathtt{vsc}}$

λ_{xpos} is directly useful while $\lambda_{\mathtt{vsc}}$ is not.

In order to relate $\lambda_{\mathtt{vsc}}$ to λ_{xpos}, we define a core calculus of $\lambda_{\mathtt{vsc}}$ which is essentially equivalent to $\lambda_{\mathtt{vsc}}$ and captures direct usefulness.

Step 1: Separate e-rules for variables ($\rightarrow_{\mathsf{e_{var}}}$) and abstractions ($\rightarrow_{\mathsf{e_{abs}}}$).

Step 2: Distinguish (directly) useful e-steps ($\rightarrow_{\mathsf{e_u}}$) from non useful e-steps ($\rightarrow_{\mathsf{e_{nu}}}$) for abstractions.

Core reduction $= \rightarrow_{\mathsf{m}} + \rightarrow_{\mathsf{e_{var}}} + \rightarrow_{\mathsf{e_u}}$

Non-useful reduction $= \rightarrow_{\mathsf{e_{nu}}}$

Dissecting $\lambda_{\mathtt{vsc}}$

λ_{xpos} is directly useful while $\lambda_{\mathtt{vsc}}$ is not.

In order to relate $\lambda_{\mathtt{vsc}}$ to λ_{xpos}, we define a core calculus of $\lambda_{\mathtt{vsc}}$ which is essentially equivalent to $\lambda_{\mathtt{vsc}}$ and captures direct usefulness.

Step 1: Separate e-rules for variables ($\rightarrow_{\mathsf{e_{var}}}$) and abstractions ($\rightarrow_{\mathsf{e_{abs}}}$).

Step 2: Distinguish (directly) useful e-steps ($\rightarrow_{\mathsf{e_u}}$) from non useful e-steps ($\rightarrow_{\mathsf{e_{nu}}}$) for abstractions.

Core reduction $= \rightarrow_{\mathsf{m}} + \rightarrow_{\mathsf{e_{var}}} + \rightarrow_{\mathsf{e_u}}$

Non-useful reduction $= \rightarrow_{\mathsf{e_{nu}}}$

Dissecting $\lambda_{\mathtt{vsc}}$

λ_{xpos} is directly useful while $\lambda_{\mathtt{vsc}}$ is not.

In order to relate $\lambda_{\mathtt{vsc}}$ to λ_{xpos}, we define a core calculus of $\lambda_{\mathtt{vsc}}$ which is essentially equivalent to $\lambda_{\mathtt{vsc}}$ and captures direct usefulness.

Step 1: Separate e-rules for variables ($\rightarrow_{\mathsf{e}_{\mathsf{var}}}$) and abstractions ($\rightarrow_{\mathsf{e}_{\mathsf{abs}}}$).

Step 2: Distinguish (directly) useful e-steps ($\rightarrow_{\mathsf{e}_{\mathsf{u}}}$) from non useful e-steps ($\rightarrow_{\mathsf{e}_{\mathsf{nu}}}$) for abstractions.

Core reduction $= \rightarrow_{\mathsf{m}} + \rightarrow_{\mathsf{e}_{\mathsf{var}}} + \rightarrow_{\mathsf{e}_{\mathsf{u}}}$

Non-useful reduction $= \rightarrow_{\mathsf{e}_{\mathsf{nu}}}$

Dissecting $\lambda_{\mathtt{vsc}}$

λ_{xpos} is directly useful while $\lambda_{\mathtt{vsc}}$ is not.

In order to relate $\lambda_{\mathtt{vsc}}$ to λ_{xpos}, we define a core calculus of $\lambda_{\mathtt{vsc}}$ which is essentially equivalent to $\lambda_{\mathtt{vsc}}$ and captures direct usefulness.

Step 1: Separate *e*-rules for variables ($\rightarrow_{\mathsf{e}_{\mathsf{var}}}$) and abstractions ($\rightarrow_{\mathsf{e}_{\mathsf{abs}}}$).

Step 2: Distinguish (directly) useful *e*-steps ($\rightarrow_{\mathsf{e}_{\mathsf{u}}}$) from non useful *e*-steps ($\rightarrow_{\mathsf{e}_{\mathsf{nu}}}$) for abstractions.

Core reduction $= \rightarrow_{\mathsf{m}} + \rightarrow_{\mathsf{e}_{\mathsf{var}}} + \rightarrow_{\mathsf{e}_{\mathsf{u}}}$

Non-useful reduction $= \rightarrow_{\mathsf{e}_{\mathsf{nu}}}$

Dissecting $\lambda_{\mathtt{vsc}}$

λ_{xpos} is directly useful while $\lambda_{\mathtt{vsc}}$ is not.

In order to relate $\lambda_{\mathtt{vsc}}$ to λ_{xpos}, we define a core calculus of $\lambda_{\mathtt{vsc}}$ which is essentially equivalent to $\lambda_{\mathtt{vsc}}$ and captures direct usefulness.

Step 1: Separate e-rules for variables ($\rightarrow_{\mathsf{e_{var}}}$) and abstractions ($\rightarrow_{\mathsf{e_{abs}}}$).

Step 2: Distinguish (directly) useful e-steps ($\rightarrow_{\mathsf{e_u}}$) from non useful e-steps ($\rightarrow_{\mathsf{e_{nu}}}$) for abstractions.

Core reduction $= \rightarrow_{\mathsf{m}} + \rightarrow_{\mathsf{e_{var}}} + \rightarrow_{\mathsf{e_u}}$

Non-useful reduction $= \rightarrow_{\mathsf{e_{nu}}}$

Dissecting $\lambda_{\mathtt{vsc}}$

λ_{xpos} is directly useful while $\lambda_{\mathtt{vsc}}$ is not.

In order to relate $\lambda_{\mathtt{vsc}}$ to λ_{xpos}, we define a core calculus of $\lambda_{\mathtt{vsc}}$ which is essentially equivalent to $\lambda_{\mathtt{vsc}}$ and captures direct usefulness.

Step 1: Separate *e*-rules for variables ($\rightarrow_{\mathsf{e_{var}}}$) and abstractions ($\rightarrow_{\mathsf{e_{abs}}}$).

Step 2: Distinguish (directly) useful *e*-steps ($\rightarrow_{\mathsf{e_u}}$) from non useful *e*-steps ($\rightarrow_{\mathsf{e_{nu}}}$) for abstractions.

Core reduction $= \rightarrow_{\mathsf{m}} + \rightarrow_{\mathsf{e_{var}}} + \rightarrow_{\mathsf{e_u}}$

Non-useful reduction $= \rightarrow_{\mathsf{e_{nu}}}$

Positive Focusing is Directly Useful

Big picture

$$\lambda_{\mathtt{ovsc}} \quad (= \text{Core } \lambda_{\mathtt{ovsc}} + \text{Non-useful})$$

Big picture

$$\lambda_{\texttt{ovsc}} \quad (= \text{Core } \lambda_{\texttt{ovsc}} + \text{Non-useful})$$

$$t \xrightarrow{\hspace{10em}}^{*} u$$

Big picture

$$\lambda_{\texttt{ovsc}} \quad (= \text{Core } \lambda_{\texttt{ovsc}} + \text{Non-useful})$$

t t' u

Big picture

$$\lambda_{\texttt{ovsc}} \quad (= \text{Core } \lambda_{\texttt{ovsc}} + \text{Non-useful})$$

$$t \xrightarrow{\text{Core } \lambda_{\texttt{ovsc}}}{}^{*} t' \xrightarrow{\text{Non-useful}}{}^{*} u$$

Big picture

Termination $\qquad \lambda_{\mathtt{ovsc}} \quad (= \text{Core } \lambda_{\mathtt{ovsc}} + \text{Non-useful})$

Equivalence $\cdots\rightarrow \equiv$

$$t \xrightarrow{\text{Core } \lambda_{\mathtt{ovsc}}}{}^{*} t' \xrightarrow{\text{Non-useful}}{}^{*} u$$

Big picture

Termination
Equivalence

$\lambda_{\mathtt{ovsc}}$ (= Core $\lambda_{\mathtt{ovsc}}$ + Non-useful)

$$\begin{tikzcd}
t \arrow[rr, "\text{Core } \lambda_{\mathtt{ovsc}}"] \arrow[d, dotted] \arrow[drr, phantom, "\text{preserves the number of } m\text{-steps}"] & & {}^{*}\, t' \arrow[r, "\text{Non-useful}"] \arrow[d, dotted] & {}^{*}\, u \\
\llbracket t \rrbracket \arrow[rr, "\lambda_{\mathsf{oxpos}}"'] & & {}^{*}\, \llbracket t' \rrbracket &
\end{tikzcd}$$

Big picture

Termination $\qquad \lambda_{\texttt{ovsc}} \quad (=$ Core $\lambda_{\texttt{ovsc}}$ + Non-useful$)$
Equivalence $\dashrightarrow$ $\equiv$

$$
\begin{tikzcd}
t \arrow[r, "\text{Core } \lambda_{\texttt{ovsc}}"] \arrow[d, dotted] \arrow[dr, phantom, "|||"] & {}^{*}\, t' \arrow[r, "\text{Non-useful}"] \arrow[d, dotted] & {}^{*}\, u \\
\llbracket t \rrbracket \arrow[r, "\lambda_{\mathsf{oxpos}}"'] & {}^{*}\, \llbracket t' \rrbracket &
\end{tikzcd}
$$

Conclusion and Future work

- We show that the compactness of λ_{pos} allows one to capture the essence of usefulness. What is remarkable is that λ_{pos} is an outcome of a study of term representation inspired by focusing.
- Future work:
  1. efficient implementation of meta-level renamings involved in λ_{pos}. We expect this to be doable in an efficient way via an appropriate abstract machine.
  2. λ_{pos} for call-by-need evaluation

Conclusion and Future work

- We show that the compactness of $\lambda_{\mathtt{pos}}$ allows one to capture the essence of usefulness. What is remarkable is that $\lambda_{\mathtt{pos}}$ is an outcome of a study of term representation inspired by focusing.
- Future work:
  1. efficient implementation of meta-level renamings involved in $\lambda_{\mathtt{pos}}$. We expect this to be doable in an efficient way via an appropriate abstract machine.
  2. $\lambda_{\mathtt{pos}}$ for call-by-need evaluation

Conclusion and Future work

- We show that the compactness of λ_{pos} allows one to capture the essence of usefulness. What is remarkable is that λ_{pos} is an outcome of a study of term representation inspired by focusing.
- Future work:
  1. efficient implementation of meta-level renamings involved in λ_{pos}. We expect this to be doable in an efficient way via an appropriate abstract machine.
  2. λ_{pos} for call-by-need evaluation

Thank you for your attention!