

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.

→ 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.

→ 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.

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

In a call-by-value setting, general applications tu become somewhat redundant.

→ 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.

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

In a call-by-value setting, general applications tu become somewhat redundant.

→ 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.

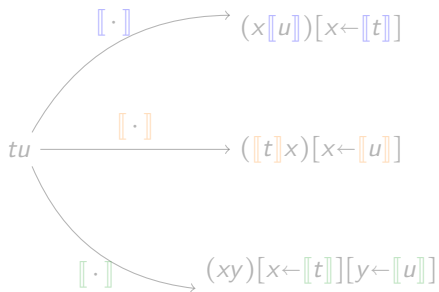
$$t, u ::= x \mid tu \mid \lambda x. t \mid t[x \leftarrow u] \text{ (explicit substitution)}$$

In a call-by-value setting, general applications tu become somewhat redundant.

↔ It is possible to restrict the shape of applications.

Shape of applications in a CbV setting

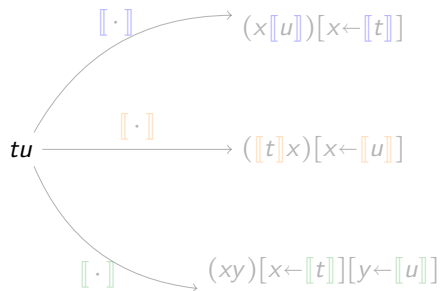
In CbV, there are many possible ways to restrict the shape of applications:



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

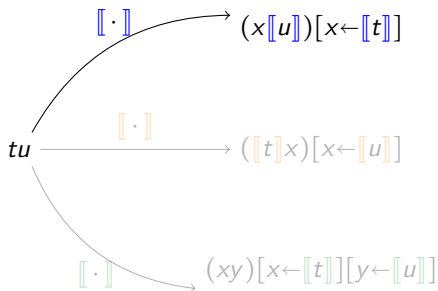
In CbV, there are many possible ways to restrict the shape of applications:



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

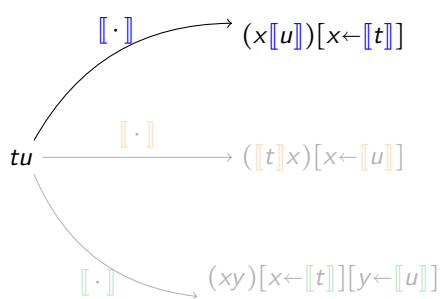
In CbV, there are many possible ways to restrict the shape of applications:



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:

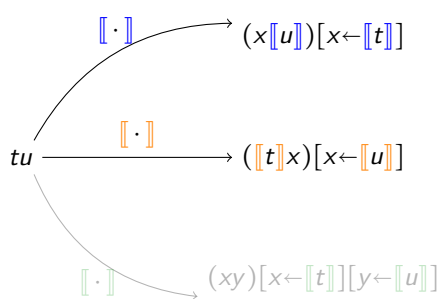


value as the left subterm
of an application

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:

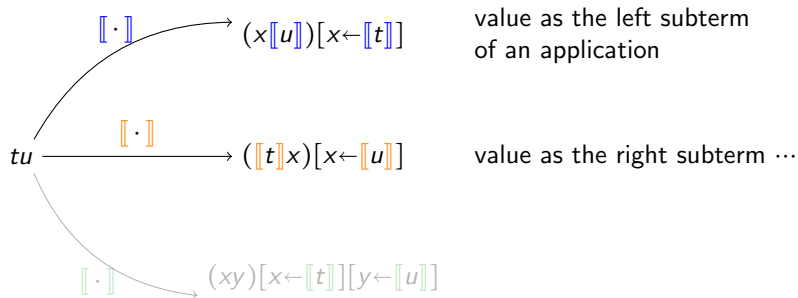


value as the left subterm
of an application

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

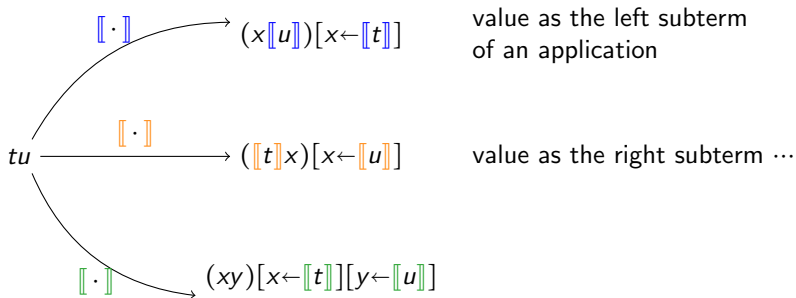
In CbV, there are many possible ways to restrict the shape of applications:



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

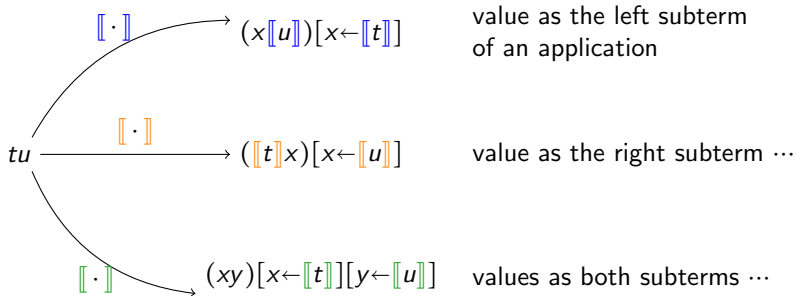
In CbV, there are many possible ways to restrict the shape of applications:



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

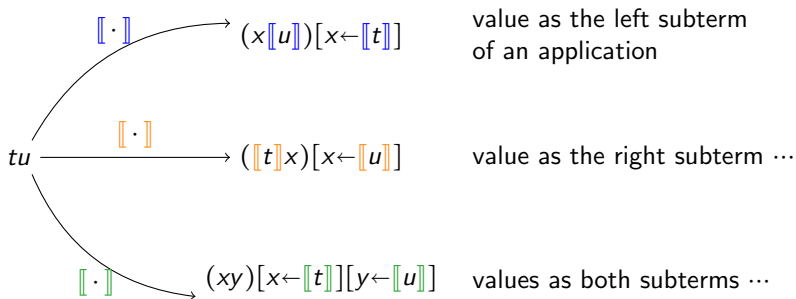
In CbV, there are many possible ways to restrict the shape of applications:



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

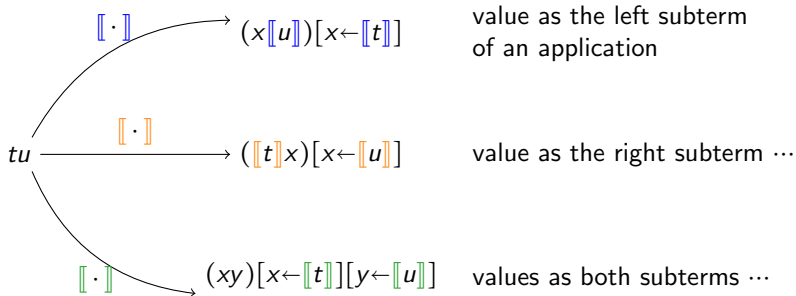
In CbV, there are many possible ways to restrict the shape of applications:



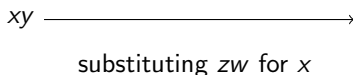
These restrictions are typical in a call-by-value setting, as substitutions of applications sometimes are simply **blocked** by the syntax:

Shape of applications in a CbV setting

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

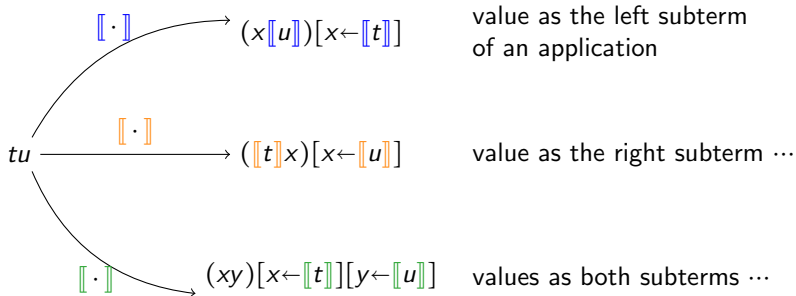


These restrictions are typical in a call-by-value setting, as substitutions of applications sometimes are simply **blocked** by the syntax:



Shape of applications in a CbV setting

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



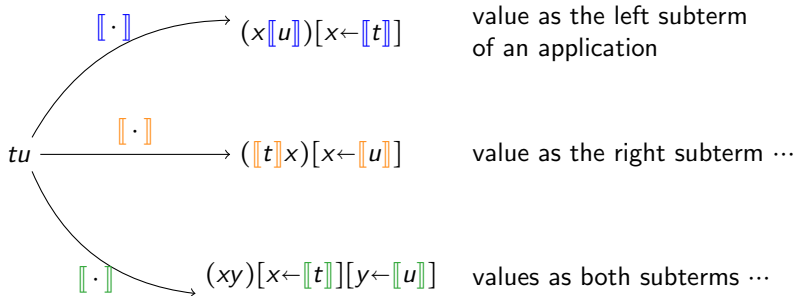
These restrictions are typical in a call-by-value setting, as substitutions of applications sometimes are simply **blocked** by the syntax:

$$xy \longrightarrow (zw)y$$

substituting zw for x

Shape of applications in a CbV setting

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



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

$$xy \xrightarrow{\text{X}} (zw)y$$

substituting zw for x

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:

$$(xx)[x \leftarrow I] \rightarrow II$$

vs.

$$(xx)[x \leftarrow I] \rightarrow (Ix)[x \leftarrow I] \rightarrow (II)[x \leftarrow I] \rightarrow II$$

- 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:

$$(xx)[x \leftarrow I] \rightarrow I$$

vs.

$$(xx)[x \leftarrow I] \rightarrow (Ix)[x \leftarrow I] \rightarrow (I) [x \leftarrow I] \rightarrow I$$

• 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:

$$(xx)[x \leftarrow I] \rightarrow II$$

vs.

$$(xx)[x \leftarrow I] \rightarrow (Ix)[x \leftarrow I] \rightarrow (II)[x \leftarrow I] \rightarrow II$$

- 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:

$$(xx)[x \leftarrow I] \rightarrow II$$

vs.

$$(xx)[x \leftarrow I] \rightarrow (Ix)[x \leftarrow I] \rightarrow (II)[x \leftarrow I] \rightarrow II$$

- 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:

$$(xx)[x \leftarrow I] \rightarrow II$$

vs.

$$(xx)[x \leftarrow I] \rightarrow (Ix)[x \leftarrow I] \rightarrow (II)[x \leftarrow I] \rightarrow II$$

- 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:

$$(xx)[x \leftarrow I] \rightarrow II$$

vs.

$$(xx)[x \leftarrow I] \rightarrow (Ix)[x \leftarrow I] \rightarrow (II)[x \leftarrow I] \rightarrow II$$

- 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:

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

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

Consider

$$(yx)[x \leftarrow I] \rightarrow (yI)[x \leftarrow I]$$

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

Some more examples:

- $(xy)[x \leftarrow I] \rightarrow (Iy)[x \leftarrow I]$ is **useful**
- $x[x \leftarrow I] \rightarrow I[x \leftarrow I]$ is **non-useful**
- $(xx)[x \leftarrow I]$

Useful substitutions v.s. non-useful substitutions

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

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

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

Consider

$$(yx)[x \leftarrow l] \rightarrow (yl)[x \leftarrow l]$$

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

Some more examples:

- $(xy)[x \leftarrow l] \rightarrow (ly)[x \leftarrow l]$ is **useful**
- $x[x \leftarrow l] \rightarrow l[x \leftarrow l]$ is **non-useful**
- $(xx)[x \leftarrow l]$

Useful substitutions v.s. non-useful substitutions

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

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

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

Consider

$$(yx)[x \leftarrow I] \rightarrow (yI)[x \leftarrow I]$$

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

Some more examples:

- $(xy)[x \leftarrow I] \rightarrow (Iy)[x \leftarrow I]$ is **useful**
- $x[x \leftarrow I] \rightarrow I[x \leftarrow I]$ is **non-useful**
- $(xx)[x \leftarrow I]$

Useful substitutions v.s. non-useful substitutions

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

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

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

Consider

$$(yx)[x \leftarrow l] \rightarrow (yl)[x \leftarrow l]$$

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

Some more examples:

- $(xy)[x \leftarrow l] \rightarrow (ly)[x \leftarrow l]$ is **useful**
- $x[x \leftarrow l] \rightarrow l[x \leftarrow l]$ is **non-useful**
- $(xx)[x \leftarrow l]$

Useful substitutions v.s. non-useful substitutions

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

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

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

Consider

$$(yx)[x \leftarrow I] \rightarrow (yI)[x \leftarrow I]$$

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

Some more examples:

- $(xy)[x \leftarrow I] \rightarrow (Iy)[x \leftarrow I]$ is **useful**
- $x[x \leftarrow I] \rightarrow I[x \leftarrow I]$ is **non-useful**
- $(xx)[x \leftarrow I]$

Useful substitutions v.s. non-useful substitutions

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

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

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

Consider

$$(yx)[x \leftarrow I] \rightarrow (yI)[x \leftarrow I]$$

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

Some more examples:

- $(xy)[x \leftarrow I] \rightarrow (Iy)[x \leftarrow I]$ is **(directly) useful**
- $x[x \leftarrow I] \rightarrow I[x \leftarrow I]$ is **non-useful**
- $(xx)[x \leftarrow I]$

Useful substitutions v.s. non-useful substitutions

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

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

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

Consider

$$(yx)[x \leftarrow I] \rightarrow (yI)[x \leftarrow I]$$

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

Some more examples:

- $(xy)[x \leftarrow I] \rightarrow (Iy)[x \leftarrow I]$ is **(directly) useful**
- $x[x \leftarrow I] \rightarrow I[x \leftarrow I]$ is **non-useful**
- $(xx)[x \leftarrow I]$

Usefulness: subtleties

- Contextual closure:

$x[x \leftarrow l] \rightarrow l[x \leftarrow l]$ is non-useful
while $x[x \leftarrow l]y \rightarrow l[x \leftarrow l]y$ is useful

- Indirect usefulness:

$(xy)[x \leftarrow z][z \leftarrow l] \rightarrow (xy)[x \leftarrow l][z \leftarrow l]$
 \leftrightarrow It is useful!

- Renaming chains:

$$\begin{aligned} & (x_0 t)[x_0 \leftarrow x_1][x_1 \leftarrow x_2] \cdots [x_{k-1} \leftarrow x_k][x_k \leftarrow l] \\ \rightarrow & (x_0 t)[x_0 \leftarrow x_1][x_1 \leftarrow x_2] \cdots [x_{k-1} \leftarrow l][x_k \leftarrow l] \\ \rightarrow^* & (x_0 t)[x_0 \leftarrow l][x_1 \leftarrow l] \cdots [x_{k-1} \leftarrow l][x_k \leftarrow l] \end{aligned}$$

Usefulness: subtleties

- Contextual closure:

$x[x \leftarrow I] \rightarrow I[x \leftarrow I]$ is non-useful
while $x[x \leftarrow I]y \rightarrow I[x \leftarrow I]y$ is useful

- Indirect usefulness:

$(xy)[x \leftarrow z][z \leftarrow I] \rightarrow (xy)[x \leftarrow I][z \leftarrow I]$
 \leftrightarrow It is useful!

- Renaming chains:

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

Usefulness: subtleties

- Contextual closure:

$x[x \leftarrow I] \rightarrow I[x \leftarrow I]$ is non-useful

while $x[x \leftarrow I]y \rightarrow I[x \leftarrow I]y$ is useful

- Indirect usefulness:

$(xy)[x \leftarrow z][z \leftarrow I] \rightarrow (xy)[x \leftarrow I][z \leftarrow I]$

\leftrightarrow It is useful!

- Renaming chains:

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

Usefulness: subtleties

- Contextual closure:
 $x[x \leftarrow I] \rightarrow I[x \leftarrow I]$ is non-useful
while $x[x \leftarrow I]y \rightarrow I[x \leftarrow I]y$ is useful

- Indirect usefulness:
 $(xy)[x \leftarrow z][z \leftarrow I] \rightarrow (xy)[x \leftarrow I][z \leftarrow I]$
 \rightarrow It is useful!

- Renaming chains:

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

Usefulness: subtleties

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

- Indirect usefulness:
 $(xy)[x \leftarrow z][z \leftarrow I] \rightarrow (xy)[x \leftarrow I][z \leftarrow I]$
 \leftrightarrow It is useful!

- Renaming chains:

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

Usefulness: subtleties

- Contextual closure:
 $x[x \leftarrow I] \rightarrow I[x \leftarrow I]$ is non-useful
while $x[x \leftarrow I]y \rightarrow \underline{I[x \leftarrow I]}y$ is useful
- Indirect usefulness:
 $(xy)[x \leftarrow z][z \leftarrow I] \rightarrow (xy)[x \leftarrow I][z \leftarrow I]$ is useful or not?
 \hookrightarrow It is useful!
- Renaming chains:

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

Usefulness: subtleties

- Contextual closure:

$x[x \leftarrow I] \rightarrow I[x \leftarrow I]$ is non-useful

while $x[x \leftarrow I]y \rightarrow \underline{I[x \leftarrow I]}y$ is useful

- Indirect usefulness:

$(xy)[x \leftarrow z][z \leftarrow I] \rightarrow (xy)[x \leftarrow I][z \leftarrow I] \rightarrow (Iy)[x \leftarrow I][z \leftarrow I]$

\leftrightarrow It is useful!

- Renaming chains:

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

Usefulness: subtleties

- Contextual closure:

$x[x \leftarrow I] \rightarrow I[x \leftarrow I]$ is non-useful

while $x[x \leftarrow I]y \rightarrow \underline{I[x \leftarrow I]}y$ is useful

- Indirect usefulness:

$(xy)[x \leftarrow z][z \leftarrow I] \rightarrow (xy)[x \leftarrow I][z \leftarrow I] \rightarrow (Iy)[x \leftarrow I][z \leftarrow I]$

\leftrightarrow It is (indirectly) useful!

- Renaming chains:

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

Usefulness: subtleties

- Contextual closure:

$x[x \leftarrow I] \rightarrow I[x \leftarrow I]$ is non-useful

while $x[x \leftarrow I]y \rightarrow \underline{I[x \leftarrow I]}y$ is useful

- Indirect usefulness:

$(xy)[x \leftarrow z][z \leftarrow I] \rightarrow (xy)[x \leftarrow I][z \leftarrow I] \rightarrow (Iy)[x \leftarrow I][z \leftarrow I]$

\hookrightarrow It is (indirectly) useful!

- Renaming chains:

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

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.

↪ 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.

↪ 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_{\exists} to design term structures.

Formulas are **polarized**:

- Implications are **negative**
- Atomic formulas are either **negative** or **positive**

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

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

Negative/positive λ -terms

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

Formulas are **polarized**:

- Implications are **negative**
- Atomic formulas are either **negative** or **positive**

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

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

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 **positive**

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 **positive**

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}

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

- 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:

$$\begin{aligned} & x[x \leftarrow yy][y \leftarrow zz'][z \leftarrow \lambda w.w'[w' \leftarrow ww]] \\ \rightarrow_{\text{OEL}} & x[x \leftarrow yy][y \leftarrow (\lambda w.w'[w' \leftarrow ww])z'][z \leftarrow \lambda w.w'[w' \leftarrow ww]] \\ & x[x \leftarrow w'_1 w'_1][w'_1 \leftarrow z' z'][z \leftarrow \lambda w.w'[w' \leftarrow ww]] \end{aligned}$$

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

Positive λ -calculus λ_{pos}

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

- 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:

$$\begin{aligned} & x[x \leftarrow yy][y \leftarrow zz'][z \leftarrow \lambda w.w'[w' \leftarrow ww]] \\ \rightarrow_{\text{OEL}} & x[x \leftarrow yy][y \leftarrow (\lambda w.w'[w' \leftarrow ww])z'][z \leftarrow \lambda w.w'[w' \leftarrow ww]] \\ & x[x \leftarrow w'_1 w'_1][w'_1 \leftarrow z' z'][z \leftarrow \lambda w.w'[w' \leftarrow ww]] \end{aligned}$$

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

Positive λ -calculus λ_{pos}

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

- 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:

$$\begin{aligned} & x[x \leftarrow yy][y \leftarrow zz'][z \leftarrow \lambda w.w'[w' \leftarrow ww]] \\ \rightarrow_{\text{OEL}} & x[x \leftarrow yy][y \leftarrow (\lambda w.w'[w' \leftarrow ww])z'][z \leftarrow \lambda w.w'[w' \leftarrow ww]] \\ & x[x \leftarrow w'_1 w'_1][w'_1 \leftarrow z' z'][z \leftarrow \lambda w.w'[w' \leftarrow ww]] \end{aligned}$$

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

Positive λ -calculus λ_{pos}

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

- 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:

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

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

Positive λ -calculus λ_{pos}

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

- 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:

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

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

Positive λ -calculus λ_{pos}

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

- 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:

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

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

Positive λ -calculus λ_{pos}

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

- 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:

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

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

Positive λ -calculus λ_{pos}

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

- 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:

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

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

Explicit positive λ -calculus λ_{xpos}

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

- 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:

$$\begin{aligned} & x[x \leftarrow yy][y \leftarrow zz'][z \leftarrow \lambda w.w'[w' \leftarrow ww]] \\ \rightarrow_{oe_+} & x[x \leftarrow yy][y \leftarrow (\lambda w.w'[w' \leftarrow ww])z'][z \leftarrow \lambda w.w'[w' \leftarrow ww]] \\ \rightarrow_{om_+} & x[x \leftarrow w'_1 w'_1][w'_1 \leftarrow z' z'][z \leftarrow \lambda w.w'[w' \leftarrow ww]] \end{aligned}$$

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

Explicit positive λ -calculus λ_{xpos}

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

- 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:

$$\begin{aligned} & x[x \leftarrow yy][y \leftarrow zz'][z \leftarrow \lambda w.w'[w' \leftarrow ww]] \\ \rightarrow_{oe_+} & x[x \leftarrow yy][y \leftarrow (\lambda w.w'[w' \leftarrow ww])z'][z \leftarrow \lambda w.w'[w' \leftarrow ww]] \\ \rightarrow_{om_+} & x[x \leftarrow w'_1 w'_1][w'_1 \leftarrow z' z'][z \leftarrow \lambda w.w'[w' \leftarrow ww]] \end{aligned}$$

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

Value substitution calculus λ_{VSC}

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

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

There are two rules in λ_{VSC} :

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

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

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

$$C[x \leftarrow u] \rightarrow C[v] [x \leftarrow u]$$

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

Value substitution calculus λ_{VSC}

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

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

There are two rules in λ_{VSC} :

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

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

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

$$C[x \leftarrow u] \rightarrow C[v] \quad (v \text{ is a value})$$

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

Value substitution calculus λ_{VSC}

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

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

There are two rules in λ_{VSC} :

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

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

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

$$C[x \leftarrow v] \rightarrow C[v]$$

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

Value substitution calculus λ_{VSC}

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

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

There are two rules in λ_{VSC} :

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

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

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

$$C(x)[x \leftarrow v] \rightarrow C(v)[x \leftarrow v]$$

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

Value substitution calculus λ_{VSC}

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

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

There are two rules in λ_{VSC} :

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

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

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

$$C(x)[x \leftarrow v] \rightarrow C(v)[x \leftarrow v]$$

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

Value substitution calculus λ_{VSC}

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

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

There are two rules in λ_{VSC} :

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

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

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

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

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

Value substitution calculus λ_{VSC}

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

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

There are two rules in λ_{VSC} :

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

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

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

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

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

Value substitution calculus λ_{VSC}

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

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

There are two rules in λ_{VSC} :

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

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

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

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

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

Positive Focusing is Directly Useful

Dissecting λ_{VSC}

λ_{xpos} is directly useful while λ_{VSC} is not.

In order to relate λ_{VSC} to λ_{xpos} , we define a core calculus of λ_{VSC} which is essentially equivalent to λ_{VSC} and captures **direct usefulness**.

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

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

Core reduction = $\rightarrow_{\text{m}} + \rightarrow_{e_{\text{var}}} + \rightarrow_{e_{\text{u}}}$

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

Dissecting λ_{VSC}

λ_{xpos} is directly useful while λ_{VSC} is not.

In order to relate λ_{VSC} to λ_{xpos} , we define a core calculus of λ_{VSC} which is essentially equivalent to λ_{VSC} and captures **direct usefulness**.

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

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

Core reduction = $\rightarrow_m + \rightarrow_{e_{\text{var}}} + \rightarrow_{e_u}$

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

Dissecting λ_{VSC}

λ_{xpos} is directly useful while λ_{VSC} is not.

In order to relate λ_{VSC} to λ_{xpos} , we define a core calculus of λ_{VSC} which is essentially equivalent to λ_{VSC} and captures **direct usefulness**.

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

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

Core reduction = $\rightarrow_m + \rightarrow_{e_{\text{var}}} + \rightarrow_{e_u}$

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

Dissecting λ_{VSC}

λ_{xpos} is directly useful while λ_{VSC} is not.

In order to relate λ_{VSC} to λ_{xpos} , we define a core calculus of λ_{VSC} which is essentially equivalent to λ_{VSC} and captures **direct usefulness**.

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

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

Core reduction = $\rightarrow_{\text{m}} + \rightarrow_{e_{\text{var}}} + \rightarrow_{e_{\text{u}}}$

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

Dissecting λ_{VSC}

λ_{xpos} is directly useful while λ_{VSC} is not.

In order to relate λ_{VSC} to λ_{xpos} , we define a core calculus of λ_{VSC} which is essentially equivalent to λ_{VSC} and captures **direct usefulness**.

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

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

Core reduction = $\rightarrow_{\text{m}} + \rightarrow_{e_{\text{var}}} + \rightarrow_{e_{\text{u}}}$

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

Dissecting λ_{VSC}

λ_{xpos} is directly useful while λ_{VSC} is not.

In order to relate λ_{VSC} to λ_{xpos} , we define a core calculus of λ_{VSC} which is essentially equivalent to λ_{VSC} and captures **direct usefulness**.

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

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

Core reduction = $\rightarrow_{\text{m}} + \rightarrow_{e_{\text{var}}} + \rightarrow_{e_{\text{u}}}$

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

Positive Focusing is Directly Useful

Big picture

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

Big picture

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

t ————— $\rightarrow^* u$

Big picture

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

t

t'

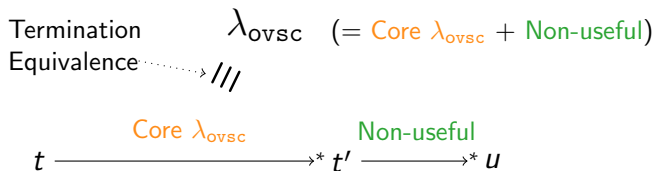
u

Big picture

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

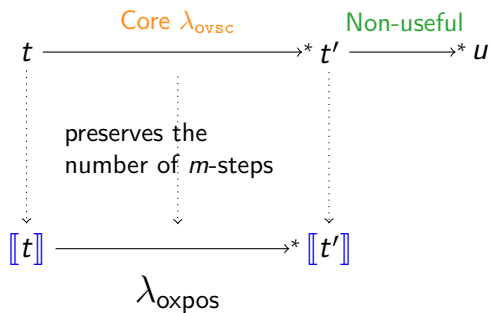


Big picture



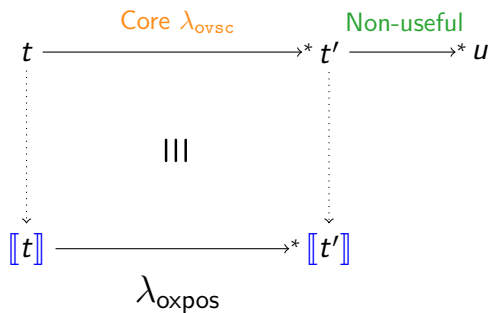
Big picture

Termination
Equivalence $\dashv\dashv\dashv$ λ_{ovsc} (= Core λ_{ovsc} + Non-useful)



Big picture

Termination
Equivalence λ_{ovsc} (= Core λ_{ovsc} + Non-useful)



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 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 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!