The Spirit of Node Replication

Loïc Peyrot*
Joint work with Delia Kesner and Daniel Ventura

FoSSaCS 2021 – March 31st

*IRIF/PPS, Université de Paris

Different substitutions

Different kinds of substitutions, from different Curry-Howard interpretations:

Full Substitution	Natural Deduction
Linear Substitution	Linear Logic Proof-Nets
Node Replication	Deep Inference

How are programming languages affected by different specific substitution mechanisms?

How are programming languages affected by different specific substitution mechanisms?

Are these different substitution behaviors observable?

Plan

Intuitions on node replication

λR, a λ-calculus for node replication

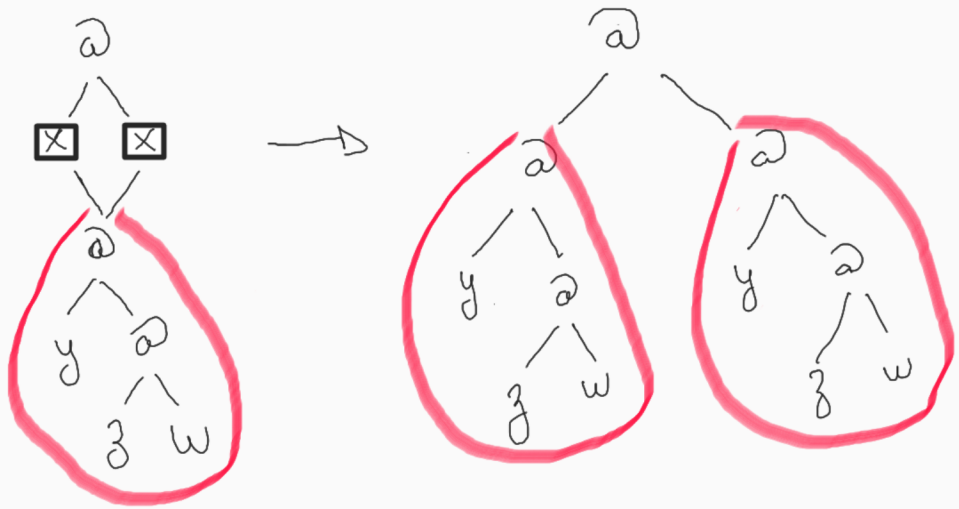
Implementing two programming languages using node replication

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

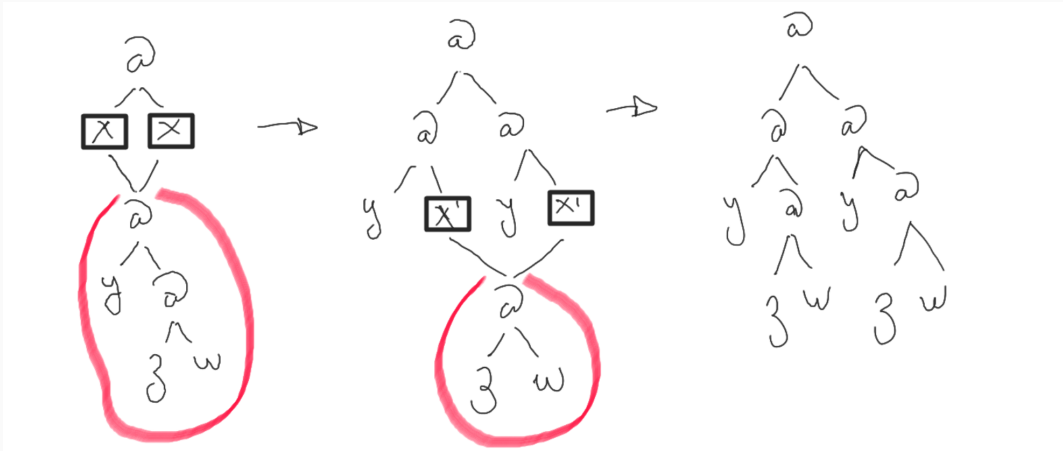
Relating the strategies through intersection type theory

Intuitions on node replication

Graphically



Full Substitution (FS)



Node Replication (NR)

Node replication

Node Replication

Substitution of terms node by node, *i.e.* constructor by constructor.

Node replication allows fine-grained substitution

- Duplication of terms node by node avoids to substitute the whole term at once.
- It is a lazy principle because it is possible to replicate only the subtree that is necessary to continue the reduction.
- Optimisations can be specified, *e.g* full laziness.

Sharing graphs use node replication

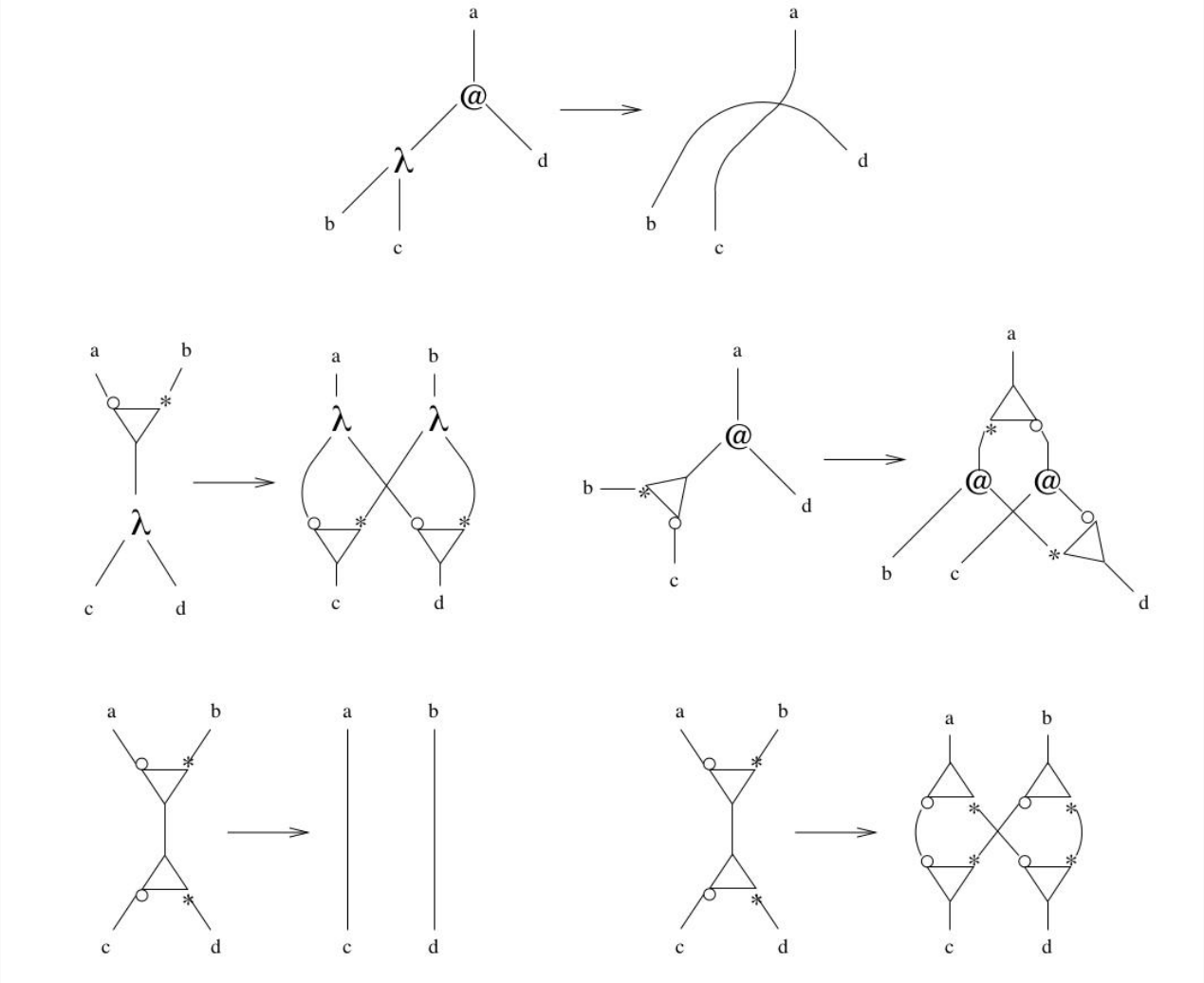


Figure 1: Reduction of sharing graphs (Asperti, Guerrini: The Optimal Implementation of Functional Programming Languages)

λR, a λ-calculus for node replication

The λR-calculus

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

$$\begin{array}{rcl}
(\lambda x.t)[\Gamma]u & \mapsto_{\mathtt{dB}} & t[x\backslash u][\Gamma] \\
t[x\backslash y[\Gamma]] & \mapsto_{\mathtt{var}} & t\{x\backslash y\}[\Gamma] \\
t[x\backslash uv[\Gamma]] & \mapsto_{\mathtt{app}} & t\{x\backslash yz\}[y\backslash u][z\backslash v][\Gamma] \\
t[x\backslash\lambda y.u[\Gamma]] & \mapsto_{\mathtt{dist}} & t[x\backslash\backslash\lambda y.z[z\backslash u]][\Gamma] \\
t[x\backslash\backslash\lambda y.u] & \mapsto_{\mathtt{abs}} & t\{x\backslash\lambda y.p\}[\Gamma]
\end{array}$$

In abs, we suppose $u \to_\pi^* p[\Gamma]$ and $y \notin \mathrm{fv}(\Gamma)$.

Figure 2: λR syntax and operational semantics

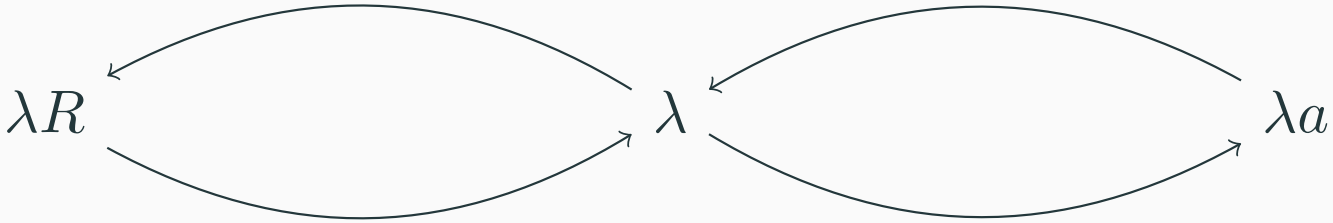
Node replication in a term calculus: the atomic λ-calculus

Our inspiration is Gundersen, Heijltjes and Parigot's atomic λ-calculus (λa):

- It is a term calculus implementing node replication.
- It is a Curry-Howard interpretation of the deep inference logical formalism.
- It deals explicitely with weakening and contraction of variables.

Relation with the λ-calculus

- We have simulation from λR to the λ-calculus (λ) and projection the other way.
- Using the λ-calculus as an intermediate, we have simulations between λR and the atomic λ-calculus λa.



Implementing two programming languages using node replication

Calculi and strategies

Calculi

- Non-deterministic rewriting relations.
- Essential implementation of a theory: can serve as a foundation of multiple programming languages.

Strategies

- Deterministic refinment of a calculus.
- Implementing a specific evaluation procedure.

Call-by-name evaluation in the λ-calculus

Call-by-name in the λ-calculus = weak head reduction $\to_{\mathsf{cbn}}$.

Example

$(\lambda x.xx)(II) \to_{\mathsf{cbn}} (II)(II)$

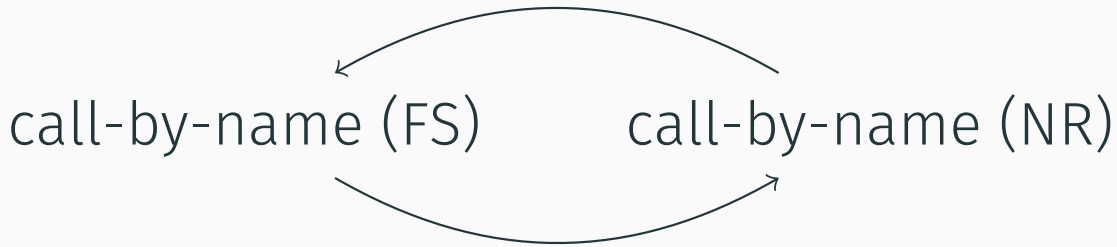
Call-by-name computes weak-head normal forms:

Weak No reduction inside abstractions.
$(\lambda x.Ix)(II) \not\to_{\mathsf{cbn}} (\lambda x.x)(II)$

Head No reduction inside arguments headed by a variable. $x(II) \not\to_{\mathsf{cbn}} xI$

Call-by-name in λR

We apply the same principles to create a strategy `nrcbn` based on λR.



Call-by-need in the λ-calculus

- In call-by-name, duplicating the argument may duplicate redexes.

Example

$(\lambda x.xx)(II) \rightarrow_{\mathsf{cbn}} (II)(II)$

- Call-by-need avoids duplication of redexes by applying memoization: the first demand-driven function call transforms the argument into a value.

Towards full laziness

Call-by-need does not avoid all unnecessary work.

Example

$(\lambda x.xx)(\lambda y.y(II)) \to_{\mathsf{cbneed}} (\lambda y.y(II))(\lambda y.y(II))$

Observation

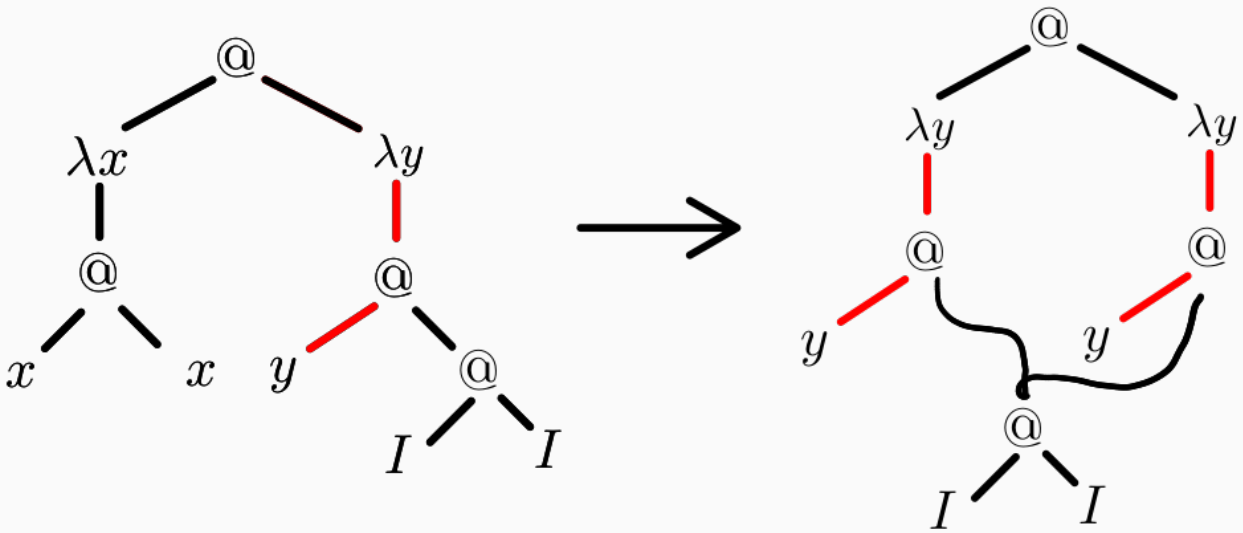
The redex II is not affected by further instantiation of $\lambda y.y(II)$. We can keep it shared!

$$\lambda y.y(II) = \underbrace{\lambda y.yz}_{\text{skeleton}} + \underbrace{[z \backslash II]}_{\text{sharing}}$$

A fully lazy call-by-need strategy

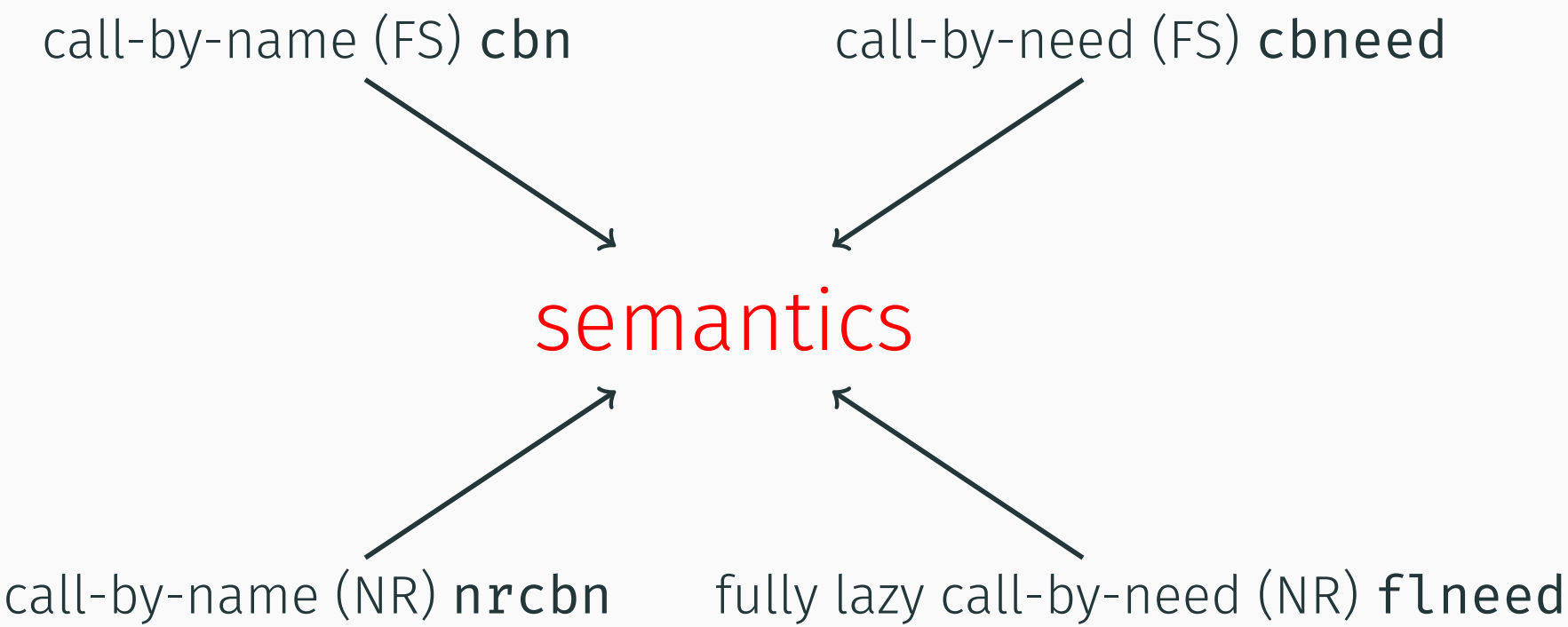
Example

$(\lambda x.xx)(\lambda y.y(II)) \to^*_{\texttt{flneed}} ((\lambda y.yz)(\lambda y.yz))[z\backslash II]$



Relating the strategies through intersection type theory

Putting everything together



Observational equivalence

Definition (Observational equivalence)

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

There is no context distinguishing t and u.

Theorem

$t \equiv_{\mathtt{cbn}} u \iff t \equiv_{\mathtt{nrcbn}} u \iff t \equiv_{\mathtt{flneed}} u \iff t \equiv_{\mathtt{cbneed}} u$.

t and u have the same behaviour, whatever the strategy is.

Intersection types

$$\dfrac{\dfrac{}{x : [[\tau] \to \tau] \vdash x : [\tau] \to \tau} \qquad \dfrac{}{x : [\tau] \vdash x : \tau}}{\dfrac{x : [[\tau] \to \tau, \tau] \vdash xx : \tau}{\vdash \lambda x.xx : [[\tau] \to \tau, \tau] \to \tau}}$$

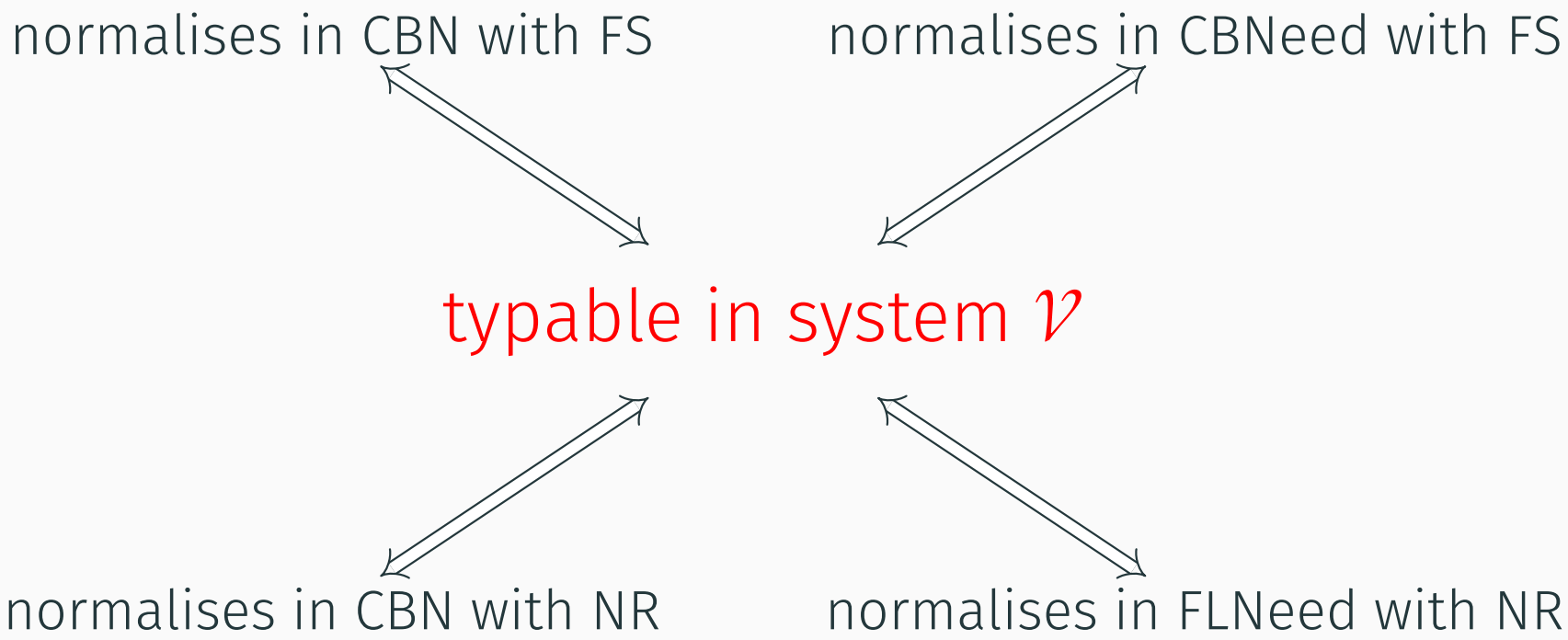
With intersection types:

Typability $\iff$ termination.

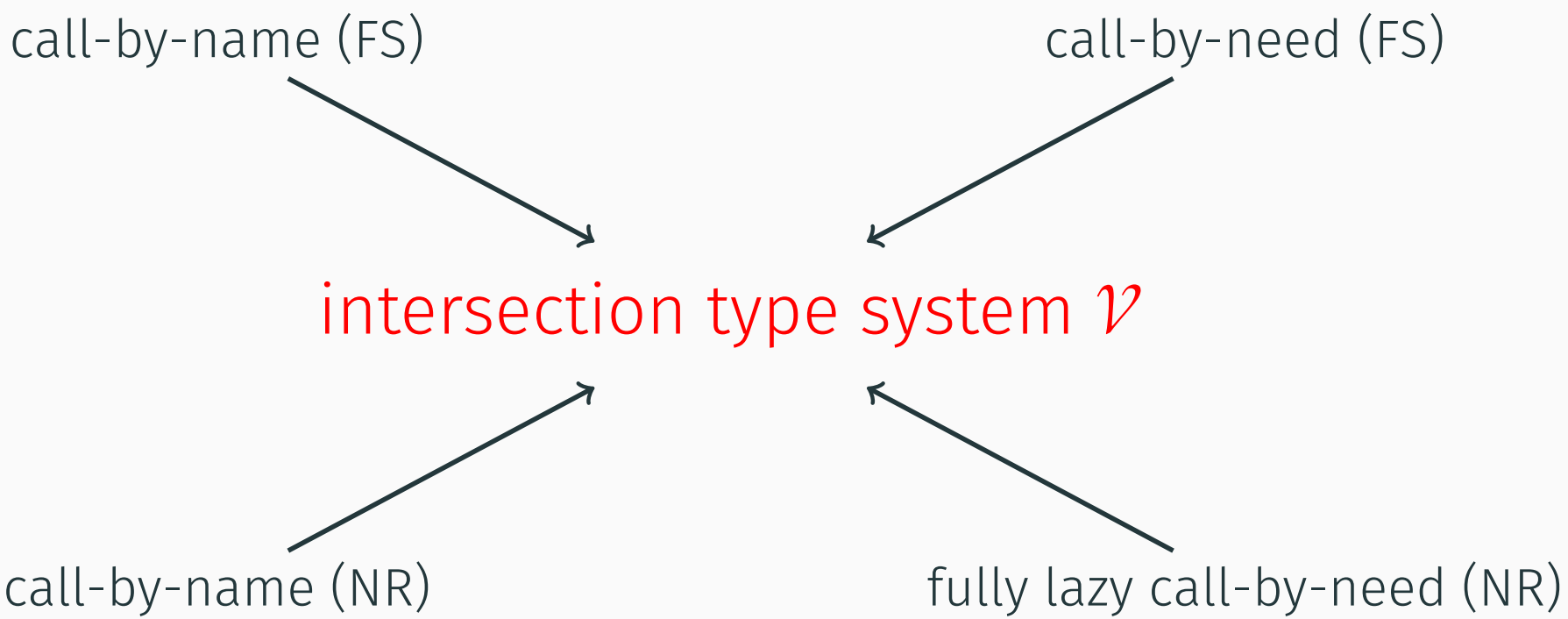
Equivalence of the normalisations

Idea

We use a unique type system $\mathcal{V}$ for all strategies.



A common model



Conclusion

Summary

- Different mechanisms of substitution give rise to different implementations of programming languages.
- Node replication has the same observational behaviour than usual full substitution.
- But operationally, it allows to implement non-trivial optimisations such as full laziness.

Questions and future works

- Abstract machines, complexity costs.
- Extension to strong call-by-need.
- Spine duplication.
- A quantitative deep type system for node replication?

Thank you for your attention!