The Spirit of Node Replication

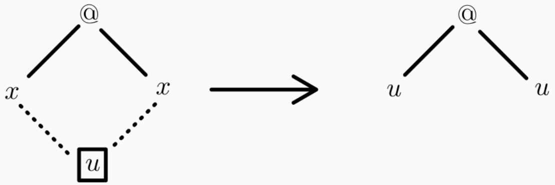
Loïc Peyrot
Joint work with Delia Kesner and Daniel Ventura

October 30, 2020

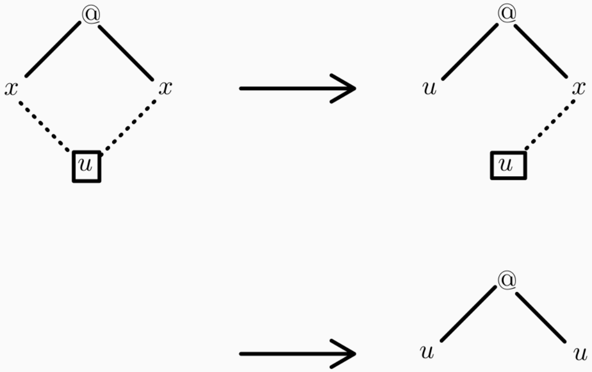
IRIF/PPS, Université de Paris

Introduction

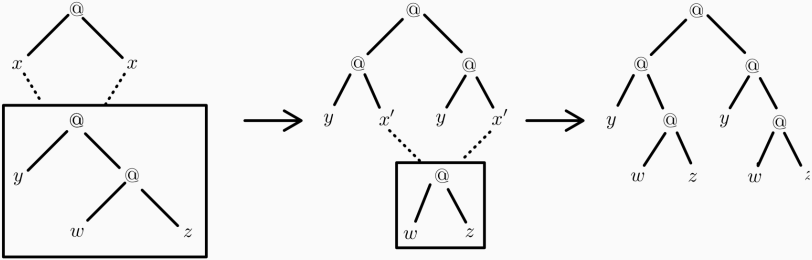
Full substitution



Partial substitution



Atomic substitution



Node Replication

Replication of λ-terms node by node, *i.e.* constructor by constructor.

Sharing equals less work

Idea: Sharing avoids useless work.

Node replication is fine-grained, so that it gives precise control over what stays shared and what is replicated.

Sharing graphs implement node replication

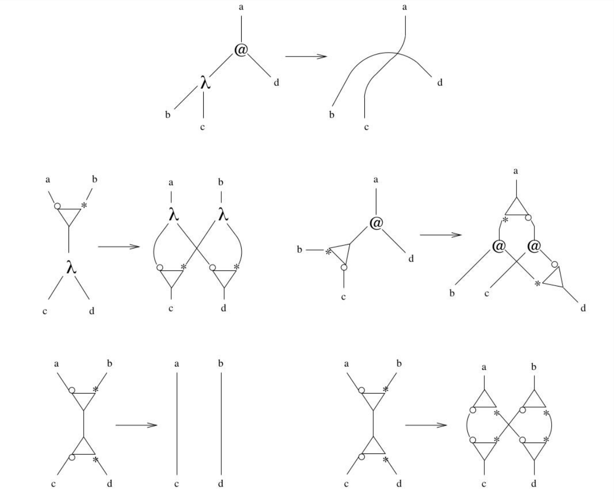


Figure 1: Reduction of sharing graphs (Asperti, Guerrini: The Optimal Implementation of Functional Programming Languages)

Optimality

Definition (Optimality (Lévy))

A strategy is optimal if it reaches a normal form in the same number of steps as the shortest possible reduction sequence in the usual λ-calculus without sharing.

Sharing graphs implement (strong) Lévy-optimal reduction.

A tool for sharing: explicit substitutions

Explicit substitutions are one of the tools used to implement sharing in the λ-calculus.
Advantages: simple syntax, intuitive semantics, close relation to concrete implementations and abstract machines...

Terms: $x \quad | \quad \lambda x.t \quad | \quad tu \quad | \quad t[x\backslash u]$.

We want to relate explicit substitutions and node replication.

Rules of explicit substitutions

Reduction of a redex to an explicit substitution is implemented by a B-step.

$$(\lambda x.t)u \to_{\mathsf{B}} t[x\backslash u]$$

Notice that explicit substitutions can block potential redexes: $(\lambda y.t)[x\backslash u]s$ can't be fired with the B-rule. Permutation rules can be used to move substitutions (avoiding capture), *e.g.*:

$$t[x\backslash u]s \to_{\pi} (ts)[x\backslash u].$$

Example

$(\lambda y.t)[x\backslash u]s \to_{\pi} ((\lambda y.t)s)[x\backslash u] \to_{\pi} t[y\backslash s][x\backslash u]$

Distance highlights the computational content

Permutation rules do not hold computational content, rather structural one. With distance, we keep only computationally relevant rules.

distant B (dB) rule

$\mathsf{L}\langle(\lambda x.t)\rangle u \to_{\mathsf{dB}} \mathsf{L}\langle t[x\backslash u]\rangle$.

Example

$(\lambda y.t)[x\backslash u]s \to_{\pi} ((\lambda y.t)s)[x\backslash u] \to_{\pi} t[y\backslash s][x\backslash u]$ becomes $(\lambda y.t)[x\backslash u]s \to_{\mathsf{dB}} t[y\backslash s][x\backslash u]$.

Reduction with distance is strongly related with reduction in a graphical formalism.

Explicit substitutions implement full and partial substitution

We can convenientely handle different kinds of substitutions thanks to explicit substitutions and distance.

1. *Full* substitution: $t[x \backslash u] \to t\{x \backslash u\}$.
2. *Partial* substitution: $\mathsf{C}\langle\!\langle x \rangle\!\rangle[x \backslash u] \to \mathsf{C}\langle\!\langle t \rangle\!\rangle[x \backslash u]$.

Call-by-need is lazy

Call-by-name (CBN) can replicate redexes ($I \triangleq \lambda z.z$).

$$(\lambda x.xx)(II) \to_\beta (II)(II) \to_\beta I(II) \to_\beta II \to_\beta I$$

By sharing the argument after reduction, call-by-need (CBNeed) avoids duplication of work:

$$\begin{array}{llllll}
(\lambda x.xx)(II) & \to_{\mathsf{dB}} & (xx)[x\backslash II] & \to_{\mathsf{dB}} & (xx)[x\backslash z[z\backslash I]] \\
& \to_{\mathsf{sub}} & (xx)[x\backslash I] & \to_{\mathsf{sub}} & (Ix)[x\backslash I] \\
& \to_{\mathsf{dB}} & z[z\backslash x][x\backslash I] & \to_{\mathsf{sub}} & x[x\backslash I] \\
& \to_{\mathsf{sub}} & I & &
\end{array}$$

But even CBNeed can be redundant

The amount of sharing in usual call-by-need is not sufficient to avoid all useless work.

$$
\begin{array}{lll}
(\lambda x.xx)(\lambda z.z(II)) & \to_{\mathsf{dB}} & (xx)[x\backslash\lambda z.z(II)] \\
 & \to_{\mathsf{sub}} & ((\lambda z.z(II))x)[x\backslash\lambda z.z(II)] \\
 & \to_{\mathsf{dB}} & (z(II))[z\backslash x][x\backslash\lambda z.z(II)] \\
 & \to_{\mathsf{sub}} & (z(II))[z\backslash\lambda z.z(II)] \\
 & \to_{\mathsf{sub}} & (\lambda z.z(II))(II) \\
 & \to_{\mathsf{dB}} & (z(II))[z\backslash(II)] \\
 & \to_{\mathsf{sub}} & (II)(II) \\
 & \to^{*} & I
\end{array}
$$

Maximal free expressions are duplicated by CBNeed

II has no bound occurences of z: it is a (maximal) free expression.

Free expressions: subexpressions not on the path between the binders and the corresponding free occurences of the variables.

Keeping MFEs shared

In the previous example, the abstraction must be duplicated in order to execute the β-reduction. But II can stay shared because it is a free expression.

$$\begin{array}{lll} (\lambda x.xx)(\lambda z.z(II)) & \to_{\mathsf{dB}} & (xx)[x \backslash \lambda z.z(II)] \\ & \to & ((\lambda z.zy)x)[x \backslash \lambda z.zy][y \backslash II] \end{array}$$

$\lambda z.z\Box$ is called the skeleton of the abstraction $\lambda z.z(II)$.

Fully lazy sharing

This optimization of call-by-need which keeps MFEs shared before duplicating an abstraction is called fully lazy sharing.

With this level of sharing, we reach optimality *à la* Lévy in the *weak* setting.

Node replication in a term calculus: the atomic λ-calculus

Gundersen, Heijltjes and Parigot's atomic λ-calculus (λa) is an extended λ-calculus implementing node replication through a variant of explicit substitutions.

It is a Curry-Howard interpretation of the deep inference logical formalism.

They argue that their calculus implements full laziness.

Drawbacks of the atomic λ-calculus

Problem: the atomic λ-calculus implements explicit contraction and weakening of variables.

Example

The explicit substitutions are of the shape: $[x_1, \ldots, x_n \backslash t]$.

This makes the syntax and semantics of the calculus fairly complicated…

We want to keep only the mechanism of node replication.

Plan

The λR-calculus: a calculus focusing on node replication

Two weak strategies

- Call-by-name
- Fully lazy call-by-need

Quantitative types, a tool for observational equivalence

The λR-calculus: a calculus focusing on node replication

An example of reduction duplicating applications

$$
\begin{aligned}
(\lambda x.xx)(y(wz)) &\to_{\mathtt{dB}} (xx)[x\backslash y@(wz)] \\
&\to_{\mathtt{app}} ((x_1@x_2)(x_1@x_2))[x_1\backslash y][x_2\backslash wz] \\
&\to_{\mathtt{sub}} ((yx_2)(yx_2))[x_2\backslash w@z] \\
&\to_{\mathtt{app}} (y(x_3@x_4))(y(x_3@x_4))[x_3\backslash w][x_4\backslash z] \\
&\to_{\mathtt{sub}} (y(x_3z))(y(x_3z))[x_3\backslash w] \\
&\to_{\mathtt{sub}} (y(wz))(y(wz))
\end{aligned}
$$

An example duplicating an abstraction

$$
\begin{aligned}
(\lambda x.xx)(\lambda y.y) &\to_{\texttt{dB}} (xx)[x \backslash \lambda y.y] \\
&\to_{\texttt{dist}} (xx)[x \backslash\!\!\backslash \lambda y.z[z \backslash y]] \\
&\to_{\texttt{sub}} (xx)[x \backslash\!\!\backslash \lambda y.y] \\
&\to_{\texttt{abs}} (\lambda y.y)(\lambda y.y)
\end{aligned}
$$

λR-terms

We add two constructors to the λ-calculus: explicit substitutions and distributors.

$$\textbf{Terms } t, u, s ::= x \mid \lambda x.t \mid tu \mid t[x \backslash u] \mid t[x \backslash\!\!\backslash \lambda y.u]$$

We call explicit cut a constructor which is either an explicit substitution or a distributor, and we write it $[x \lhd u]$.
Pure terms (p, q) are terms without explicit cuts.

$$\textbf{List contexts } \mathsf{L} ::= \Box \mid \mathsf{L}[x \lhd u]$$

Operational semantics

$$
\begin{array}{lll}
\mathsf{L}\langle \lambda x.t\rangle u & \mapsto_{\mathtt{dB}} & \mathsf{L}\langle t[x\backslash u]\rangle \\
t[x\backslash \mathsf{L}\langle uv\rangle] & \mapsto_{\mathtt{app}} & \mathsf{L}\langle t\{x\backslash yz\}[y\backslash u][z\backslash v]\rangle \\
t[x\backslash \mathsf{L}\langle \lambda y.u\rangle] & \mapsto_{\mathtt{dist}} & \mathsf{L}\langle t[x\backslash\!\!\backslash \lambda y.z[z\backslash u]]\rangle \\
t[x\backslash\!\!\backslash \lambda y.u] & \mapsto_{\mathtt{abs}} & \mathsf{L}\langle t\{x\backslash \lambda y.p\}\rangle \\
t[x\backslash \mathsf{L}\langle y\rangle] & \mapsto_{\mathtt{var}} & \mathsf{L}\langle t\{x\backslash y\}\rangle
\end{array}
$$

In `dist`, we suppose $u \to_{\pi}^{*} \mathsf{L}\langle p\rangle$ and $y \notin \mathtt{fv}(\mathsf{L})$.

λR and λ simulate each other

We define an unfolding $t^{\downarrow}$ of cuts, such that $(t[x \lhd u])^{\downarrow} = t^{\downarrow}\{x \backslash u^{\downarrow}\}$.

Lemma (Simulation to the λ-calculus)

- $t \to_{\mathsf{s}} u \implies t^{\downarrow} = u^{\downarrow}$.
- $t \to_{\mathsf{dB}} u \implies t^{\downarrow} \to_{\beta}^{*} u^{\downarrow}$.

Lemma (Simulation from the λ-calculus)

$p \to_{\beta} q \implies p \to_{\mathsf{dB}} \to_{\mathsf{s}}^{+} q$.

λR and λa simulate each other

λa simulates the λ-calculus and vice-versa.
Thus, taking the λ-calculus as an intermediate language gives a simulation between λR and λa.

λR is confluent

The calculus is confluent.

The proof is simple and relies on simulation in the λ-calculus, and on the termination of $\rightarrow_s$.

A witness for $\to_{\mathtt{s}}$: levels

We define a combinatorial tool to prove termination of $\to_{\mathtt{s}}$.

Definition (Level of a variable)

Maximal number of explicit substitutions one has to go through to reach the variable.

Example

$t = x[x\backslash z[y\backslash w]][w\backslash w']$

1. $\mathtt{lv}_z(t) = 1$,
2. $\mathtt{lv}_{w'}(t) = 3$,
3. $\mathtt{lv}_y(t) = 0$ because $y \notin \mathtt{fv}(t)$.

A witness for $\rightarrow_{\texttt{var}}$

The level of a variable decreases when using rule `var`.

Example

In $t = (yy)[y\backslash x][x\backslash z] \rightarrow_{\texttt{var}} (xx)[x\backslash z] = u$,
$\texttt{lv}_z(t) = 2 > 1 = \texttt{lv}_z(u)$.

We can naturally extend the notion of level to terms and constructors in order to account for other `s`-rules.

Example

In $t = (yy)[y\backslash \lambda z.xz]$, the level of $\lambda z.xz$ and of the abstraction and application constructors is $1 = \texttt{lv}_y(yy) + 1$.

Witnesses for $\to_{\texttt{app}}$ and $\to_{\texttt{dist}}$

The level of an application decreases when using rule `app`.

Example

In $t = (yy)[y\backslash x@z] \to_{\texttt{app}} ((y_1@y_2)(y_1@y_2))[y_1\backslash x][y_2\backslash z]$,
$\texttt{lv}_@(t) = 2 > 1 = \texttt{lv}_@(u)$.

The level of an abstraction decreases when using rule `abs`.

Example

In $t = (yy)[y\backslash \lambda z.z] \to_{\texttt{app}} (yy)[y\backslash\!\!\backslash \lambda z.x[x\backslash z]]$,
$\texttt{lv}_{\lambda z}(t) = 2 > 1 = \texttt{lv}_{\lambda z}(u)$.

Two weak strategies

Calculi and strategies

Calculi Non-deterministic rewriting relations.

Strategies Implement a specific deterministic evaluation.

The atomic λ-calculus is only studied as a calculus, no strategy is formalised.

Two strategies

We give two weak strategies (no reduction under abstraction) to argue for the following statements.

CBN Node replication implements standard evaluation strategies.

FLNeed Node replication is a tool of choice for implementing full laziness.

A natural restriction on the syntax simplifies the semantics

No explicit cuts under abstractions (except distributors).

Invariant of the reduction, and true when starting from a λ-term.

Simplifies the semantics of the calculus and avoids π-rules: the distributors are of the shape $[x \mathbin{\backslash\!\!\backslash} \lambda y.\mathsf{L}\langle p\rangle]$.

Properties of CBN: diamond and simulation

CBN is not deterministic, but it is diamond.

Call-by-name in the λ-calculus is also known as weak head reduction $\rightarrow_{\mathtt{whr}}$.

Lemma (Simulation)

- $t \rightarrow_{\mathtt{name}} u \implies t^{\downarrow} \rightarrow^{*}_{\mathtt{whr}} u^{\downarrow}$.
- $p \rightarrow_{\mathtt{whr}} q \implies p \rightarrow^{+}_{\mathtt{name}} q$.

A key operation: splitting the skeleton and MFEs

Remember: full laziness splits an abstraction into its skeleton and its MFEs.

A skeleton is a term with a finite number of holes in place of the MFEs.

Examples

- $t = \lambda z.z(II)$. Skeleton: $\lambda z.z\Box$. Multiset of MFEs: $[II]$.
- $t = \lambda z.(II)z(II)$. Skeleton: $\lambda z.\Box z\Box$. MFEs: $[II, II]$.
- $t = \lambda y.\lambda z.(yw)z$. Skeleton: $\lambda y.\lambda z.(y\Box)z$. MFEs: $[w]$.

Splitting in λR

$$\begin{aligned}
(\lambda x.xx)(\lambda z.z(II)) &\to_{\mathtt{dB}} && (xx)[x\backslash\lambda z.z(II)] \\
&\to_{\mathtt{dist}} && (xx)[x\backslash\!\backslash\lambda z.y[y\backslash z(II)]] \\
&\to_{\mathtt{app}} && (xx)[x\backslash\!\backslash\lambda z.(y_1 y_2)[y_1\backslash z][y_2\backslash II]] \\
&\to_{\mathtt{var}} && (xx)[x\backslash\!\backslash\lambda z.(z y_2)[y_2\backslash II]]
\end{aligned}$$

There must be no free occurence of z inside a cut in the distributor.

Steps necessary for splitting

Before replicating the value $\lambda y.p$ in $t[x \backslash \lambda y.p]$, we must:

1. Apply rule `dist` to create a distributor.
2. Use `s`-rules on the cuts in which y is free to get the skeleton.

A function for splitting

We want to assemble all these steps into one, for a more elegant presentation. We will define a function $\Downarrow_{\mathsf{st}}$ such that

$$\lambda y.z[z\backslash p] \Downarrow_{\mathsf{st}} \lambda y.\mathsf{L}\langle p\text{'}\rangle,$$

where L is a list of explicit substitutions containaing the MFEs, linked to the skeleton p' by fresh variables in the holes.

Example

$\lambda z.x[x\backslash(II)z(II)] \Downarrow_{\mathsf{st}} \lambda z.(x_1 z x_2)[x_1\backslash II][x_2\backslash II]$

Defining $\Downarrow_{\mathsf{st}}$

$\Downarrow_{\mathsf{st}}$ is defined on terms of the shape $\lambda y.\mathsf{L}\langle p\rangle$.

We use s-rules, parametrized by the binding variable we are considering (*e.g.* y).

This reduction relation is confluent and terminating so that the function $\Downarrow_{\mathsf{st}}$ is well-defined.

Example

Let the abstraction to duplicate be $\lambda y.\lambda z.(yt)z$, where t is an MFE. Its skeleton is $\lambda y.\lambda z.(y\Box)z$.

$$
\begin{array}{lll}
\lambda y.x[x\backslash\lambda z.(yt)z] & \rightarrow^{y}_{\texttt{dist}} & \lambda y.x[x\backslash\!\!\backslash\lambda z.w[w\backslash(yt)z]] \\
 & \rightarrow^{z}_{\texttt{app}} & \lambda y.x[x\backslash\!\!\backslash\lambda z.(w_1 w_2)[w_1\backslash yt][w_2\backslash z]] \\
 & \rightarrow^{z}_{\texttt{var}} & \lambda y.x[x\backslash\!\!\backslash\lambda z.(w_1 z)[w_1\backslash yt]] \\
 & \rightarrow^{y}_{\texttt{abs}} & \lambda y.(\lambda z.w_1 z)[w_1\backslash yt] \\
 & \rightarrow^{y}_{\texttt{app}} & \lambda y.(\lambda z.(x_1 x_2)z)[x_1\backslash y][x_2\backslash t] \\
 & \rightarrow^{y}_{\texttt{var}} & \lambda y.(\lambda z.(y x_2)z)[x_2\backslash t]
\end{array}
$$

The innermost abstraction is treated first.

The fully lazy strategy

$$
\begin{array}{lll}
\mathtt{L}\langle \lambda x.p \rangle u & \mapsto_{\mathtt{dB}} & \mathtt{L}\langle p[x \backslash u] \rangle \\
\mathtt{N}\langle\!\langle x \rangle\!\rangle[x \backslash \mathtt{L}_1 \langle \lambda y.p \rangle] & \mapsto_{\mathtt{spl}} & \mathtt{L}_1 \langle \mathtt{L}_2 \langle \mathtt{N}\langle\!\langle x \rangle\!\rangle[x \backslash\!\!\backslash \lambda y.p'] \rangle \rangle \\
\mathtt{N}\langle\!\langle x \rangle\!\rangle[x \backslash\!\!\backslash v] & \mapsto_{\mathtt{sub}} & \mathtt{N}\langle\!\langle v \rangle\!\rangle[x \backslash\!\!\backslash v]
\end{array}
$$

Where in rule $\mapsto_{\mathtt{spl}}$, $\lambda y.z[z \backslash p] \Downarrow_{\mathtt{st}} \lambda y.\mathtt{L}_2 \langle p' \rangle$

Note that the substitution is partial, as usual in call-by-need: only the needed occurence is duplicated.

Example

$$
\begin{aligned}
(\lambda x.xx)(\lambda y.\lambda z.(yt)z) \quad &\rightarrow_{\texttt{dB}} \quad (xx)[x\backslash\lambda y.\lambda z.(yt)z] \\
&\rightarrow_{\texttt{spl}} \quad (xx)[x\backslash\!\!\backslash\lambda y.\lambda z.(yw)z][w\backslash t] \\
&\rightarrow_{\texttt{sub}} \quad ((\lambda y.\lambda z.(yw)z)x)[w\backslash t]
\end{aligned}
$$

Quantitative types, a tool for observational equivalence

Intersection types

$$\frac{\Gamma \vdash t : \tau \qquad \Delta \vdash t : \sigma}{\Gamma + \Delta \vdash t : \tau \wedge \sigma}$$

Intersection types have a strong denotational flavor: typability and normalisation coincide.

The type system characterizes normalisation

We define a unique type system $\mathcal{V}$ for both strategies. In both strategies we prove (for $\mathtt{x} \in \{\mathtt{name}, \mathtt{flneed}\}$):

Lemma

t is $\mathtt{x}$-normalisable $\iff$ t is typable in $\mathcal{V}$.

Proof that normalisable terms are typable

Proof. t **x-normalisable** $\implies$ t **typable.**

1. Show that normal forms are typable.
2. Show subject expansion: if there is a type derivation of $\Gamma \vdash t_1 : \tau$ in $\mathcal{V}$ and a term $t_0 \to_{\mathsf{x}} t_1$, then there is a type derivation of $\Gamma \vdash t_0 : \tau$ in $\mathcal{V}$.

$\square$

Proof that typable terms are normalisable

Proof. t **typable** $\implies$ t **x-normalisable.**

1. Show subject reduction: if there is a type derivation of $\Gamma \vdash t_0 : \tau$ in $\mathcal{V}$ and $t_0 \rightarrow_{\mathsf{x}} t_1$, then there is a type derivation of $\Gamma \vdash t_1 : \tau$ in $\mathcal{V}$.
2. Show that a reduction of typed terms terminate.

$\square$

Non-idempotent intersection types

Second step of the proof: we use non-idempotent intersection (*a.k.a* quantitatives) types.

$$\frac{\Gamma \vdash t : \tau \qquad \Delta \vdash t : \tau}{\Gamma + \Delta \vdash t : [\tau, \tau]}$$

Non-idempotence gives quantitative information and a combinatorial proof of termination.

Weighted subject reduction

Using quantitative types, we can generally find a measure $\mathsf{D}(\Phi)$ on type derivations, such that:

Lemma (Weighted subject reduction)

Let $t_0 \to t_1$ such that there exists $\Phi \triangleright \Gamma \vdash t_0 : \tau$. Then there exists $\Psi \triangleright \Gamma \vdash t_1 : \tau$ such that $\mathsf{D}(\Phi) > \mathsf{D}(\Psi)$.

Which measure should we take?

dB decreases the size of the derivation

$$
\dfrac{\dfrac{\dfrac{}{x : [\mathsf{a}] \vdash x : \mathsf{a}}\ (\mathtt{ax})}{\vdash \lambda x.x : [\mathsf{a}] \to \mathsf{a}}\ (\mathtt{abs}) \qquad \dfrac{\dfrac{}{\vdash \lambda y.z : \mathsf{a}}\ (\mathtt{ans})}{\vdash \lambda y.z : [\mathsf{a}]}\ (\mathtt{many})}{\vdash (\lambda x.x)(\lambda y.z) : \mathsf{a}}\ (\mathtt{app})
$$

dB decreases the size of the derivation

$$
\dfrac{\dfrac{\dfrac{}{x : [\mathsf{a}] \vdash x : \mathsf{a}}\ (\mathtt{ax})}{\vdash \lambda x.x : [\mathsf{a}] \to \mathsf{a}}\ (\mathtt{abs}) \qquad \dfrac{\dfrac{}{\vdash \lambda y.z : \mathsf{a}}\ (\mathtt{ans})}{\vdash \lambda y.z : [\mathsf{a}]}\ (\mathtt{many})}{\vdash (\lambda x.x)(\lambda y.z) : \mathsf{a}}\ (\mathtt{app})
$$

$$
\to_{\mathtt{dB}} \quad \dfrac{\dfrac{}{x : [\mathsf{a}] \vdash x : \mathsf{a}}\ (\mathtt{ax}) \qquad \dfrac{\dfrac{}{\vdash \lambda y.z : \mathsf{a}}\ (\mathtt{ans})}{\vdash \lambda y.z : [\mathsf{a}]}\ (\mathtt{many})}{\vdash x[x \backslash \lambda y.z] : \mathsf{a}}\ (\mathtt{cut})
$$

Size is not a good measure for us

Taking the size of the proof does not work: in rules `app` and `dist` we add new fresh variables which make the proof grow by a number of axiom rules.

$$
\begin{array}{lll}
t[x\backslash \mathsf{L}\langle uv\rangle] & \mapsto_{\texttt{app}} & \mathsf{L}\langle t\{x\backslash yz\}[y\backslash u][z\backslash v]\rangle \\
t[x\backslash \mathsf{L}\langle \lambda y.u\rangle] & \mapsto_{\texttt{dist}} & \mathsf{L}\langle t[x\backslash\!\!\backslash \lambda y.z[z\backslash u]]\rangle
\end{array}
$$

An typed measure decreasing on all rules

Idea: we weight the constructors with their levels. The level of a constructor decreases with `s`, and so does the measure.

We define the size of a derivation $|\Phi|$ by the number of (`abs`), (`ans`) and (`app`) rules.

Definition (Measure)

$\mathrm{D}(\Phi) = (l, m, n)$ (ordered lexicographically), where:

- $l = |\Phi|$,
- m is the size of Φ weighted by the levels,
- n is the number of `ax` rules.

Example

We consider the following reduction.

$$\begin{aligned}
(\lambda x.x)(yz) &\rightarrow_{\mathtt{dB}} x[x\backslash yz] \\
&\rightarrow_{\mathtt{app}} (x_1 x_2)[x_1\backslash y][x_2\backslash z] \\
&\rightarrow_{\mathtt{sub}} (y x_2)[x_2\backslash z] \\
&\rightarrow_{\mathtt{sub}} yz
\end{aligned}$$

Example

Let $\sigma = [\tau] \to \tau$.

$$\dfrac{\dfrac{}{x : [\tau] \vdash x : \tau}\,(\mathtt{ax})}{\vdash \lambda x.x : \sigma}\,(\mathtt{abs}) \quad \dfrac{\dfrac{\dfrac{}{y : [\sigma] \vdash y : \sigma}\,(\mathtt{ax}) \quad \dfrac{}{z : [\tau] \vdash z : \tau}\,(\mathtt{ax})}{y : [\sigma], z : [\tau] \vdash yz : \tau}\,(\mathtt{app})}{y : [\sigma], z : [\tau] \vdash yz : [\tau]}\,(\mathtt{many})$$
$$\overline{\Phi_1 \rhd y : [\sigma], z : [\tau] \vdash (\lambda x.x)(yz) : \tau}\,(\mathtt{app})$$

$\mathsf{D}(\Phi_1) = (3, 3, 3)$

Example

Let $\sigma = [\tau] \to \tau$.

$$
\dfrac{\dfrac{}{x : [\tau] \vdash x : \tau}\,(\mathtt{ax}) \qquad \dfrac{\dfrac{\dfrac{}{y : [\sigma] \vdash y : \sigma}\,(\mathtt{ax}) \quad \dfrac{}{z : [\tau] \vdash z : \tau}\,(\mathtt{ax})}{y : [\sigma], z : [\tau] \vdash yz : \tau}\,(\mathtt{app})}{y : [\sigma], z : [\tau] \vdash yz : [\tau]}\,(\mathtt{many})}{\Phi_2 \triangleright y : [\sigma], z : [\tau] \vdash x[x \backslash yz] : \tau}\,(\mathtt{cut})
$$

$\mathsf{D}(\Phi_2) = (1, 2, 3)$

Example

Let $\sigma = [\tau] \to \tau$.

$$
\dfrac{\dfrac{\Phi_{x_1 x_2} \qquad \dfrac{\dfrac{}{y : [\sigma] \vdash y : \sigma}\ (\texttt{ax})}{y : [\sigma] \vdash y : [\sigma]}\ (\texttt{many})}{x_1 : [\sigma], z : [\tau] \vdash (x_1 x_2)[x_1 \backslash y] : \tau}\ (\texttt{cut}) \qquad \dfrac{\dfrac{}{z : [\tau] \vdash z : \tau}\ (\texttt{ax})}{z : [\tau] \vdash z : [\tau]}\ (\texttt{many})}{\Phi_3 \triangleright y : [\sigma], z : [\tau] \vdash (x_1 x_2)[x_1 \backslash y][x_2 \backslash z] : \tau}\ (\texttt{cut})
$$

$$
\dfrac{\dfrac{}{x_1 : [\sigma] \vdash x_1 : \sigma}\ (\texttt{ax}) \qquad \dfrac{\dfrac{}{x_2 : [\tau] \vdash x_2 : \tau}\ (\texttt{ax})}{x_2 : [\tau] \vdash x_2 : [\tau]}\ (\texttt{many})}{\Phi_{x_1 x_2} \triangleright x_1 : [\sigma], x_2 : [\tau] \vdash x_1 x_2 : \tau}\ (\texttt{app})
$$

$\mathsf{D}\left(\Phi_3\right) = (1, 1, 4)$

Example

Let $\sigma = [\tau] \to \tau$.

$$\dfrac{\dfrac{\dfrac{}{y : [\sigma] \vdash y : \sigma}\,(\mathtt{ax}) \quad \dfrac{\dfrac{}{x_2 : [\tau] \vdash x_2 : \tau}\,(\mathtt{ax})}{x_2 : [\tau] \vdash x_2 : [\tau]}\,(\mathtt{many})}{y : [\sigma], x_2 : [\tau] \vdash y x_2 : \tau}\,(\mathtt{app}) \quad \dfrac{\dfrac{}{z : [\tau] \vdash z : \tau}\,(\mathtt{ax})}{z : [\tau] \vdash z : [\tau]}\,(\mathtt{many})}{\Phi_4 \rhd y : [\sigma], z : [\tau] \vdash (y x_2)[x_2 \backslash z] : \tau}\,(\mathtt{cut})$$

$\mathsf{D}\left(\Phi_4\right) = (1, 1, 3)$

Example

Let $\sigma = [\tau] \to \tau$.

$$
\dfrac{\dfrac{}{y : [\sigma] \vdash y : \sigma}\;(\mathtt{ax}) \quad \dfrac{\dfrac{}{z : [\tau] \vdash z : \tau}\;(\mathtt{ax})}{z : [\tau] \vdash z : [\tau]}\;(\mathtt{many})}{y : [\sigma], z : [\tau] \vdash yz : \tau}\;(\mathtt{app})
$$

$\mathsf{D}\,(\Phi_5) = (1, 1, 2)$

Observational equivalence

Lemma (restatement)

t is x-normalisable $\iff$ t is typable in $\mathcal{V}$.

Observational equivalence is a trivial corollary.

Definition (Observational equivalence)

$t \equiv_{\mathsf{x}} u$ if and only if for any C, $\mathsf{C}\langle t\rangle$ terminates in x $\iff$ $\mathsf{C}\langle u\rangle$ terminates in x.

Theorem

$t \equiv_{\mathtt{name}} u \iff t \equiv_{\mathtt{flneed}} u$.

Conclusion

To sum up, we:

- Defined a new calculus of explicit substitutions inspired by λa which achieves node replication;
- Gave two weak strategies CBN and FLNeed;
- Studied the strategies by means of non-idempotent intersection types and show observational equivalence.

Future works

Future works could be:

- Defining a calculus based on spine duplication; or a classical version of the calculus.
- Relate FLNeed to other strategies of full laziness in the literature.
- Defining a type system giving exact bounds.

Thank you for your attention!