Classical by-need

Pierre-Marie Pédrot and Alexis Saurin[⋆]

Laboratoire PPS, CNRS, UMR 7126, Univ Paris Diderot,
Sorbonne Paris Cité, PiR2, INRIA Paris Rocquencourt, F-75205, Paris, France

Abstract. Call-by-need calculi are complex to design and reason with. When adding control effects, the very notion of canonicity is irremediably lost, the resulting calculi being necessarily ad hoc. This calls for a design of call-by-need guided by logical rather than operational considerations. Ariola et al proposed such an extension of call-by-need with control making use of Curien and Herbelin's *duality of computation* framework.
In this paper, *Classical by-need* is developed as an alternative extension of call-by-need with control, better-suited for a programming-oriented reader. This method is proof-theoretically oriented by relying on linear head reduction (LHR) – an evaluation strategy coming from linear logic – and on the $\lambda\mu$-calculus – a classical extension of the λ-calculus.
More precisely, the paper contains three main contributions:

- LHR is first reformulated by introducing closure contexts and extended to the $\lambda\mu$-calculus;
- it is then shown how to derive a call-by-need calculus from LHR. The result is compared with standard call-by-need calculi, namely those of Ariola–Felleisen and Chang–Felleisen;
- it is finally shown how to lift the previous item to classical logic, that is from the λ-calculus to the $\lambda\mu$-calculus, providing a classical by-need calculus, that is a lazy $\lambda\mu$-calculus. The result is compared with the call-by-need with control of Ariola et al.

Keywords: call-by-need, classical logic, control operators, lambda-calculus, lambda-mu-calculus, lazy evaluation, linear head reduction, linear logic, Krivine abstract machine, sigma equivalence,

1 Introduction

In his survey on the origins of continuations, Reynolds noticed that *"in the early history of continuations, basic concepts were independently discovered an extraordinary number of times"* [28]. It is actually a well-known fact of the (long) history of science that deep, structuring ideas, are re-discovered several times. Computer science and modern proof theory have much shorter history but are no exception. Very much related to the question of continuations, we may think of double-negation translations or, more recently, Girard and Reynolds' discoveries of, respectively, System F [17] and of the polymorphic λ-calculus [27].

[⋆] This work was partially supported by ANR Project RAPIDO, ANR-14-CE25-0007.

We think that this convergence of structuring ideas and independent discoveries is at play, to some extent, with call-by-need evaluation and linear head reduction: while the first is operationally motivated, the latter comes from the structure of linear logic proofs. This paper aims at demonstrating this and applying this in incorporating first-class control in call-by-need.

Computation on demand. Executing computations which may not be used to produce a value may obviously lead to unnecessary work being done, potentially resulting in non-termination even when a value exists. An alternative is to fire a redex only when it happens to be necessary to pursue the evaluation towards a value.

For instance, it is well-known that while call-by-value may trigger computations that could be completely avoided, resulting in potential non-termination, call-by-name evaluates programs on demand. This is exemplified in:

$$\begin{array}{l} t \equiv \boxed{(\lambda x.\, I)\ (\varDelta\ \varDelta)} \rightarrow_{\text{cbn}} I \\ t \equiv (\lambda x.\, I)\ \boxed{(\varDelta\ \varDelta)} \rightarrow_{\text{cbv}} t \rightarrow_{\text{cbv}} \dots \rightarrow_{\text{cbv}} \dots \end{array}$$

In this example[1], call-by-value reduction will reduce $\varDelta\ \varDelta$ again and again when the redex is of no use for reaching a value while call-by-name simply discards the argument.

Call-by-name, and more precisely (weak) head reduction thus realizes a form of demand-driven computation: a redex is fired only if it contributes to the (weak) head normal form (usually abbreviated as (w)hnf).

On the other hand, call-by-value will happen to be more parsimonious when it comes to arguments which are actually used in the computation: they are evaluated only once, before substituting the value, while call-by-name discipline will redo the same computation several times, as in:

$$\begin{array}{l} u \equiv \boxed{\varDelta\ (I\ I)} \rightarrow_{\text{cbn}} \boxed{I\ I}\ (I\ I) \rightarrow_{\text{cbn}} \boxed{I\ (I\ I)} \rightarrow_{\text{cbn}} \boxed{I\ I} \rightarrow_{\text{cbn}} I \\ u \equiv \varDelta\ \boxed{(I\ I)} \rightarrow_{\text{cbv}} \boxed{\varDelta\ I} \rightarrow_{\text{cbv}} \boxed{I\ I} \rightarrow_{\text{cbv}} I \end{array}$$

In the above example, call-by-name reduction duplicates the computation of $I\ I$ while call-by-value only duplicates value I, resulting in a shorter reduction path to value.

Interestingly, demand-driven computation resulted in two lines of work, one motivated by theoretical purposes and rooted in logic, Danos and Regnier's linear head reduction, the other being motivated by more practical concerns and resulting in the study of lazy evaluation strategies for functional languages.

Linear Head Reduction. Linear head reduction (referred as LHR) was first described by Regnier [25] in his 1992 PhD thesis, albeit under the name of *spinal*

[1] As is usual, $\varDelta$ stands for $\lambda x.\, x\ x$, I for $\lambda y.\, y$ and we write $\rightarrow_{\text{cbn}}$ (resp. $\rightarrow_{\text{cbv}}$) the reductions associated with call-by-name (resp. by value). The redex which is involved in a reduction is emphasized by showing it in a grey box. We will be implicitly working up to α-conversion, and we will use Barendregt's conventions not to capture variables unwillingly.

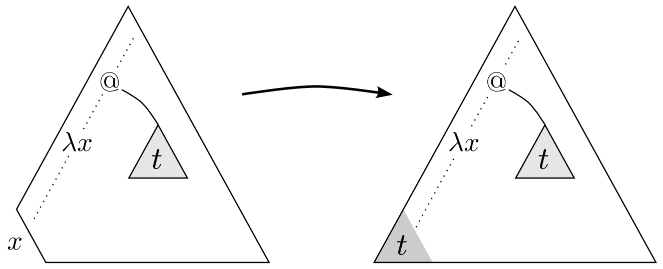


Fig. 1. Linear head reduction

reduction. It was then studied again by Danos and Regnier [14,15] following similar observations made amongst different computational paradigms, namely the Krivine abstract machine [20], proof-nets [18,25], and game semantics [19]. The crucial remark is that the core of these systems does not implement the usual head reduction as thought commonly, but rather uses some more parsimonious reduction, which they define under the name of linear head reduction, which realizes a stronger form of computation-on-demand than call-by-name: the argument of a function cannot be said to truly contribute to the result if it never reaches head position; if it does not, the corresponding redex may only contribute to the (w)hnf in a non-essential way; for instance by blocking other redexes as in $(\lambda x\, y.\, y)\ t\ u$. Linear head reduction makes this observation formal. LHR has two main features:

- first it reduces only the β-redex binding to the leftmost variable occurrence (therefore the "head" from its name) and
- secondly it substitutes for the argument only the head occurrence of the variable (therefore the "linear" from its name) without destroying the fired redex.

A third noticeable point is that linear head reduction is not truly a reduction since it does not reduce only redexes (at least not only β-redexes), but also sorts of "hidden β-redexes" that are true β-redexes only up to an equivalence on λ-terms induced by their encoding in proof nets, namely σ-equivalence (this point shall be made clear later on).

Lazy Evaluation. Wadsworth introduced lazy evaluation [30] as a way to overcome defects of both call-by-name and call-by-value evaluation recalled in the above paragraphs. Lazy evaluation, or Call-by-need, can be viewed as a strategy conciling the best of the by-value and by-name worlds in terms of reductions: a computation is triggered only when it is needed for the evaluation to progress and, in this case, it avoids redoing computations. The price to pay is that the by-need strategy is tricky to formulate and reason about. For instance, Wadsworth had to introduce a graph reduction in order to allow sharing of subterms, and the following developments on lazy evaluation essentially dealt with machines. The essence of call-by-need is summarized by Danvy et al. [16]:

Demand-driven computation & memoization of intermediate results

Linear head reduction		Call-by-need λ-calculus
$\Delta\ (I\ I)$		$\Delta\ (I\ I)$
$\equiv\ (\lambda x.\, x\ x)\ ((\lambda y.\, y)\ I)$		$\equiv\ (\lambda x.\, x\ x)\ ((\lambda y.\, y)\ I)$
$\to_{lh}\ (\lambda x.\, (\lambda y_0.\, y_0)\ I\ x)\ ((\lambda y.\, y)\ I)$		$\to_{\textsc{Deref}}\ (\lambda x.\, x\ x)\ ((\lambda y.\, I)\ I)$
$\to_{lh}\ (\lambda x.\, (\lambda y_0\, z_0.\, z_0)\ I\ x)\ ((\lambda y.\, y)\ I)$	$(\star)$	$\to_{\textsc{Assoc}}\ (\lambda y.\, (\lambda x.\, x\ x)\ I)\ I$
$\to_{lh}\ (\lambda x.\, (\lambda y_0\, z_0.\, x)\ I\ x)\ ((\lambda y.\, y)\ I)$		$\to_{\textsc{Deref}}\ (\lambda y.\, (\lambda x.\, (\lambda z.\, z)\ x)\ I)\ I$
$\to_{lh}\ (\lambda x.\, (\lambda y_0\, z_0.\, (\lambda y_1.\, y_1)\ I)\ I\ x)\ ((\lambda y.\, y)\ I)$		$\to_{\textsc{Deref}}\ (\lambda y.\, (\lambda x.\, (\lambda z.\, z)\ I)\ I)\ I$
$\to_{lh}\ (\lambda x.\, (\lambda y_0\, z_0.\, (\lambda y_1.\, \boxed{\lambda z_1.\, z_1})\ I)\ I\ x)\ ((\lambda y.\, y)\ I)$		$\to_{\textsc{Deref}}\ (\lambda y.\, (\lambda x.\, (\lambda z.\, \boxed{\lambda z_1.\, z_1})\ I)\ I)\ I$

Fig. 2. Example of a linear head reduction and a reduction in Ariola-Felleisen calculus.

Designing a proper calculus for call-by-need remained open for about two decades, until the mid-nineties when, in 1994, two very similar solutions to this problem were simultaneously presented by Ariola and Felleisen on the one hand, and Maraist, Odersky and Wadler on the other [9,8,22].

Ariola and Felleisen's calculus can be presented as follows:

Definition 1. *AF-calculus is defined by the following syntax:*

$$\begin{array}{lll}
\textit{Term} & t, u & ::= x \mid \lambda x.\, t \mid t\ u \\
\textit{Value} & v & ::= \lambda x.\, t \\
\textit{Answer} & A & ::= v \mid (\lambda x.\, A)\ t \\
\textit{Evaluation context} & E & ::= [\cdot] \mid E\ t \mid (\lambda x.\, E)\ t \mid (\lambda x.\, E[x])\ E
\end{array}$$

$$\begin{array}{lll}
(\textsc{Deref}) & (\lambda x.\, E[x])\ v & \to (\lambda x.\, E[v])\ v \\
(\textsc{Lift}) & (\lambda x.\, A)\ t\ u & \to (\lambda x.\, A\ u)\ t \\
(\textsc{Assoc}) & (\lambda x.\, E[x])\ ((\lambda y.\, A)\ t) & \to (\lambda y.\, (\lambda x.\, E[x])\ A)\ t
\end{array}$$

Intuitively, the above calculus shall be understood as follows:

- The lazy behaviour of the calculus is coded in the structure of contexts: term $E[x]$ evidences that variable x is in needed position in term $E[x]$.
- Rule Deref then gets the argument, in case it has already been computed and it has been detected as needed. In that case, the argument is substituted for one copy of the variable x, the one in needed position. As a consequence, the application is not erased and a single occurrence of the variable has been substituted. (E is a single-hole context.)
- Rules Lift and Assoc allow for the commutation of evaluation contexts in order for deref redexes to appear despite the persisting binders.

We gave an example of a reduction sequence in Ariola-Felleisen call-by-need λ-calculus in figure 2. In the last line we highlighted the term that would remain after applying the garbage-collection rule considered by Maraist et al [22]. Even though this is not part of the calculus, this convention of garbage-collecting weakening redexes is used in the rest of the paper to ease the reading of values.

Comparison Between LHR and Call-by-need. Figure 2 shows two reductions, on the left a LHR reduction and on the right a reduction in AF-calculus. There are striking common features:

– call-by-need can be seen as an optimization of both call-by-name and call-by-value while LHR can be seen as an optimization of head reduction;
– both rely on a linear, rather than destructive, substitution (at least in Ariola-Felleisen calculus presented above);
– more importantly, both share with call-by-name the same notion of convergence and the induced observational equivalences. Being observationally indistinguishable in the pure λ-calculus, they require instead side-effects to be told apart from call-by-name.

LHR made very scarce appearances in the literature for fifteen years, seemingly falling into oblivion except for the original authors. Yet, it made a surprise comeback by the beginning of the 2010's through a research line initiated by Accattoli and Kesner [4]. Their article describes the so-called structural λ-calculus, featuring explicit substitutions and at-distance reduction, taking once again inspiration from the computational behaviour of proof-nets and revamping the σ-equivalence relation in this framework. In their system, blocks of explicits substitutions are stuck where they were created and are considered transparent for all purposes but the rule of substitution of variables, contrasting sharply with the usual treatment of explicit substitutions. In practice, this is done by splitting β-reduction in multiplicative steps (corresponding to the creation of explicit substitutions) and exponential steps (corresponding to the effective substitution of a variable by some term). LHR naturally arises from the call-by-name flavour of the at-distance rules, and indeed the connection with the historical LHR is made explicit in many articles from the subsequent trend [6,4,2] and is furthemore used to obtain results ranging from computational complexity to factorization of rewriting theories of the λ-calculus [5,3].

While it took two decades for call-by-need to be equipped with a proper calculus, the way LHR is usually defined is intricate and inconvenient to work with. We actually view this fact, together with the observational indistinguishability, as one of the reasons for the almost complete nonexistence of LHR in literature until rediscovery by Accattoli and Kesner.

Towards a Logical and Classical "By-need" Calculus. The connections between the two formalisms are striking and actually not the least contingent. Understanding the precise relationships between the two may be useful to build call-by-need on firm, logical grounds. While comparing the merits of various call-by-need calculi in order to evidence one such calculus as being more canonical than the other may be quite dubious [2], when extending call-by-need with

[2] For instance Maraist, Odersky and Wadler calculus differ in their 1998 journal version from the calculus introduced by Ariola and Felleisen, but in no essential way since both calculi share the same standard reductions.

control we make call-by-name and call-by-need observational equivalences differ. In presence of first-class continuations, one can observe intensional behavior discriminating between call-by-name and call-by-need. Consider the term

```
let a = callcc (fun k ⇒ (true, fun x ⇒ throw k x)) in
let p = fst a in
let q = snd a in
if p then q (false, fun x ⇒ 0) else 99
```

In by-name, a is immediately substituted both in p and q, duplicating the callcc so that it reduces to 0. In by-need, a must be fully evaluated before being substituted, and the callcc is fired once. This forces the term to reduce to 99 instead. The impact of control is actually deeper: we can actually distinguish between several call-by-need calculi as evidenced by the second author in joint work with Ariola and Herbelin [10] about defining call-by-need extensions of $\overline{\lambda}\mu\tilde{\mu}$ which are sequent-style $\lambda\mu$-calculus [13].

In this context, it does make sense to wonder which calculus to pick and what observational impact these choices may have. We can summarize the aim of the present paper as integrating logically call-by-need and control operators. We take a different approach from that of Ariola *et al.* [10]: instead of starting from the sequent calculus which readily integrates control [13], we show how to transform systematically LHR into call-by-need and show that this derivation can be smoothly lifted to the case of the $\lambda\mu$-calculus.

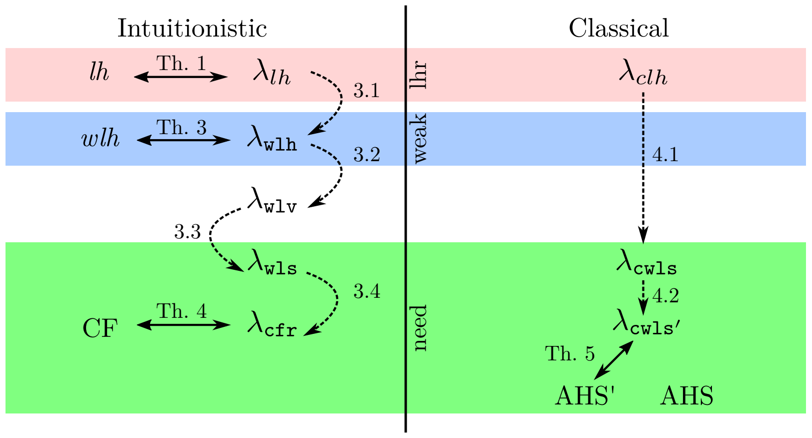
Contributions and Organization of the Paper. The contributions of the present paper are threefold.

- First, we reformulate LHR by introducing *closure contexts* and extend LHR to the $\lambda\mu$-calculus in Section 2.
- Then, after recalling Ariola-Felleisen's call-by-need calculus, we show in Section 3 how to derive a call-by-need calculus from LHR in three simple steps:
  1. restriction to a weak LHR by specializing closure contexts in 3.1,
  2. then enforcing memoization of intermediate results (by restricting to value passing) in 3.2;
  3. and finally implementing some sharing (thanks to closure contexts) in 3.3.

  We validate our constructions by comparing the resulting calculus with well-known call-by-need calculi, namely Ariola and Felleisen's or Chang and Felleisen's call-by-need. This justifies the following motto:

Lazy = Demand-driven comp.	+ Memoization	+ Sharing
(weak linear head reduction)	*(by value)*	*(closure sharing)*

- Third, we finally show in Section 4 how to lift the previous derivation to classical logic, that is from the λ-calculus to the $\lambda\mu$-calculus, synthesizing two classical by-need calculi, that is a call-by-need $\lambda\mu$-calculus, from classical LHR. The result is compared with Ariola et al call-by-need with control.



The whole picture is summarized in the above diagram. Plain arrows indicate some form of equivalence between two calculi with the corresponding theorem indicated. Dashed arrows indicate that a calculus is obtained from another by a small transformation, which is described in the section aside. Blocks indicate to which family of reduction a calculus pertains.

2 A Modern Linear Head Reduction

In the introduction, we informally introduced LHR. We now turn to the actual study of LHR, first recalling its historical presentation [15] and σ-equivalence and then giving a new formulation of the reduction based on *closure contexts*, that allows us to provide a classical variant seamlessly.

2.1 Historical presentation of linear head reduction

We first define Danos and Regnier's linear head reduction:

Definition 2. *The* **spine** *of a λ-term t is the set $\upharpoonleft t$ of left subterms of t, inductively defined as:*

$$\boxed{\frac{}{t \in \upharpoonleft t} \qquad \frac{r \in \upharpoonleft t}{r \in \upharpoonleft (t)\, u} \qquad \frac{r \in \upharpoonleft t}{r \in \upharpoonleft \lambda x.\, t}}$$

By construction, exactly one element of $\upharpoonleft t$ is a variable, written $\mathrm{hoc}\,(t)$*, for* **head occurrence***; it is the leftmost variable of t.*

Definition 3 (Head lambdas, Prime redexes). *Let t be a λ-term. Head lambdas, $\lambda_h(t)$, and prime redexes, $p(t)$, of t are defined by induction on t:*

$$\begin{aligned} \lambda_h(x) &\equiv \varepsilon & p(x) &\equiv \emptyset \\ \lambda_h(\lambda x.\, t) &\equiv x :: \lambda_h(t) & p(\lambda x.\, t) &\equiv p(t) \\ \lambda_h(t\ u) &\equiv \begin{cases} \varepsilon \text{ if } \lambda_h(t) = \varepsilon \\ \ell \text{ if } \lambda_h(t) = x :: \ell \end{cases} & p(t\ u) &\equiv \begin{cases} p(t) & \text{if } \lambda_h(t) = \varepsilon \\ p(t) \cup \{x \leftarrow u\} & \text{if } \lambda_h(t) = x :: \ell \end{cases} \end{aligned}$$

Remark 1. To understand head lambdas and prime redexes, it is convenient to consider blocks applications. We have indeed the following equalities:

$$\lambda_h((\lambda x.\, t)\, u\, \boldsymbol{r}) = \lambda_h((t)\, \boldsymbol{r}) \qquad\qquad p((\lambda x.\, t)\, u\, \boldsymbol{r}) = \{x \leftarrow u\} \cup p((t)\, \boldsymbol{r})$$

Head lambdas are precisely lambdas from the spine which will not be fed with arguments during head reduction. Now that we are equipped with the above notions, we can now formally define the linear head reduction:

Definition 4 (Linear head reduction). *Let u be a λ-term, let $x := \mathrm{hoc}\,(u)$. We say that u linear-head reduces to r, written $u \to_{lh} r$, when:*

1. *there exists some term t s.t. $\{x \leftarrow t\} \in p(u)$;*
2. *r is u where the variable occurrence $\mathrm{hoc}\,(u)$ has been substituted by t.*

Remark 2. Linear head reduction only substitutes one occurrence of a variable at a time and never destroys an application node. Likewise, it does not decrease the number of prime redexes. Thus terms keep growing, hence the name "linear" taken for linear *substitution*. An example of linear head reduction is given in Figure 2 where prime redexes are shown in grey boxes.

2.2 Reduction Up To σ-equivalence

It is noteworthy that LHR reduces terms which are not yet redexes for β_h, i.e. lh may get the argument of a binder even if it is not directly applied to it. The third reduction ($\star$) of the example from Figure 2 features such a cross-redex reduction. In this reduction, the λy_0 binder steps across the prime redex $\{z_0 \leftarrow x\}$ in order to recover its argument x. This kind of reduction would not have been allowed by the usual head reduction β_h. This peculiar behaviour can be made more formal thanks to a rewriting up to equivalence, also introduced by Regnier [25,26].

Definition 5 (σ-equivalence). *σ-equivalence is the reflexive, symmetric and transitive closure of the binary relation on λ-terms generated by:*

$$\begin{array}{ll} (\lambda x.\, t)\ u\ v \cong_\sigma (\lambda x.\, t\ v)\ u & \textit{with } x \textit{ fresh for } v \\ (\lambda x\, y.\, t)\ u \cong_\sigma \lambda y.\, (\lambda x.\, t)\ u & \textit{with } y \textit{ fresh for } u \end{array}$$

Intuitively, σ-equivalence allows reduction in a term where it would have been forbidden by other essentially transparent redexes.

Proposition 1. *If $t \cong_\sigma u$, then $p(t) = p(u)$.*

Proof. By case analysis on the rules of σ-equivalence.

The following proposition highlights the strong kinship relating LHR and σ-equivalence. Let us recall that a left context L is inductively defined by the following grammar:

$$L := [\cdot] \mid L\ t \mid \lambda x.\, L$$

Proposition 2. *If $t \rightarrow_{lh} r$ then there exist two left contexts L_1, L_2 such that*

$$t \cong_\sigma L_1[(\lambda x.\, L_2[x])\ u] \text{ and } r \cong_\sigma L_1[(\lambda x.\, L_2[u])\ u]$$

Proof. By induction on $p(t)$. Existence of L_1 follows from p being inductively defined over a left context, that of L_2 from the fact that the hoc is the leftmost variable.

The previous result can be slightly refined. The $\cong_\sigma$ relation is reversible, so that we can rebuild r by applying to $L_1[(\lambda x.\, L_2[u])\ u]$ the reverse σ-equivalence steps from the rewriting from t to $L_1[(\lambda x.\, L_2[x])\ u]$. We will not detail this operation here but rather move to the definition of closure contexts.

2.3 Closure Contexts and the λ_{lh}-calculus

With the aim to give a first-class status to the reduction up to σ-equivalence of proposition 2, we introduce *closure contexts*, which will allow to reformulate linear head reduction.

Definition 6 (Closure contexts). *Closure contexts are inductively defined as:*

$$\mathcal{C} := [\cdot] \mid \mathcal{C}_1[\lambda x.\, \mathcal{C}_2]\ t$$

Closure contexts feature all the required properties that provide them with a nice algebraic behaviour, that is, composability and factorization. Composition of contexts, $E_1[E_2]$, will be written $E_1 \circ E_2$ in the following.

Proposition 3 (Composition). *Let $\mathcal{C}_1$, $\mathcal{C}_2$ be closure contexts. $\mathcal{C}_1 \circ \mathcal{C}_2$ is a closure context.*

Proposition 4 (Factorization). *Any term t can be uniquely decomposed as a maximal closure context, in the usual meaning of composition, and a subterm t_0.*

Actually, we get even more: closure contexts precisely capture the notion of prime redex as asserted by the following proposition.

Proposition 5. *Let t be a term. Then $\{x \leftarrow u\} \in p(t)$ if and only if there exist a left context L, a closure context $\mathcal{C}$ and a term t_0 such that $t = L[\mathcal{C}[\lambda x.\, t_0]\ u]$.*

Proof. By induction on $p(t)$. The proof goes same way as for Proposition 2, except that we make explicit the context $\mathcal{C}$ instead of writing it as a σ-equivalence.

Owing to the fact that closure contexts capture prime redexes, we will provide an alternative and more conventional definition for the LHR. It will result in the λ_{lh}-calculus, based on contexts rather than ad-hoc variable manipulations.

Definition 7 (λ_{lh}-calculus). *The λ_{lh}-calculus is defined by the reduction rule:*

$$L_1[\mathcal{C}[\lambda x.\, L_2[x]]\ u] \rightarrow_{\lambda_{lh}} L_1[\mathcal{C}[\lambda x.\, L_2[u]]\ u]$$

$$\begin{array}{lll@{\quad}lll}
\text{Closures} & c & ::= (t,\sigma) & \textsc{Push} & \langle (t\,u,\sigma) \mid \pi\rangle & \rightarrow \langle (t,\sigma) \mid (u,\sigma)\cdot\pi\rangle \\
\text{Environments} & \sigma & ::= \emptyset \mid \sigma + (x := c) & \textsc{Pop} & \langle (\lambda x.\, t,\sigma) \mid c\cdot\pi\rangle & \rightarrow \langle (t,\sigma + (x := c)) \mid \pi\rangle \\
\text{Stacks} & \pi & ::= \varepsilon \mid c\cdot\pi & \textsc{Grab} & \langle (x,\sigma + (x := c)) \mid \pi\rangle & \rightarrow \langle c \mid \pi\rangle \\
\text{Processes} & p & ::= \langle c \mid \pi\rangle & \textsc{Garbage} & \langle (x,\sigma + (y := c)) \mid \pi\rangle & \rightarrow \langle (x,\sigma) \mid \pi\rangle
\end{array}$$

Fig. 3. Krivine abstract machine

where L_1, L_2 are left contexts, $\mathcal{C}$ is a closure context, t and u are λ-terms, with the usual freshness conditions to prevent variable capture in u.

Proposition 6 (Stability of λ_{lh} under σ). *Let t, u and v be terms such that $t \cong_\sigma u \rightarrow_{lh} v$, then there is w such that $t \rightarrow_{lh} w \cong_\sigma v$.*

Proof. By induction over the starting σ-equivalence, and case analysis of the possible interactions between contexts. For instance, if the λ-abstraction of the rule interacts with $\mathcal{C}$ through the second generator of the σ-equivalence, this amounts to transfer a fragment from $\mathcal{C}$ into L_2 which is transparent for the reduction rule. Similar interactions may appear at context boundaries or inside contexts.

Theorem 1. *The λ_{lh}-calculus captures the linear head reduction.*

$$t \rightarrow_{\lambda_{lh}} r \qquad \textit{iff} \qquad t \rightarrow_{lh} r.$$

Proof. Indeed, the x from the rule is precisely the hoc of the term since closure contexts are in particular left contexts, and because we are reducing up to closure contexts, Proposition 5 ensures that $\{x \leftarrow u\}$ is a prime redex.

2.4 Closure Contexts and the KAM: a Strong Relationship

Remarkably enough, closure contexts are not totally coming out of the blue. They are indeed already present in Chang-Felleisen call-by-need calculus [12], even if their intrinsic interest, their properties as well as the natural notion of LHR stemming from them were not made explicit. Maybe the main contribution of our work is to put them at work as a design principle.

Closure contexts are morally transparent for some well-behaved head reduction: one can consider that $(\mathcal{C}[\lambda x.\,[\cdot]])\,t$ is a context that only adds a binding $(x := t)$ to the environment, as well as the bindings contained in $\mathcal{C}$. This intuition can be made formal thanks to the Krivine abstract machine (KAM), recalled in Figure 3. As stated by the following result, transitions Push and Pop of the KAM implement the computation of closure contexts.

Proposition 7. *Let t be a term, σ an environment, π a stack and $\mathcal{C}$ a closure context. We have the following reduction*

$$\langle (\mathcal{C}[t],\sigma) \mid \pi\rangle \longrightarrow^*_{\textsc{Push},\,\textsc{Pop}} \langle (t,\sigma + [\mathcal{C}]_\sigma) \mid \pi\rangle$$

where $[\mathcal{C}]_\sigma$ is defined by induction over $\mathcal{C}$ as follows:

$$[[\cdot]]_\sigma \equiv \emptyset \quad [\mathcal{C}_1[\lambda x.\, \mathcal{C}_2]\ t]_\sigma \equiv [\mathcal{C}_1]_\sigma + (x := (t, \sigma)) + [\mathcal{C}_2]_{\sigma + [\mathcal{C}_1]_\sigma + (x := (t,\sigma))}$$

Conversely, for all t_0 and σ_0 such that

$$\langle (t, \sigma) \mid \pi \rangle \longrightarrow^*_{\textsc{Push},\,\textsc{Pop}} \langle (t_0, \sigma_0) \mid \pi \rangle$$

there exists $\mathcal{C}_0$ such that $t = \mathcal{C}_0[t_0]$, where $\mathcal{C}_0$ is inductively defined over σ_0.

Proof. The first property is given by a direct induction on $\mathcal{C}$, while the second is done by induction on the KAM reduction.

Actually, the KAM can even be seen as an implementation of a (weak) LHR rather than the (weak) head reduction. Indeed, the substitution is delayed in practice until a variable appears in head position, i.e. when it is the hoc of a term. Such a phenomenon was formalized by Danos and Regnier [15] who proved that the sequence of substitutions from the LHR and the sequence of closures substituted by the Grab rule are the same.

2.5 Classical Linear Head Reduction

Thanks to the intuition provided by the closure contexts, we propose here a classical variant of the linear head calculus, presented in the $\lambda\mu$-calculus [24]. The additional binder μ quantifies over a special class of variable called *stack* variables. It allows to capture and reinstate the stack at will. Expressions of the form $[\alpha]\, t$ are called *processes* or *commands*. We recall the syntax and (call-by-name) reduction below.

$$t, u ::= x \mid \lambda x.\, t \mid t\ u \mid \mu\alpha.\, c$$

$$c ::= [\beta]\, t$$

$$\begin{array}{lcr} (\lambda x.\, t)\ u & \rightarrow & t\{x := u\} \\ (\mu\alpha.\, c)\ u & \rightarrow & \mu\alpha.\, c\{[\alpha]\, r := [\alpha]\, r\ u\} \\ [\alpha]\, \mu\beta.\, c & \rightarrow & c\{\beta := \alpha\} \end{array}$$

We will only be interested in the reduction of commands in the remainder of this section. Our calculus is a direct elaboration of the aforementioned linear head calculus, and we dedicate this section to its thorough description.

Definition 8 (Classical LHR). *We define left stack contexts K by induction, and then a classical extension of left contexts and closure contexts as follows.*

$$\begin{aligned} K &::= [\cdot] \mid [\alpha]\, L[\mu\beta.\, K] \\ \mathcal{C} &::= [\cdot] \mid \mathcal{C}_1[\lambda x.\, \mathcal{C}_2]\ t \mid \mathcal{C}_1[\mu\alpha.\, K[[\alpha]\, \mathcal{C}_2]] \\ L &::= [\cdot] \mid \lambda x.\, L \mid L\ t \mid \mu\beta.\, [\alpha]\, L \end{aligned}$$

The classical linear head calculus is then defined by the following reduction.

$$\begin{array}{lll} \sigma ::= \cdots \mid \sigma + (\alpha := \pi) & \textsc{Save} & \langle (\mu\alpha.c, \sigma) \mid \pi \rangle \rightarrow \langle (c, \sigma + (\alpha := \pi)) \mid \varepsilon \rangle \\ \pi ::= \cdots \mid (\alpha, \sigma) & \textsc{Restore} & \langle ([\alpha]t, \sigma) \mid \varepsilon \rangle \ \ \rightarrow \langle (t, \sigma) \mid \sigma(\alpha) \rangle \end{array}$$

Fig. 4. μ-KAM

$$[\alpha]\, L_1[\mathcal{C}[\lambda x.\, L_2[x]]\ t] \rightarrow_{\lambda_{clh}} [\alpha]\, L_1[\mathcal{C}[\lambda x.\, L_2[t]]\ t]$$

We then adapt the σ-equivalence to the classical setting following Laurent [21].

Definition 9 (Classical σ-equivalence). *The σ-equivalence is extended to the $\lambda\mu$-calculus with the following generators.*

$$\begin{array}{rll} (\lambda x.\, \mu\alpha.\, [\beta]\, t)\ u \cong_\sigma \mu\alpha.\, [\beta]\, (\lambda x.\, t)\ u & & \textit{with } \alpha \notin u \\ [\alpha]\, (\mu\beta.\, [\gamma]\, (\mu\delta.\, c)\ u)\ t \cong_\sigma [\gamma]\, (\mu\delta.\, [\alpha]\, (\mu\beta.\, c)\ t)\ u & & \textit{with } \beta \notin u,\ \delta \notin t \\ [\alpha]\, \lambda x.\, \mu\beta.\, [\gamma]\, \lambda y.\, \mu\delta.\, c \cong_\sigma [\gamma]\, \lambda y.\, \mu\delta.\, [\alpha]\, \lambda x.\, \mu\beta.\, c & & \\ [\alpha]\, (\mu\beta.\, [\gamma]\, \lambda x.\, \mu\delta.\, c)\ t \cong_\sigma [\gamma]\, \lambda x.\, \mu\delta.\, [\alpha]\, (\mu\beta.\, c)\ t & & \textit{with } x \notin t,\ \beta \notin t \end{array}$$

Proposition 8 (Stability under σ). *Let t, u and v be terms such that $t \cong_\sigma u \rightarrow_{\lambda_{clh}} v$, then there is w such that $t \rightarrow_{\lambda_{clh}} w \cong_\sigma v$.*

We now relate the classical LHR calculus with the classical extension to KAM [21,29] given in Figure 4 where only the modifications with respect to Figure 3 are shown.

Danos and Regnier obtain a simulation theorem relating the KAM with LHR by defining substitution sequences [15]. This can be lifted to the $\lambda\mu$-calculus: there is a simulation theorem relating the μ-KAM with the classical LHR. In order to state theorem 2 which concludes this section, one first needs to introduce some preliminary definitions which are motivated by the following remark.

Remark 3. The λ_{clh} reduction rule is actually abusive. Recall that in its definition, t can be any term. This means that to ensure later capture-free substitution by preserving the Barendregt condition on terms, one has to rename the variables of t on the fly, so that the legitimate rule would rather be

$$[\alpha]\, L_1[\mathcal{C}[\lambda x.\, L_2[x]]\ t] \rightarrow_{\lambda_{clh}} [\alpha]\, L_1[\mathcal{C}[\lambda x.\, L_2[\#\, t]]\ t]$$

where $\#\, t$ stands for t where bound variables have been replaced by fresh variable instances.

We need to define properly the relation between the original and the substituted terms in the above rule.

Definition 10 (One-step residual). *In the above rule, we say that t is the residual of $\#t$ in the source term.*

It turns out that this definition can be extended to a reduction of arbitrary length thanks to the following lemma.

Proposition 9. *For any reduction of the form*

$$[\alpha]\, t \to_{\lambda_{clh}}{}^{*} [\alpha]\, L_1[\mathcal{C}[\lambda x.\, L_2[x]]\ r_0] \to_{\lambda_{clh}} [\alpha]\, L_1[\mathcal{C}[\lambda x.\, L_2[\# r_0]]\ r_0]$$

there exists a subterm r of t such that $r_0 \equiv \# r$, the residual *of r_0 in t.*

Proof. By induction on the reduction. The key point is that all along the reduction, all terms on the right of an application node are subterms of the original term, up to some variable renaming. The original subterm can then be traced back by jumping up into the term being substituted at each step.

Definition 11 (Substitution sequence). *Given two terms t and t_0 s.t. $t_0 \to^*_{\lambda_{clh}} t$, we define the substitution sequence of t w.r.t. t_0 as the (possibly infinite) sequence $\mathfrak{S}_{t_0}(t)$ of subterms of t_0 defined as follows, where α is a fresh stack variable.*

- *If $[\alpha]\, t \not\to_{\lambda_{clh}}$ then $\mathfrak{S}_{t_0}(t) ::= \emptyset$.*
- *If $[\alpha]\, t \equiv [\alpha]\, L_1[\mathcal{C}[\lambda x.\, L_2[x]]\ r] \to_{\lambda_{clh}} [\alpha]\, t'$ then $\mathfrak{S}_{t_0}(t) ::= r_0 :: \mathfrak{S}_{t_0}(t')$ where r_0 is the residual of r in t_0.*

We finally pose $\mathfrak{S}(t) ::= \mathfrak{S}_t(t)$.

The μ-KAM naturally features a similar behaviour w.r.t. residuals.

Proposition 10. *If $\langle (t, \cdot) \mid \varepsilon \rangle \to^* \langle (t_0, \sigma) \mid \pi \rangle$ then t_0 is a subterm of t.*

Proof. By a straightforward induction over the reduction path.

This proposition can (and actually needs to) be generalized to any source process whose stacks and closures only contain subterms of t. This leads to the definition of a similar notion of substitution sequence for the KAM.

Definition 12 (KAM substitution sequence). *For any term t, we define the KAM substitution sequence of a process p as the possibly infinite sequence of terms $\mathfrak{K}(p)$ defined as:*

- *If $p \not\to$ then $\mathfrak{K}(p) ::= \emptyset$.*
- *If $p \equiv \langle (x, \sigma) \mid \pi \rangle \to \langle (t, \tau) \mid \pi \rangle$ then $\mathfrak{K}(p) ::= t :: \mathfrak{K}(\langle (t, \tau) \mid \pi \rangle)$.*
- *Otherwise if $p \to q$ then $\mathfrak{K}(p) ::= \mathfrak{K}(q)$.*

Finally, the KAM substitution sequence of any term t is defined as $\mathfrak{K}(t) ::= \mathfrak{K}(\langle (t, \cdot) \mid \varepsilon \rangle)$.

By the previous lemma, $\mathfrak{K}(t)$ is a sequence of subterms of t. We can therefore formally relate it to $\mathfrak{S}(t)$.

Proposition 11. *Let t be a term. Then $\mathfrak{K}(t)$ is a prefix of $\mathfrak{S}(t)$.*

Proof. By coinduction, for each step of $\mathfrak{S}(t)$, it is sufficient either to construct a matching step in $\mathfrak{K}(t)$ or to stop. Let us assume that

$$[\alpha]\, t \equiv [\alpha]\, L_1[\mathcal{C}[\lambda x.\, L_2[x]]\ r_0] \to_{\lambda_{clh}} [\alpha]\, L_1[\mathcal{C}[\lambda x.\, L_2[\# r_0]]\ r_0] \equiv [\alpha]\, t_r$$

There are now two cases, depending on the KAM reduction of $\mathfrak{K}(\langle(t,\cdot)\mid\varepsilon\rangle)$. By a simple generalization of Lemma 7, the normal form of this process in the Grab-free fragment of the KAM rules can be one of the two following form:

- either $\langle(x,\sigma+(x:=(r_0,\tau))\mid\pi\rangle$ for some σ, τ and π
- or a blocked state of the KAM for all rules

The second case can occur if there are too many λ-abstractions in the left contexts of the above reduction rule or if there is an free stack variable appearing in a command part of the left contexts. In this case $\mathfrak{K}(t)=\emptyset$, which is indeed a prefix of $\mathfrak{S}(t)$.

Otherwise, one has $\mathfrak{K}(t)=r_0::\mathfrak{K}(\langle(r_0,\tau)\mid\pi\rangle)$ and $\mathfrak{S}(t)=r_0::\mathfrak{S}_t(t_r)$. It it therefore sufficient to show that the property holds for the tail of those two sequences.

It is a noteworthy fact that, when we put t_r in the KAM, we obtain a reduction of the form

$$\langle(t_r,\cdot)\mid\varepsilon\rangle\to^*\langle(\#\,r_0,\sigma+(x:=(r_0,\tau))+\sigma_0)\mid\pi\rangle$$

for some σ_0, where the Grab rule does not appear. This reduction follows indeed the very same transitions as the process made of the source term. Moreover, a careful inductive analysis of the possible transitions shows that $\sigma=\tau+\sigma_1$ for some σ_1, where the variables bound by σ_1 are not free in $\#\,r_0$. Therefore,

$$\mathfrak{K}(t_r)=\mathfrak{K}(\langle(\#\,r_0,\sigma+(x:=(r_0,\tau))+\sigma_0)\mid\pi\rangle)=\mathfrak{K}(\langle(\#\,r_0,\tau)\mid\pi\rangle)$$

because the KAM reduction is not affected by extension of closure environements with variables absent from the closure term. By applying the coinduction hypothesis, we immediatly obtain than $\mathfrak{K}(t_r)$ is a prefix of $\mathfrak{S}(t_r)$.

But now, we can conclude, because $\mathfrak{K}(\langle(\#\,r_0,\tau)\mid\pi\rangle)$ and $\mathfrak{K}(\langle(r_0,\tau)\mid\pi\rangle)$ (resp. $\mathfrak{S}(t_r)$ and $\mathfrak{S}_t(t_r)$) are the same sequence up to a renaming of the bound variables coming from $\#\,r_0$ which is common to both kinds of reduction. Thus $\mathfrak{K}(\langle(r_0,\tau)\mid\pi\rangle)$ is a prefix of $\mathfrak{S}_t(t_r)$ and we are done.

The following theorem is a direct corollary of Proposition 11.

Theorem 2. *Let $c_1\to_{\lambda_{clh}}c_2$ where $c_1:=[\alpha]\,L_1[\mathcal{C}[\lambda x.\,L_2[x]]\;t]$, then the substitution sequence of process c_1 is either empty or of the form $t::\ell$ where ℓ is the substitution sequence of process c_2.*

Proposition 8 and theorem 2 validate our calculus as a sound classical extension of LHR.

3 Towards Call-by-need

Our journey from LHR to call-by-need will now follow three steps: first restricting LHR to a weak reduction, imposing a value-restriction and finally adding an amount of sharing.

3.1 Weak Linear Head Reduction

The LHR as given at paragraph 2.3 is a *strong* reduction: it reduces under abstractions. We now adapt λ_{lh}-calculus to the weak case. It is easy to give a weak version of the reduction in the historical LHR, which inherits the same defects as its strong counterpart.

Definition 13. *(Historical wlh-reduction) We say that t weak-linear-head reduces to r, written $t \to_{wlh} r$, iff $t \to_{lh} r$ and t does not have any head λ.*

On the other hand, the λ_{lh} reduction can be denied the possibility to reduce under abstractions by restricting the evaluation contexts inside which it can be triggered. This requires some care though. Indeed, the contexts may contain λ-abstractions, assuming they have been compensated by as many previous applications. That is, those binders must pertain to a prime redex as in $(\lambda z_1 \ldots z_n\, x.\, E^w[x])\, r_1 \ldots\, r_n\, u$. Plain closure contexts are not expressive enough to capture this situation.

To solve this issue, we extend the λ-calculus in a way which is inspired both by techniques used for studying reductions, residuals and developments in λ-calculus [11] and by λ`let`-calculi [8] or explicit substitutions [1], and in particular the structural λ-calculus [4]. We are indeed going to recognize when a prime redex has been created by marking them explicitly. Yet, contrarily to standard λ-calculus rewriting theory, we will mark *lambda*s which are not necessarily actually involved in β-redexes and contrarily to λ`let`-calculi or the structural λ-calculus, we will preserve the underlying structure of the λ-term by only marking abstractions rather than creating `let`-bindings and making them transparent for all rules. We shall discuss the significance of this design choice in section 3.5.

Definition 14 (Marked λ-calculus). *The marked λ-calculus is thus inductively defined as*

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

where $\ell x.\, t$ is a marked version of $\lambda x.\, t$. For any marked λ-term t, we will write $[t]$ for the usual λ-term obtained from t by unmarking all abstractions. Likewise, σ-equivalence is adapted in a straightforward fashion.

We have to update the definition of closure contexts to fit into this presentation. It is actually enough to restrict all abstractions appearing inside a closure context to marked abstractions.

Definition 15 (Closure contexts). *From now on, closure contexts $\mathcal{C}$ will be defined by the inductive grammar below.*

$$\mathcal{C} ::= [\cdot] \mid \mathcal{C}_1[\ell x.\, \mathcal{C}_2]\ t$$

In general, arbitrary marked terms do not make sense, because marked abstractions may not have a matching application. This is why we define a notion of well-formed marked terms.

Definition 16 (Well-formed marked terms). *A marked term t is well-formed whenever for any decomposition $t \equiv E[\ell x.\, u]$ where E is an arbitrary context, E can be further decomposed as $E \equiv E_0[\mathcal{C}\ r]$ where E_0 is an arbitrary context, $\mathcal{C}$ a marked closure context and r a marked term.*

In the rest of the paper, we will only work with *well-formed* marked terms even when this remains implicit: all our constructions and reductions will preserve well-formedness as in the following definition:

Definition 17 ($\lambda_{\mathtt{wlh}}$-calculus). *The weak linear head calculus $\lambda_{\mathtt{wlh}}$ is defined by the rules:*

$$\begin{aligned} \mathcal{C}[\lambda x.\, t]\ u \quad &\rightarrow_{\lambda_{\mathtt{wlh}}} \mathcal{C}[\ell x.\, t]\ u \\ \mathcal{C}[\ell x.\, E^w[x]]\ u &\rightarrow_{\lambda_{\mathtt{wlh}}} \mathcal{C}[\ell x.\, E^w[u]]\ u \end{aligned}$$

together with compatibility with E^w contexts, inductively defined as follows.

$$E^w ::= [\cdot] \mid E^w\ t \mid \ell x.\, E^w.$$

Proposition 12 (Stability of λ_{wlh} by σ). *Let t, u and v be terms such that $t \cong_\sigma u \rightarrow_{\lambda_{wlh}} v$, then there is w such that $t \rightarrow_{\lambda_{wlh}} w \cong_\sigma v$.*

This property is proved similarly to Proposition 6. We can now prove that λ_{wlh} and the historical *wlh*-reduction coincide:

Theorem 3. *$t \rightarrow_{\lambda_{wlh}} r$ iff $t \rightarrow_{wlh} r$.*

Proof. They correspond since not having a head lambda is exactly equivalent to having all its subcontexts starting with an abstraction marked.

3.2 Call-by-"value" Linear Head Reduction

In order to obtain a call-by-value LHR, we will restrict contexts that trigger substitutions to react only in front of a value. In addition, the up-to-closure paradigm used so far will also incite us to consider values up to closures defined as $W ::= \mathcal{C}[V]$ when $V ::= \lambda x.\, t$ stands for values.

Going from the usual call-by-name to the usual call-by-value is then simply a matter of adding a context forcing values. Likewise, we just add a context forcing up-to values. This construction is made in a systematic way according to the standard call-by-value encoding.

Definition 18 (Call-by-value contexts). *We define the call-by-value contexts inductively as follows.*

$$E^v ::= [\cdot] \mid E^v\ t \mid \ell x.\, E^v \mid \mathcal{C}[\ell x.\, E^v_1[x]]\ E^v_2$$

The call-by-value weak linear head reduction is obtained straightforwardly.

Definition 19 ($\lambda_{\mathtt{wlv}}$-calculus). *The $\lambda_{\mathtt{wlv}}$-calculus is the calculus defined by the E^v-compatible closure of following rules.*

$$\begin{aligned}
\mathcal{C}[\lambda x.\, t]\ u &\rightarrow_{\lambda_{\mathtt{wlv}}} \mathcal{C}[\ell x.\, t]\ u \\
\mathcal{C}[\ell x.\, E^v[x]]\ W &\rightarrow_{\lambda_{\mathtt{wlv}}} \mathcal{C}[\ell x.\, E^v[W]]\ W
\end{aligned}$$

It is easy to check that the reduction was not deeply modified, the difference lies in the clever choice of contexts. Stability by σ is proved as in Proposition 6.

Proposition 13 (Stability of $\lambda_{\mathtt{wlv}}$ by σ). *Let t, u and v be terms such that $t \cong_\sigma u \rightarrow_{\lambda_{\mathtt{wlv}}} v$, then there is w such that $t \rightarrow_{\lambda_{\mathtt{wlv}}} w \cong_\sigma v$.*

Although we branded this calculus as a call-by-value one, it already implements a call-by-need strategy since it triggers the reduction of an argument if and only if it was made necessary by the encounter of a corresponding variable in (call-by-value) hoc position. We give the reduction on our running example:

$$\begin{aligned}
\Delta\ (I\ I) \ \equiv\ & (\lambda x.\, x\ x)\ ((\lambda y.\, y)\ I) \\
\rightarrow_{\mathtt{wlv}}\ & (\ell x.\, x\ x)\ ((\ell y.\, y)\ I) \\
\rightarrow_{\mathtt{wlv}}\ & (\ell x.\, x\ x)\ ((\ell y.\, y)\ I) \\
\rightarrow_{\mathtt{wlv}}\ & (\ell x.\, x\ x)\ ((\ell y.\, I)\ I) \\
\rightarrow_{\mathtt{wlv}}\ & (\ell x.\, (\ell y_1.\, \lambda z_1.\, z_1)\ I\ x)\ ((\ell y.\, I)\ I) \\
\rightarrow_{\mathtt{wlv}}\ & (\ell x.\, (\ell y_1\, z_1.\, z_1)\ I\ x)\ ((\ell y.\, I)\ I) \\
\rightarrow_{\mathtt{wlv}}\ & (\ell x.\, (\ell y_1\, z_1.\, z_1)\ I\ ((\ell y.\, I)\ I))\ ((\ell y.\, I)\ I) \\
\rightarrow_{\mathtt{wlv}}\ & (\ell x.\, (\ell y_1\, z_1.\, (\ell y_2.\, \boxed{I})\ I)\ I\ ((\ell y.\, I)\ I))\ ((\ell y.\, I)\ I)
\end{aligned}$$

In the first transition, the reduction occurs in the argument required by x, returning a value (up to closure) that will then be substituted.

3.3 Closure Sharing

The $\lambda_{\mathtt{wlv}}$-calculus does not realize call-by-need reduction schemes from the literature [8]. The above example reveals a duplication of computation:

$$\mathcal{C}'[\ell x.\, E^v[x]]\ \mathcal{C}[v] \rightarrow_{\lambda_{\mathtt{wlv}}} \mathcal{C}'[\ell x.\, E^v[\mathcal{C}[v]]]\ \mathcal{C}[v]$$

In that case, $\mathcal{C}$ is copied, which will end up in recomputing its bound terms if ever they are going to be used throughout the reduction. While our running example $\Delta\ (I\ I)$ does not feature such a behaviour, this can instead be seen on $(\lambda x.\, x\ I\ x)\ (\lambda y\, z.\, y)\ (I\ I)$ because while $(\lambda y\, z.\, y)\ (I\ I)$ is already an up-to value, it also uses the argument $I\ I$ from its closure. During the substitution, this subterm is copied as-is, resulting in its recomputation at each call to x in the body of the abstraction.

It is possible to solve this issue in an elegant way akin to the Assoc rule of Ariola-Felleisen calculus. This is achieved by the extrusion of the closure of the value at the instant it is substituted. There is no need to refine contexts further, because everything is already in order. We obtain the calculus below:

Definition 20. *The call-by-value LHR with sharing is defined by the rules:*

$$\begin{aligned}
\mathcal{C}[\lambda x.\, t]\ u &\rightarrow_{\lambda_{\mathtt{wls}}} \mathcal{C}[\ell x.\, t]\ u \\
\mathcal{C}'[\ell x.\, E^v[x]]\ \mathcal{C}[v] &\rightarrow_{\lambda_{\mathtt{wls}}} \mathcal{C}[\mathcal{C}'[\ell x.\, E^v[v]]\ v]
\end{aligned}$$

with the usual freshness conditions to prevent variable capture in $\mathcal{C}'$.

Values	v	$::= \lambda x.\, t$
Answers	a	$::= A[v]$
Answer contexts	A	$::= [\cdot] \mid A_1[\lambda x.\, A_2]\, u$
Inner answer contexts	A^λ	$::= [\cdot] \mid A[\lambda x.\, A^\lambda]$
Outer answer contexts	$A^@$	$::= [\cdot] \mid A[A^@]\, u$
Contexts	E	$::= [\cdot] \mid E\, u \mid A[E]$
		$\mid A^@[A[\lambda x.\, A^\lambda[E[x]]]\, E]$
		where $A^@[A^\lambda] \in A$

$$A^@[A_1[\lambda x.\, A^\lambda[E[x]]]\, A_2[v]] \longrightarrow A^@[A_1[A_2[(A^\lambda[E[x]])\{x := v\}]]]$$
$$\text{if } A^@[A^\lambda] \in A \qquad (\beta_{\mathrm{cf}})$$

Fig. 5. Chang and Felleisen's call-by-need calculus

3.4 $\lambda_{\mathtt{wls}}$ is a Call-by-need Calculus

Remarkably enough, the resulting calculus is almost exactly Chang and Felleisen's call-by-need calculus [12] (CF-calculus) which is presented in figure 5. The main difference lies in the fact that the latter features the usual destructive substitution, while ours is linear. One can convince oneself that their answer contexts correspond to our closure contexts, while the balancing of inner and outer contexts is transparent in our calculus thanks to the restriction to marked terms. There is a reduction mismatch though, which is that CF-calculus plugs closures in the reverse order compared to $\lambda_{\mathtt{wls}}$. The reduction β_{cfr} (see below) can be described in our formalism in a straightforward manner.

Definition 21 (CF-calculus revisited).
The marked CF-calculus is given by the E_v-compatible closure of the rules below.

$$\begin{array}{ll} \mathcal{C}[\lambda x.\, t]\ u & \to_{\lambda_{\mathtt{cfr}}} \mathcal{C}[\ell x.\, t]\ u \\ \mathcal{C}'[\ell x.\, E^v[x]]\ \mathcal{C}[v] & \to_{\lambda_{\mathtt{cfr}}} \mathcal{C}'[\mathcal{C}[E^v[x]\{x := v\}]] \end{array}$$

Theorem 4. *For any well-formed marked terms t and r, if $t \to^*_{\lambda_{\mathtt{cfr}}} r$ then $[t] \to^*_{\beta_{cf}} [r]$ where the length of the second reduction is the number of uses of the second rule in the first reduction. Conversely, for any unmarked terms t and r s.t. $t \to_{\beta_{cf}} r$ there exist markings t' and r' of t and r where $t' \to^*_{\lambda_{\mathtt{cfr}}} r'$ using exactly once the second rule.*

Proof. It is sufficient to observe that well-formedness in the marked calculus is equivalent to the existence of a decomposition into inner and outer contexts that are balanced in the unmarked calculus.

The difference in the order of closure plugging may seem irrelevant in Chang and Felleisen's framework because they use non-linear destructive substitutions and both orders are possible: an ad-hoc choice was made there. On the contrary, our design – strongly guided by logic – directly led us to a plugging order compatible with linear substitution.

3.5 Comparison with other call-by-need calculi.

CF-calculus is a bit peculiar in works on call-by-need. It would be better to also compare our approach to more standard calculi such as AF-calculus. Moreover, a third variant of call-by-need calculus has been defined recently by considering the linear substitution λ-calculus [2] (LS-calculus), and it turns out to be very close to our presentation. This section is devoted to the comparison of $\lambda_{\mathtt{wls}}$ with those calculi.

The major source of difference between the three calculi lies in the handling and encoding of term bindings.

– AF-calculus uses a microscopic reduction (simple, small steps) and relies on rewriting rules to build up flat binding contexts;
– LS-calculus uses a macroscopic reduction ("at distance", relying on a very-elaborated and structured context) and enforces reduction rules to be transparent w.r.t. binding contexts;
– $\lambda_{\mathtt{wls}}$ does the same thing but uses a refined version of explicit substitutions embodied by closure contexts.

We now explain to which extent these calculi are essentially the same except for the technology used for the implementation of binding contexts. Both for AF-calculus and LS-calculus, it would be natural to switch to a calculus featuring `let`-binders, or equivalently explicit substitutions [1]. Yet, we will describe them in the usual λ-calculus for uniformity with the rest of the paper and because marked λ-abstractions are not needed in this case.

Micro-steps linear head reduction In order to compare our approach to AF-calculus, we now propose another approach to linear head reduction which starts from the rules of σ-equivalence and integrate them in the LHR calculus. It will serve to rearrange the term in order to create appropriate β-redexes: as a consequence, we will not work with redexes up to closure context (or equivalently up to σ-equivalence) but will have reductions dedicated to the creation of the linear head redex. Still, while rearranging terms, we do not want to keep the β-redexes which are not triggered by a hoc-variable yet; a restricted form of closure context will remain, which would naturally be expressed with `let`-bindings or explicit substitutions:

$$\mathcal{L} ::= [\cdot] \mid (\lambda x.\, \mathcal{L})\, t$$

Because of the slight mismatch between usual closure contexts and restricted context closures, there is a choice to be made in the microscopic reduction rule. While the original closure contexts made prime redexes appear naturally, now we need to explicitly create β-redexes by making the potential redexes commute with the surrounding context. There are two ways to do so, each one corresponding to a generating rule of the σ-equivalence, either by pushing the application node inside closure contexts (Lift) or by extruding applied λ-abstractions from the surrounding context (Dig):

$$((\lambda x.\, \mathcal{L}[\lambda y.\, t])\, u)\, v \to (\lambda x.\, (\mathcal{L}[\lambda y.\, t])\, v)\, u \quad (\textsc{Lift})$$
$$(\mathcal{L}[(\lambda x.\, \lambda y.\, t)\, u])\, v \to (\mathcal{L}[\lambda y.\, (\lambda x.\, t)\, u])\, v \quad (\textsc{Dig})$$

These rules admit extensions, taking opportunity of the structure of restricted closure contexts:

$$(\mathcal{L}[\lambda y.\, t])\, v \to \mathcal{L}[(\lambda y.\, t)\, v] \quad (\textsc{Lift}\star)$$
$$(\mathcal{L}[\lambda y.\, t])\, v \to (\lambda y.\, \mathcal{L}[t])\, v \quad (\textsc{Dig}\star)$$

Reduction (Lift) is the most common in literature, probably because it is easier to formulate without a clear notion of closure contexts. Equipped with (Lift), we actually obtain a calculus which precisely (up to the use of explicit `lets`) to the calculus considered by Danvy *et al.* in the preliminary section of [16], where answers A are no more than terms of the form $\mathcal{L}[\lambda x.\, t]$. Reduction (Dig) is an alternative choice. The two reductions are not only different, but also incompatible, in the sense that their left-hand sides are the same while their right-hand sides are not convertible, thus breaking confluence. Yet, the reduced terms still agree up to σ-equivalence by construction. Finally, the last rule performs the linear substitution:

$$(\lambda x.\, E[x])\, t \to (\lambda x.\, E[t])\, t$$

Here, $E[x]$ represents a left context: $E ::= [\cdot] \mid E\, t \mid \lambda x.\, E$ so that the substitution only replaces exactly one variable.

Any choice between rules (Lift) or (Dig) will lead to call-by-need calculi. Still we consider it is more interesting to opt for (Lift) since it allows us to recover known calculi. From the microscopic linear head calculus with (Lift), we can apply the same three transformations as in the macroscopic case:

1. weak reduction constrain evaluation contexts to be applicative contexts up to closures: $E ::= [\cdot] \mid E\, t \mid (\lambda x.\, E)\, t$
2. restriction to value (up to closure) substitutions, which creates new call-by-value, evaluation contexts: $E ::= \cdots \mid (\lambda x.\, E[x])\, E$
3. sharing of closures, introducing the rule for commutation of closure contexts and which happens to be, with the simplified contexts, the usual (Assoc) rule: $(\lambda x.\, E[x])\, (\lambda y.\, A)\, t \to (\lambda y.\, (\lambda x.\, E[x])\, A)\, t \quad (\textsc{Assoc})$

Proposition 14. *The resulting calculus is precisely AF-calculus.*

Closure contexts as refined explicit substitutions It turns out that LS-calculus is essentially $\lambda_{\texttt{wls}}$ where closure contexts are collapsed to a flat list of binders, i.e. $\mathcal{L}$ contexts (once again, up to the use of an explicit `let` construction). The rules can be rephrased with the previous notations as follows.

$$N ::= [\cdot] \mid \mathcal{L}[N] \mid N\ t \mid (\lambda x.\, N[x])\ N$$
$$\mathcal{L}[\lambda x.\, t]\ u \to_{\lambda_{\texttt{lsc}}} \mathcal{L}[(\lambda x.\, t)\ u]$$
$$(\lambda x.\, N[x])\ \mathcal{L}[v] \to_{\lambda_{\texttt{lsc}}} \mathcal{L}[(\lambda x.\, N[v])\ v]$$

As one can witness, the first rule, known as the distant by-name β-rule, is just (Lift$\star$). The second rule corresponds to the dereferencing rule of $\lambda_{\texttt{wls}}$ without a closure context around the λ-abstraction because (Lift$\star$) builds up explicit substitutions by putting every abstraction in front of its corresponding argument.

We can easily translate any well-formed marked term into a term with explicit substitutions: it is indeed sufficient to collapse all closure contexts $\mathcal{C}$ into flat binding contexts $\mathcal{L}(\mathcal{C})$ inductively:

$$\mathcal{L}([\cdot]) := [\cdot] \qquad \mathcal{L}(\mathcal{C}_1[\ell x.\, \mathcal{C}_2]\ t) := \mathcal{L}(\mathcal{C}_1)[(\lambda x.\, \mathcal{L}(\mathcal{C}_2))\ t]$$

Up to this translation, which is no more than taking a normal form for σ-equivalence, LS-calculus and $\lambda_{\texttt{wls}}$ describe the same calculus. Our calculus incurs a small technical penalty because we need to restrict the set of terms w.r.t. well-formedness, while this comes for free in LS-calculus as explicit substitutions pertain to the syntax and force balancing of prime redexes.

Nonetheless, we advocate that $\lambda_{\texttt{wls}}$ is more fine-grained and that there are cases where this may matter. Indeed, Proposition 7 shows that closure contexts are faithful reifications of KAM environments. This is not the case for flat $\mathcal{L}$ contexts which may represent several KAM environments. This mismatch can be indeed observed in presence of side-effects sensitive to the structure of environments, most notably forcing [23], but probably linear effects as well. As we precisely want to extend call-by-need with effects, we claim that closure contexts should be preferred over flat context in this setting.

4 Classical By-need

To extend our classical LHR calculus to a fully-fledged call-by-need calculus, we follow the same three-step path that led us from LHR to call-by-need. We will not give the full details for the three steps though, and we will instead only give the final calculus.

The most delicate point is actually the introduction of weak reduction. In a classical setting, the actual applicative context of a variable may be strictly larger than it seems, because in commands of the form $[\alpha]\, t$, the α variable may be bound to a stack featuring supplementary applications. This means that we need to take into account supernumerary abstractions at the beginning of commands. Yet, the marking procedure allows us to remember which abstractions are actually paired with a corresponding application in a direct way.

We present in this section two variants of a classical by-need calculus, one effectively taking into account supernumerary arguments as described above, and a less smart variant that perfoms classical substitution upfront without caring for abstraction-application balancing on command boundaries. The advantage of the latter over the former is that it can be easily linked to a previous classical by-need calculus [10].

4.1 Classical By-need with Classical Closure Contexts

To implement the mechanism described above, we simply need to acknowledge the requirement to go through μ binders in our contexts. This leads to the mutual definition of classical-by-value contexts E^v and closure stack fragments K^v, where closure contexts are updated as well to handle classical binders.

$$\begin{aligned}
\mathcal{C} &::= [\cdot] \mid \mathcal{C}_1[\ell x.\, \mathcal{C}_2]\ t \mid \mathcal{C}_1[\mu\alpha.\, K^v[[\alpha]\, \mathcal{C}_2]] \\
E^v &::= [\cdot] \mid E^v\ t \mid \ell x.\, E^v \mid \mathcal{C}[\ell x.\, E^v_1[x]]\ E^v_2 \mid \mu\alpha.\, K^v[[\alpha]\, E^v] \\
K^v &::= [\cdot] \mid [\alpha]\, E^v[\mu\beta.\, K^v]
\end{aligned}$$

Definition 22 (Classical-by-need with classical closure contexts). *The classical-by-need calculus with classical closure contexts,* `cwls`*, is defined by the following reduction rules.*

$$\begin{aligned}
\mathcal{C}[\lambda x.\, t]\ u &\to_{\lambda_{\texttt{cwls}}} \mathcal{C}[\ell x.\, t]\ u \\
\mathcal{C}'[\ell x.\, E^v[x]]\ \mathcal{C}[v] &\to_{\lambda_{\texttt{cwls}}} \mathcal{C}[\mathcal{C}'[\ell x.\, E^v[v]]\ v] \\
\mathcal{C}'[\ell x.\, E^v[x]]\ \mathcal{C}[\mu\alpha.\, c] &\to_{\lambda_{\texttt{cwls}}} \mathcal{C}[\mu\alpha.\, c\{\alpha := [\alpha]\, \mathcal{C}'[\ell x.\, E^v[x]]\ \_\}]
\end{aligned}$$

The issue of this calculus is that the delayed classical substitution is a quite novel phenomenon that does not ressemble anything from the literature as far as we know. It is, in particular, difficult to compare with previous attempts at a call-by-need variant of the $\lambda\mu$-calculus.

4.2 Classical By-need with Intuitionistic Closure Contexts

We describe here a modification of the above calculus whose laziness has been watered down. Instead of delaying classical substitutions, it performs them as soon as possible. The main difference with the smart calculus lies in the fact that closure contexts remain intuitionistic and do not allow to go down under μ-binders. The corresponding contexts are inductively defined as follows.

$$\begin{aligned}
\mathcal{C} &::= [\cdot] \mid \mathcal{C}_1[\ell x.\, \mathcal{C}_2]\ t \\
E^v &::= [\cdot] \mid E^v\ t \mid \ell x.\, E^v \mid \mathcal{C}[\ell x.\, E^v_1[x]]\ E^v_2 \\
K^v &::= [\cdot] \mid [\alpha]\, E^v[\mu\beta.\, K^v]
\end{aligned}$$

Definition 23 (Classical-by-need with intuitionistic closure contexts). *The classical-by-need calculus with intuitionistic closure contexts,* `cwls`$'$*, is defined by the following reduction rules.*

$$\begin{aligned}
\mathcal{C}[\lambda x.\, t]\ u &\to_{\lambda_{\texttt{cwls}'}} \mathcal{C}[\ell x.\, t]\ u \\
\mathcal{C}'[\ell x.\, E^v[x]]\ \mathcal{C}[v] &\to_{\lambda_{\texttt{cwls}'}} \mathcal{C}[\mathcal{C}'[\ell x.\, E^v[v]]\ v] \\
[\alpha]\, E^v[\mu\beta.\, K^v[[\beta]\, t]] &\to_{\lambda_{\texttt{cwls}'}} [\alpha]\, E^v[\mu\beta.\, K^v[[\alpha]\, E^v[t]]]
\end{aligned}$$

4.3 Comparison With Existing Classical Call-by-need Calculus

In order to better understand the calculi of the previous section, we now turn to the comparison with another classical-by-need calculus [10], referred to as AHS, which is obtained from a calculus derived from Curien and Herbelin's duality of computation. As a consequence, AHS features plain call-by-value β reduction and not a linear, non-destructive, deref rule à la Ariola-Felleisen:

$$(\lambda x.\, t)\ v \to t\{x := v\} \text{ with } v := x \mid \lambda x.\, t$$

Comparing precisely our calculi with AHS is tricky because AHS is built on destructive substitution which additionally is plain β_v. The reason for such a presentation of AHS is to be found in its sequent calculus origin [10]. As we did for the comparison with Chang-Felleisen calculus, we will consider a variant of AHS with a deref rule à la Ariola-Felleisen described, in sequent style, in the last section of [7] for which we will prove that established `cwls`-reduction is sound and complete. We conjecture that the same result holds for AHS but do not yet have a proof of this fact.

AHS modified calculus We now consider a slightly modified version of the previous calculus from [10]. AHS'-calculus consists in AHS-calculus where the beta reduction has been replaced by a deref rule à la Ariola-Felleisen (where variables are not values) and the notion of evaluation context has been adapted accordingly. This calculus has been first described, in sequent style, in [7].

Definition 24 (Ariola-Herbelin-Saurin modified calculus). *AHS' reduction for the $\lambda\mu$-calculus is defined below.*

$$\begin{array}{lll}
 & (\lambda x.\, t)\ u\ r & \to (\lambda x.\, t\ r)\ u \\
v \ := \lambda x.\, t & (\lambda x.\, C[x])\ v & \to (\lambda x.\, C[v])\ v \\
n \ := x \mid t\ u \mid \mu\alpha.\, c & (\lambda z.\, C[z])\ ((\lambda x.\, t)\ u) & \to (\lambda x.\, (\lambda z.\, C[z])\ t)\ u \\
E ::= [\cdot] \mid E\ t & (\mu\alpha.\, c)\ t & \to \mu\alpha.\, c\{[\alpha]\, r := [\alpha]\, r\ t\} \\
C ::= E \mid (\lambda z.\, C)\ t \mid & (\lambda x.\, C[x])\ (\mu\alpha.\, c) & \to \mu\alpha.\, c\{[\alpha]\, r := [\alpha]\, (\lambda x.\, C[x])\ r\} \\
\qquad (\lambda x.\, C[x])\ E & (\lambda x.\, \mu\alpha.\, [\beta]\, t)\ n & \to \mu\alpha.\, [\beta]\, (\lambda x.\, t)\ n \\
 & [\alpha]\, \mu\beta.\, c & \to c\{\beta := \alpha\}
\end{array}$$

Theorem 5. *For any command c, there exists an infinite standard reduction in AHS'-calculus starting from c iff there exists an infinite reduction starting from c in* `cwls`$'$*-calculus.*

Proof. We show this by giving the sketch of a pair of simulation theorems. We will separate AHS' reduction rules in three groups:

- The structural rules (S) which are made of the Lift and Assoc rules, together with the rule $(\lambda x.\, \mu\alpha.\, [\beta]\, t)\ n \to \mu\alpha.\, [\beta]\, (\lambda x.\, t)\ n$.
- The performing rules (P) which is only the dereferencing rule.
- The classical rules (C) which are the three remaining rules.

Transforming reductions in $\texttt{cwls}'$-calculus into AHS' is straightforward. First, assuming a closure stack fragment K_v, one can see that AHS' will normalize it into a delimited C context in the following way. For each splice of K_v of the form $[\alpha]\, E_v[\mu\beta.\,[\cdot]]$, the E_v context will be simplified by a series of applications of the (S) rules. According to the form of K_v, either the reduction stops (if there is no remaining splice) or it performs a certain number of (C) rules, until which the normalization procedure recursively applies. A dereferencing cannot occur at this point because while there are remaining splices, the current needed context cannot contain variables, as the splices all have a μ binder in needed position. Note that the resulting normalized context is still a closure stack fragment up to unmarking. By a simple size argument, this normalization procedure must terminate, so that we will consider it transparent for the simulation.

Now, assume that

$$\mathcal{C}_1[\ell x.\, E_v[x]]\ \mathcal{C}_2[v] \to \mathcal{C}_2[\mathcal{C}_1[\ell x.\, E_v[v]]\ v]$$

with the associated conditions for this rule. The (S) rules apply to $\mathcal{C}_1$ and $\mathcal{C}_2$. This effectively transforms them into answer contexts, so that a derefencing rule can occur after the E_v has been flattened as well. The important thing to observe is that the same normalization steps apply to the reduct $\mathcal{C}_2[\mathcal{C}_1[\ell x.\, E_v[v]]\ v]$ so that each step from $\texttt{cwls}'$ is going to be matched by a growing but finite quantity of normalization steps followed by a derefencing.

The second reduction rule, corresponding to stack substitutions, is actually directly handled by the normalization procedure described just above.

We turn to the simulation of the AHS' reduction by the $\texttt{cwls}'$-calculus. First, the standard reduction contexts are a degenerated case of K_v contexts with one splice and flattened closure contexts, which allows to easily transfer rules from the source to the target. We actually match each class of reduction (S), (P) and (C) to a given behaviour in the target calculus.

- The (S) rules are transparent for the $\texttt{cwls}'$-calculus, because they are natively handled by closure contexts. So a (S)-reduction does not give rise to a λ_{clh}-reduction.
- A group of (C) rules can be matched by an arbitrary number of reductions, including none. This depends on the way the corresponding stack variable is used.
- The (P) rule is conversely matched by exactly one rule in the $\texttt{cwls}'$ reduction.

The trick is to use the fact that $(C+S)$ is normalizing, as we already did in the previous case. Moreover, such reductions do not change the possibility to perform a derefencing in the corresponding $\texttt{cwls}'$ term. So we actually consider groups of reductions $(C+S)^*, P$ in the source calculus. This is always possible to decompose a sequence of AHS' reductions as such thanks to the normalization of $(C+S)$. It it then easy to witness that the S part will have no effect, each C reduction will be matched by a finite number of context reductions in λ_{clh}, and that the final (P) will correspond to exactly one derefencing reduction in λ_{clh}.

5 Conclusion

Nevertheless, the early history of continuations is a sharp reminder that original ideas are rarely born in full generality, and that their communication is not always a simple or straightforward task.

John C. Reynolds [28]

Reaching this point of the paper, we hope to have convinced the reader, thanks to the above developments and results, that linear head reduction can be a useful tool in order to develop a classy call-by-need with control.

Contributions. After reformulating linear head reduction in a way which is somehow intermediate between traditional LHR and Accattoli *et al.*'s approach via explicit substitutions, we demonstrated the deep connections between LHR and call-by-need by showing that weak call-by-value LHR with sharing *is* a call-by-need calculus. We then strengthened this result to the case of $\lambda\mu$-calculus, that is we introduced a call-by-need calculus with control operator obtained from classical LHR, another contribution of this paper. We obtained such a calculus by a systematic derivation from LHR in three steps: (i) by restricting it to a weak reduction, (ii) by ensuring substitution of values, and (iii) by sharing closures. Closure contexts play a central role in LHR: all the structures we consider are actually closed by this construction, making them a methodological feature of our approach. While they are not novel in the intuitionistic case, it is the first time they are put in use in an explicit and articulated way, not to speak of the classical extension of closure contexts which is properly a contribution of this work. We validated our development in two ways:

- our approach to linear head reduction (using closure contexts and, to deal with weak reduction, marked terms) is validated by the fact that it transfers smoothly to $\lambda\mu$-calculus, a new result of this paper. Additionally, one can see the use of closure contexts, in particular when dealing with marked terms, as a generalization of LSC where the structure of substitutions is encoded in closure contexts, a tree-like structure, and not in substitution contexts, which are linearized: we keep closer to the original structure of the term which is important when dealing with computational effects.
- The call-by-need λ and $\lambda\mu$-calculi that we obtain in the paper are related with previously known versions of call-by-need calculi. In the case of the λ-calculus, they are related with Ariola-Felleisen's and Chang-Felleisen's calculi. In the case of $\lambda\mu$-calculi, two classical by-need calculi are actually proposed, one being related to a variant of Ariola-Herbelin-Saurin's calculus, and the other calling for further investigations.

Related Works. The most closely related works are certainly the works of Ariola *et al.* [7,10] and Accattoli *et al.* [2]. Their relations with the present work have been discussed throughout the paper. Summing up:

– we developed a different methodology from that of Ariola *et al.* in that we stayed within the framework of natural deduction and analyzed the systematic synthesis of call-by-need from LHR in a careful way resulting in the ability to lift this to the $\lambda\mu$-calculus.
– Compared with LSC, our approach can be viewed as less specified and somehow more general than that of LSC, which obviously prevents us from having results as precise as those of LSC, for instance regarding complexity analysis. On the other hand, maintaining the structure of λ-terms suggests interesting perspectives for handling various computational effects.

Future Work. We finally describe some perspectives of future work.
More computational effects. The robustness of our approach is encouraging for testing other computational effects where maintaining the term structure may be even more crucial than for control effects.
Classical by-need. The comparisons with other proposals for classical variants of the call-by-need reduction [10,7] remain to be established more precisely. We conjecture that our calculus is sound and complete not only with AHS' but also with AHS-calculus [10]. Moreover, the classical-by-need calculus with classical closure contexts remains difficult to connect to already known calculi.
Towards full laziness. Our design guided by σ-equivalence can do more than call-by-need and can actually already encompass a weak form of full laziness. Future work will pursue this direction.
Reduction strategies versus calculi. The original motivation of our work was to relate formally LHR and call-by-need. As a result, instead of focusing on proper calculi we concentrated our attention on a specific evaluation strategy for several reasons: macroscopic and weak reductions are more naturally expressed with strategies. However, the λ_{lh}-calculus can very easily be studied as a calculus and we shall develop it as such in the future.

Acknowledgements. The authors would like to thank Beniamino Accattoli, Thibaut Balabonski, Olivier Danvy and Delia Kesner for discussions regarding this work as well as anonymous reviewers.

References

1. M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *J. Funct. Program.*, 1(4):375–416, 1991.
2. B. Accattoli, P. Barenbaum, and D. Mazza. Distilling abstract machines. In Jeuring and Chakravarty, editors, *ICFP 2014*, pages 363–376. ACM, 2014.
3. B. Accattoli, E. Bonelli, D. Kesner, and C. Lombardi. A nonstandard standardization theorem. In *POPL '14, San Diego, CA, USA*, pages 659–670, 2014.
4. B. Accattoli and D. Kesner. The structural lambda-calculus. In *CSL 2010*, volume 6247 of *LNCS*, pages 381–395. Springer, 2010.
5. B. Accattoli and D. Kesner. The permutative lambda calculus. In *LPAR-18*, Merida, Venezuela, Mar. 2012.
6. B. Accattoli and U. D. Lago. Beta Reduction is Invariant, Indeed. In *CSL–LICS 2014*, Vienna, Austria, July 2014.

7. Z. M. Ariola, P. Downen, H. Herbelin, K. Nakata, and A. Saurin. Classical call-by-need sequent calculi: The unity of semantic artifacts. In *FLOPS 2012*, volume 7294 of *LNCS*, pages 32–46. Springer, 2012.
8. Z. M. Ariola and M. Felleisen. The call-by-need lambda calculus. *Journal of Functional Programming*, 7(3):265–301, 1997.
9. Z. M. Ariola, M. Felleisen, J. Maraist, M. Odersky, and P. Wadler. The call-by-need lambda calculus. In *POPL 1995*, pages 233–246. ACM Press, 1995.
10. Z. M. Ariola, H. Herbelin, and A. Saurin. Classical call-by-need and duality. In *TLCA*, volume 6690 of *LNCS*. Springer, 2011.
11. H. Barendregt. *The Lambda Calculus, its Syntax and Semantics*. Studies in Logic and the Foundations of Mathematics. Elsevier, 1984.
12. S. Chang and M. Felleisen. The call-by-need lambda calculus, revisited. In *ESOP 2012*, LNCS. Springer, 2012.
13. P.-L. Curien and H. Herbelin. The duality of computation. In M. Odersky and P. Wadler, editors, *ICFP 2000*, pages 233–243. ACM Press, 2000.
14. V. Danos, H. Herbelin, and L. Regnier. Game semantics & abstract machines. In *LICS 1996*, pages 394–405. IEEE Press, 1996.
15. V. Danos and L. Regnier. Head linear reduction. unpublished, 2004.
16. O. Danvy, K. Millikin, J. Munk, and I. Zerny. Defunctionalized interpreters for call-by-need evaluation. In *FLOPS 2010*, LNCS. Springer, 2010.
17. J.-Y. Girard. Une extension de l'interprétation de Gödel à l'analyse, et son application a l'élimination des coupures dans l'analyse et la théorie des types. In Fenstad, editor, *Proc. of the Second Scandinavian Logic Symposium*, volume 63 of *Studies in Logic and the Foundations of Mathematics*, pages 63 – 92. Elsevier, 1971.
18. J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
19. M. Hyland and L. Ong. On full abstraction for PCF. *Information and Computation*, 163(2):285–408, Dec. 2000.
20. J.-L. Krivine. A call-by-name lambda-calculus machine. *Higher-Order and Symbolic Computation*, 20(3):199–207, 2007.
21. O. Laurent. *A study of polarization in logic*. PhD thesis, Université de la Méditerranée - Aix-Marseille II, Mar. 2002.
22. J. Maraist, M. Odersky, and P. Wadler. The call-by-need lambda-calculus. *Journal of Functional Programming*, 8(3):275–317, 1998.
23. A. Miquel. Forcing as a program transformation. In *LICS*, pages 197–206. IEEE Computer Society, 2011.
24. M. Parigot. Lambda-mu-calculus: An algorithmic interpretation of classical natural deduction. In *LPAR*, volume 624 of *LNCS*, pages 190–201. Springer, 1992.
25. L. Regnier. *Lambda-calcul et réseaux*. PhD thesis, Univ. Paris VII, 1992.
26. L. Regnier. Une équivalence sur les λ-termes. *Theoret. Comput. Sci.*, 126:281–292, 1994.
27. J. C. Reynolds. Towards a theory of type structure. In B. Robinet, editor, *Programming Symposium*, volume 19 of *LNCS*, pages 408–423. Springer, 1974.
28. J. C. Reynolds. The discoveries of continuations. *Lisp and Symbolic Computation*, 6(3-4):233–248, 1993.
29. T. Streicher and B. Reus. Classical logic, continuation semantics and abstract machines. *J. Funct. Program.*, 8(6):543–572, 1998.
30. C. P. Wadsworth. *Semantics and pragmatics of the lambda-calculus*. PhD thesis, Programming Research Group, Oxford University, 1971.